Usi (meno scontati) della visita DFS

lezione basata sul capito 3 del libro
Algorithms, di Dasgupta, Papadimitriou, Vazirani,
McGraw-Hill

Informazioni utili: tenere il tempo

procedura `visitaDFSRicorsiva`$(vertice\ v, albero\ T)$

marca e visita il vertice v

pre(v)=clock
clock=clock+1

for each (arco (v, w)) **do**

  if (w non è marcato) **then**

    aggiungi l'arco (v, w) all'albero T

    `visitaDFSRicorsiva`(w, T)

post(v)=clock; clock=clock+1

algoritmo `visitaDFS`$(vertice\ s) \rightarrow albero$

$T \leftarrow$ albero vuoto

clock=1

`visitaDFSRicorsiva`(s, T)

return T

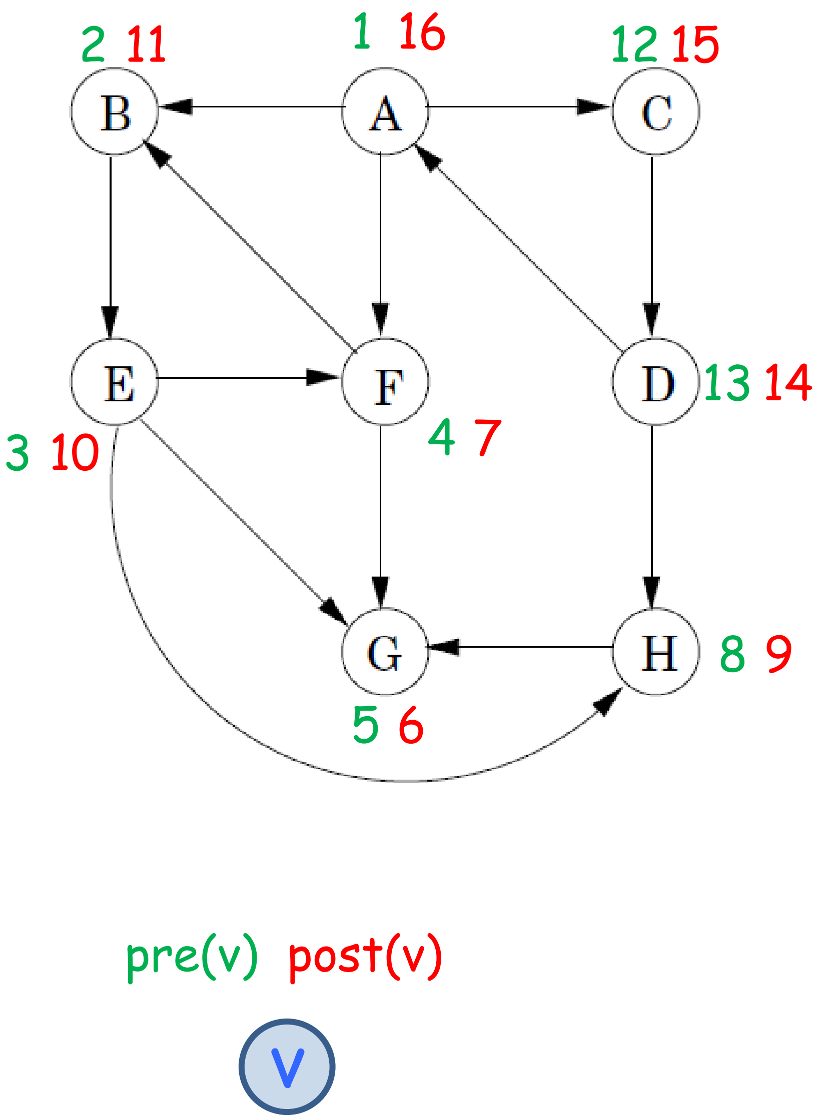
pre(v): tempo in cui viene "scoperto" v
post(v): tempo in cui si "abbandona" v

quando non tutti i nodi sono raggiungibili dal punto di partenza

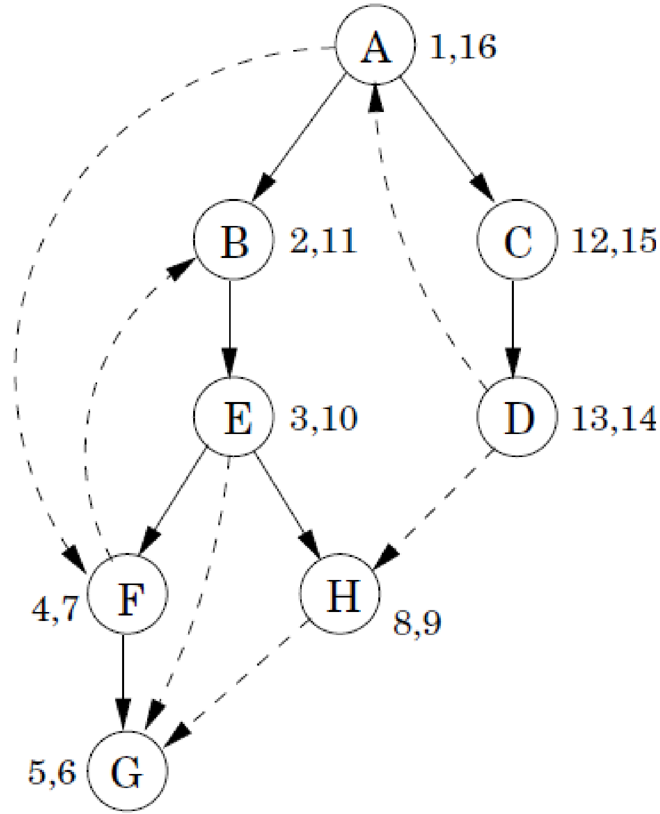
VisitaDFS (grafo *G*)

for each nodo *v* **do** imposta *v* come *non marcato*
clock=1
F ← foresta vuota
for each nodo *v* **do**
    if (*v* è *non marcato*) **then**
        T ← albero vuoto
        visitaDSFRicorsiva(*v*,*T*)
        aggiungi *T* ad *F*
return *F*

Un esempio



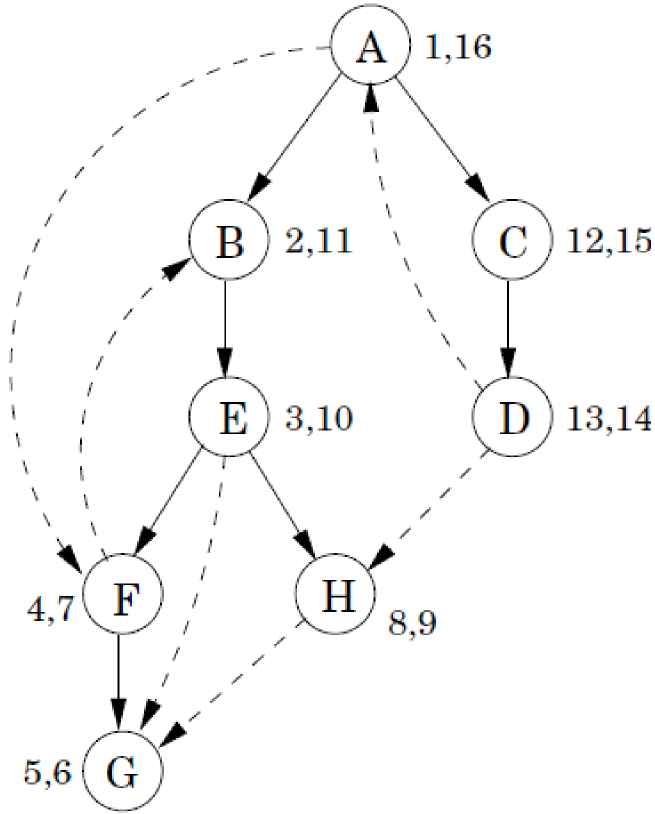
2 11
1 16
12 15
B
A
C
E
F
D
13 14
4 7
3 10
G
H
8 9
5 6
pre(v) post(v)
v



A
1,16
B
2,11
C
12,15
E
3,10
D
13,14
4,7
F
H
8,9
5,6
G

proprietà

per ogni coppia di nodi u e v, gli intervalli [pre(u),post(u)] e [pre(v),post(v)] o sono disgiunti o l'uno è contenuto nell'altro

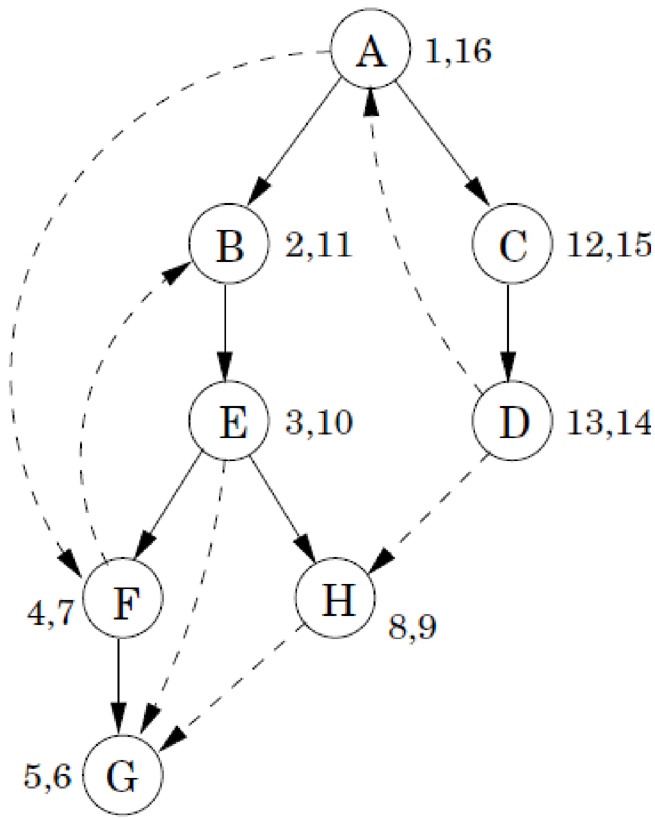


u è antenato di v nell'albero DFS, se pre(u) < pre(v) < post(v) < post(u) condizione che rappresentiamo così:

possiamo usare i tempi di visita per riconoscere il tipo di un generico arco (u,v) del grafo?

...riconoscere i tipi di arco

pre/post per l'arco (u,v)	tipo di arco
$[_u \quad [_v \quad]_v \quad]_u$	in avanti
$[_v \quad [_u \quad]_u \quad]_v$	all'indietro
$[_v \quad]_v \quad [_u \quad]_u$	trasversali



cicli, DAG e ordinamenti topologici

riconoscere la presenza di un ciclo in un grafo diretto

Algoritmo:
fai una visita DFS e controlla
se c'è un arco all'indietro

Proprietà

> Un grafo diretto *G* ha un ciclo se e solo se la visita DFS rivela un arco all'indietro.

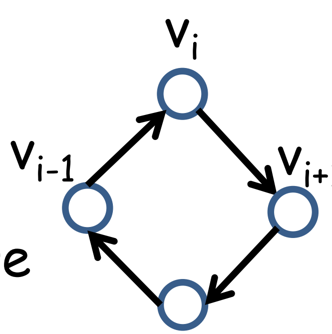
($\Leftarrow$): se c'è arco all'indietro, chiaramente *G* ha un ciclo

($\Rightarrow$): se c'è ciclo $\langle v_0, v_1, \dots, v_k = v_0 \rangle$

sia v_i è il primo nodo scoperto nella visita

poiché v_{i-1} è raggiungibile da v_i, visito v_{i-1} prima di terminare la visita di v_i

allora (v_{i-1}, v_i) è un arco all'indietro

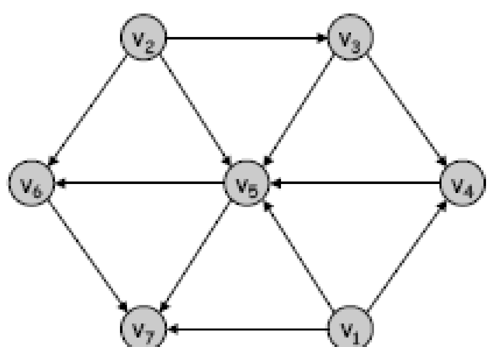


Definizione

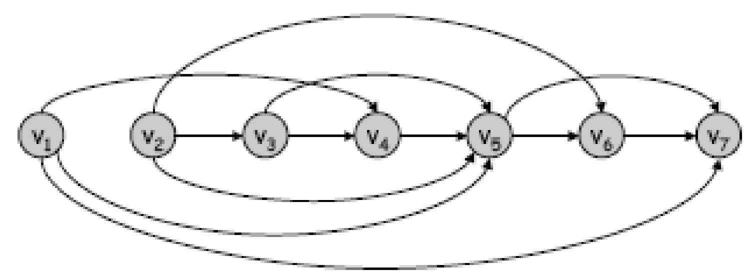
Un grafo diretto aciclico (DAG) è un grafo diretto G che non contiene cicli (diretti).

Definizione

Un ordinamento topologico di un grafo diretto $G=(V,E)$ è una funzione biettiva $\sigma:V \rightarrow \{1,2,..,n\}$ tale che per ogni arco $(u,v) \in E$, $\sigma(u)<\sigma(v)$



a DAG

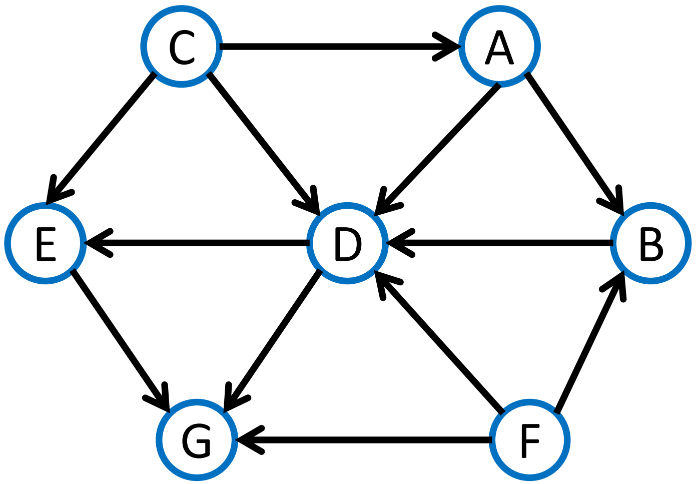


a topological ordering

pozzo: solo archi entranti

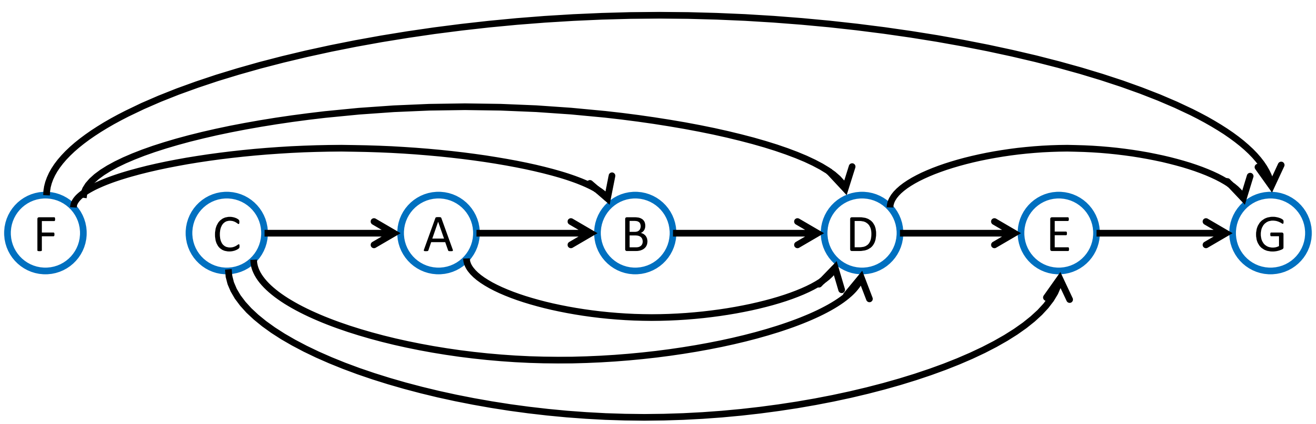
sorgente: solo archi uscenti

reti "delle dipendenze"



nodi: compiti da svolgere
arco (u,v): u deve essere eseguito prima di v

problema:
trovare un ordine in cui eseguire i compiti in modo da rispettare le dipendenze



quali grafi (diretti) ammettono un ordinamento topologico?

Teorema

Un grafo diretto G ammette un ordinamento topologico se e solo se G è un DAG

dim

($\Rightarrow$)

per assurdo: sia σ un ordinamento topologico di G

e sia $\langle v_0, v_1, \ldots, v_k = v_0 \rangle$ un ciclo

allora $\sigma(v_0) < \sigma(v_1) < \ldots < \sigma(v_{k-1}) < \sigma(v_k) = \sigma(v_0)$

($\Leftarrow$): …adesso diamo un algoritmo costruttivo.

calcolare ordinamento topologico

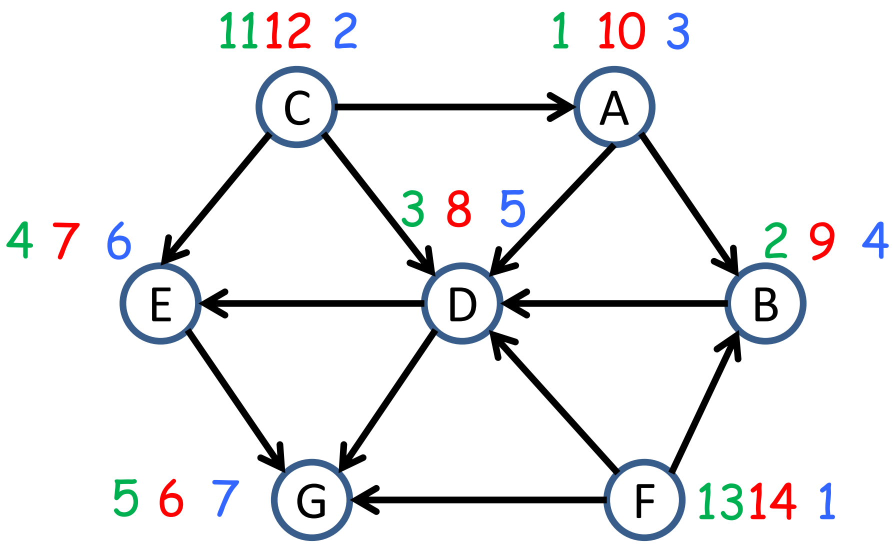
Algoritmo:
fai una visita DFS e restituisci i nodi in ordine decrescente rispetto ai tempi di fine visita post(v)

Complessità temporale:
se *G* è rappresentato con liste di adiacenza
$\Theta(n+m)$

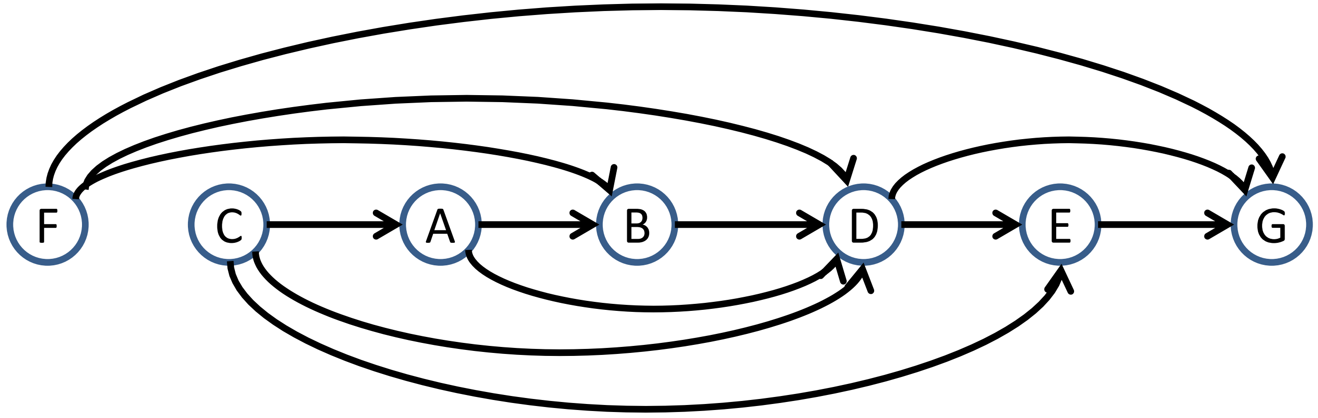
OrdinamentoTopologico (grafo *G*)

1. top=n; *L* $\leftarrow$ lista vuota;
2. chiama visita DFS ma:
   1. quando hai finito di visitare un nodo *v* (quando imposti post(v)):
   2. σ(v)=top; top=top-1;
   3. aggiungi v in testa alla lista L
3. **return** *L* e σ

Un esempio



pre(v) post(v) σ(v)



correttezza

per ogni coppia di nodi u e v, gli intervalli [pre(u),post(u)] e [pre(v),post(v)] o sono disgiunti o l'uno è contenuto nell'altro

pre/post per l'arco (u,v)	tipo di arco
$\underset{u}{[}\ \underset{v}{[}\ \underset{v}{]}\ \underset{u}{]}$	in avanti
$\underset{v}{[}\ \underset{u}{[}\ \underset{u}{]}\ \underset{v}{]}$	~~all'indietro~~
$\underset{v}{[}\ \underset{v}{]}\ \underset{u}{[}\ \underset{u}{]}$	trasversali

non ci possono essere archi all'indietro

Un algoritmo alternativo

algoritmo `ordinamentoTopologico`$(grafo\, G) \rightarrow lista$

$\widehat{G} \leftarrow G$

$ord \leftarrow$ lista vuota di vertici

while (esiste un vertice u senza archi entranti in $\widehat{G}$) **do**

  appendi u come ultimo elemento di ord

  rimuovi da $\widehat{G}$ il vertice u e tutti i suoi archi uscenti

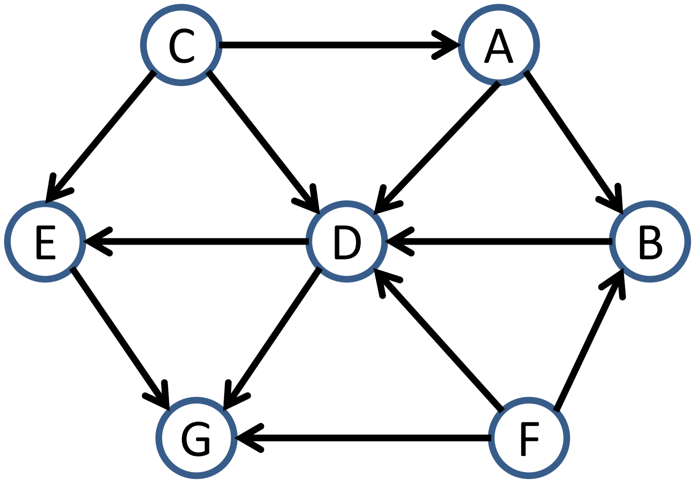
(*) **if** ($\widehat{G}$ non è diventato vuoto) **then errore** il grafo G non è aciclico

return ord

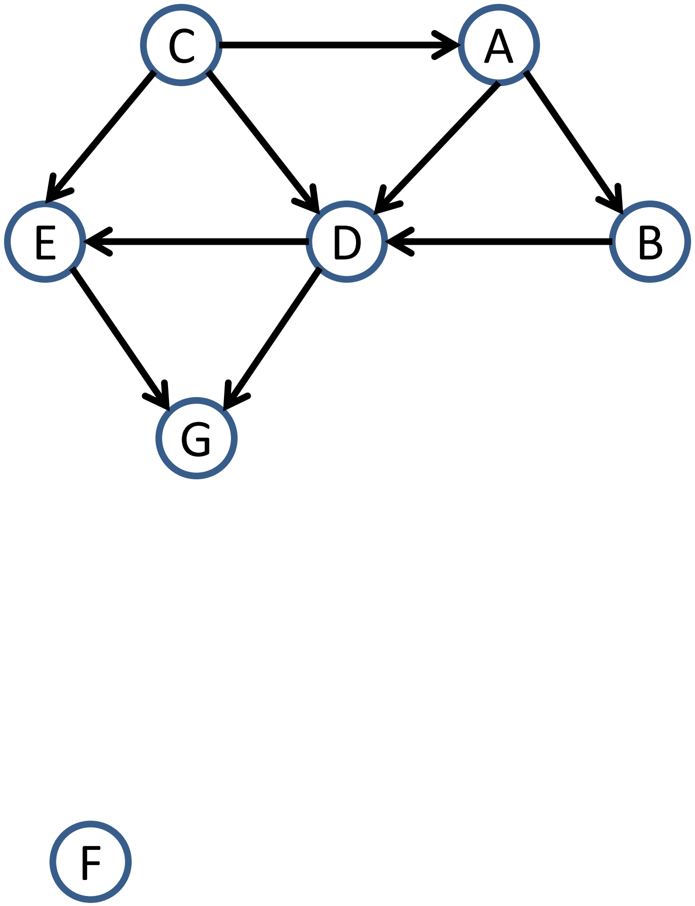
(*) perché altrimenti in $\hat{G}$ ogni vertice deve avere almeno un arco entrante, e quindi posso trovare un ciclo percorrendo archi entranti a ritroso, e quindi G non può essere aciclico)

Tempo di esecuzione (con liste di adiacenza): $\Theta(n+m)$ (dimostrare!)

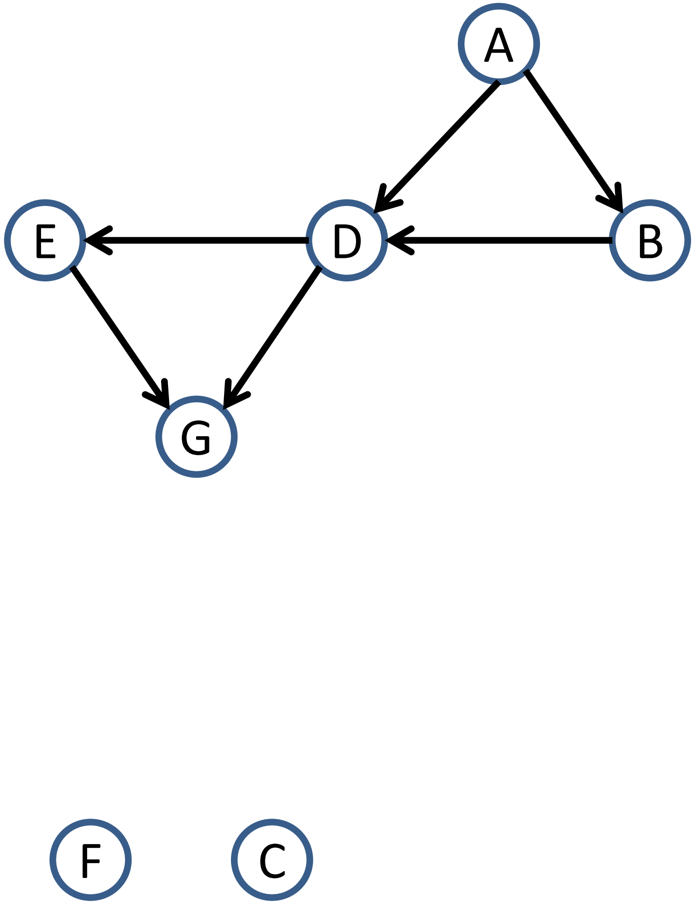
Un esempio



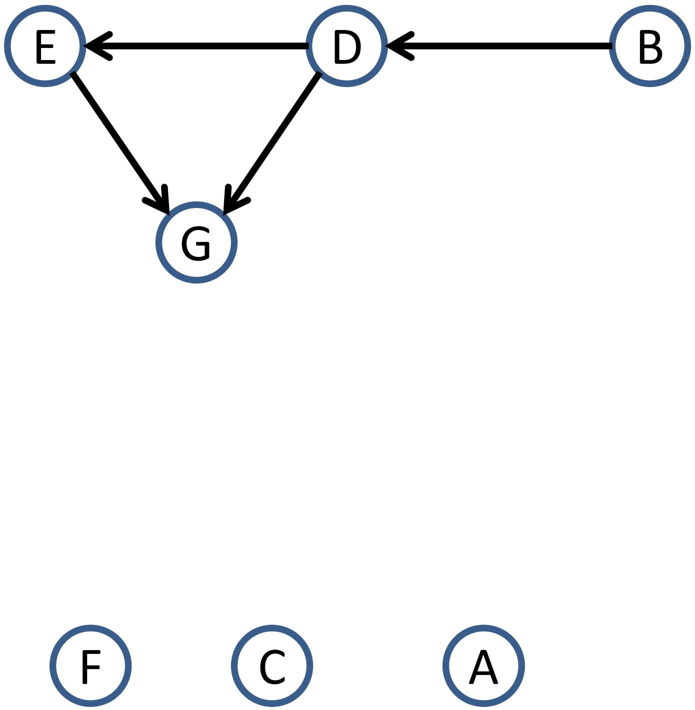
Un esempio



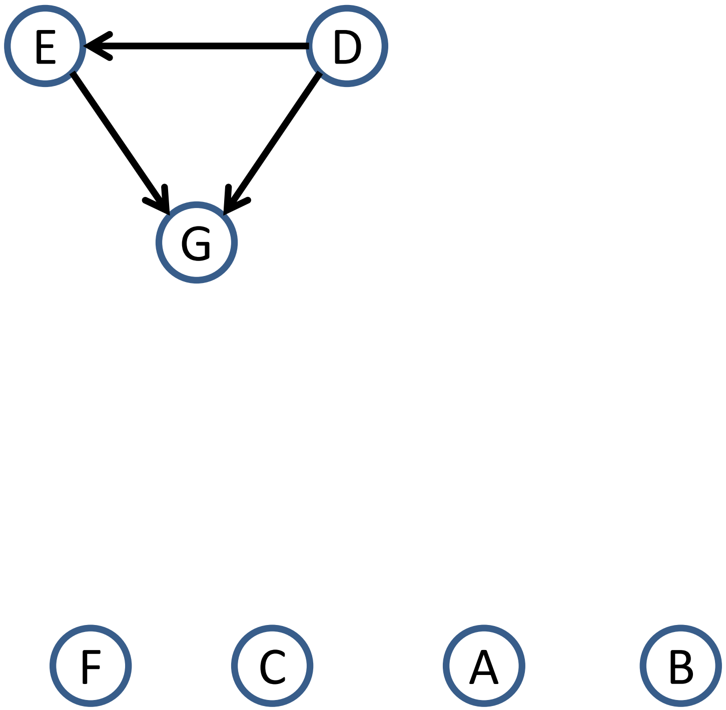
Un esempio



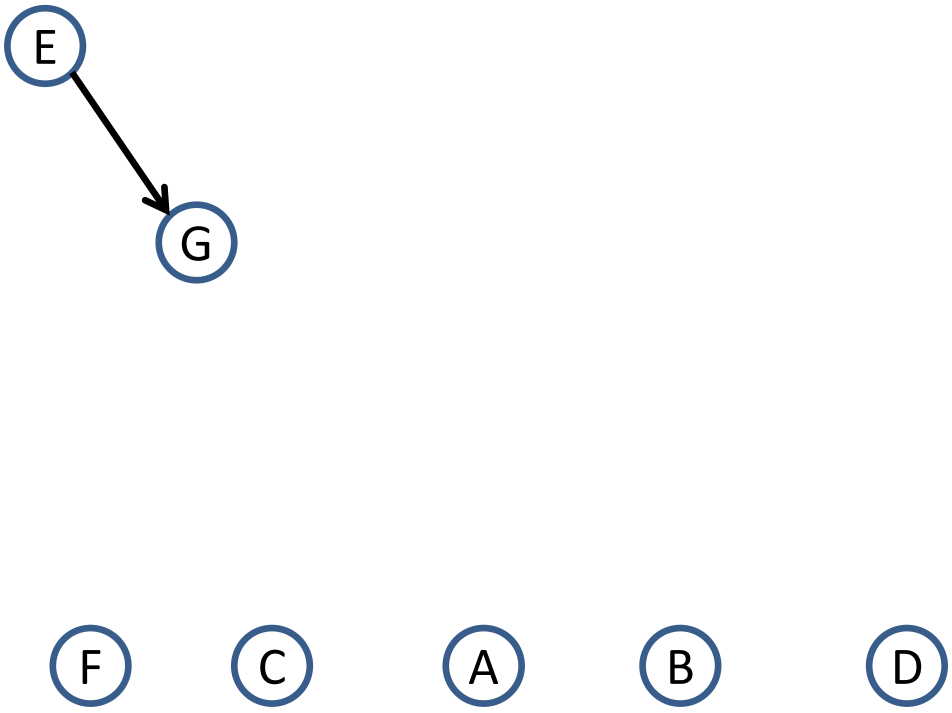
Un esempio



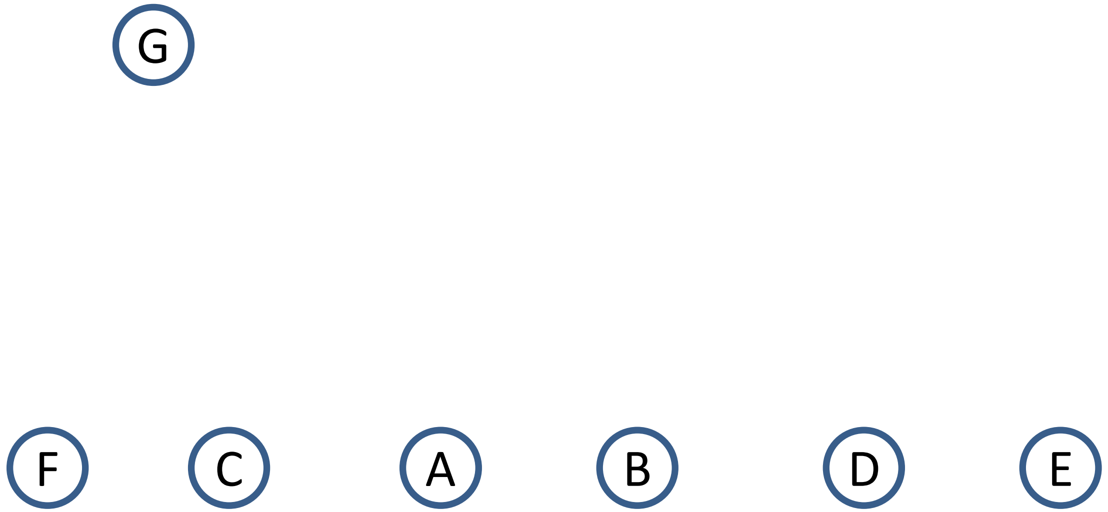
Un esempio



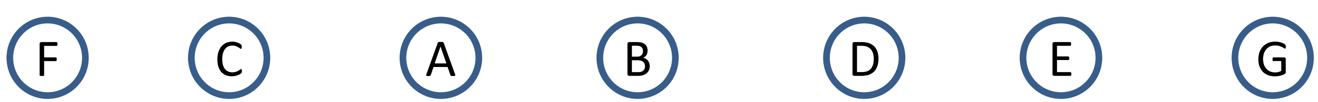
Un esempio



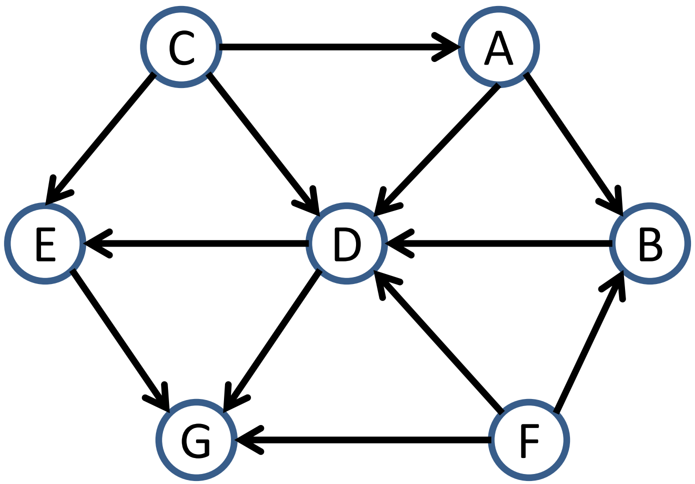
Un esempio

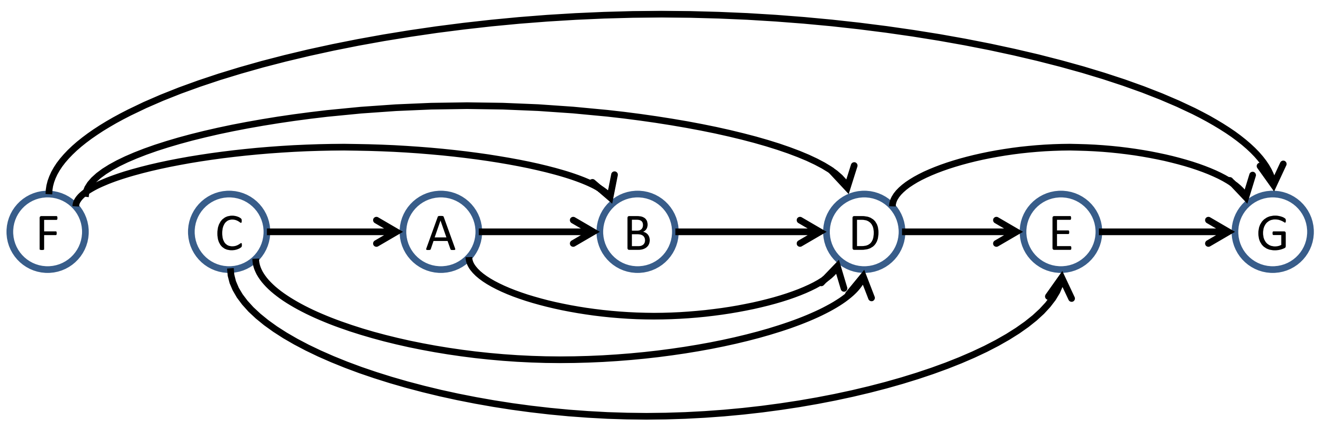


Un esempio

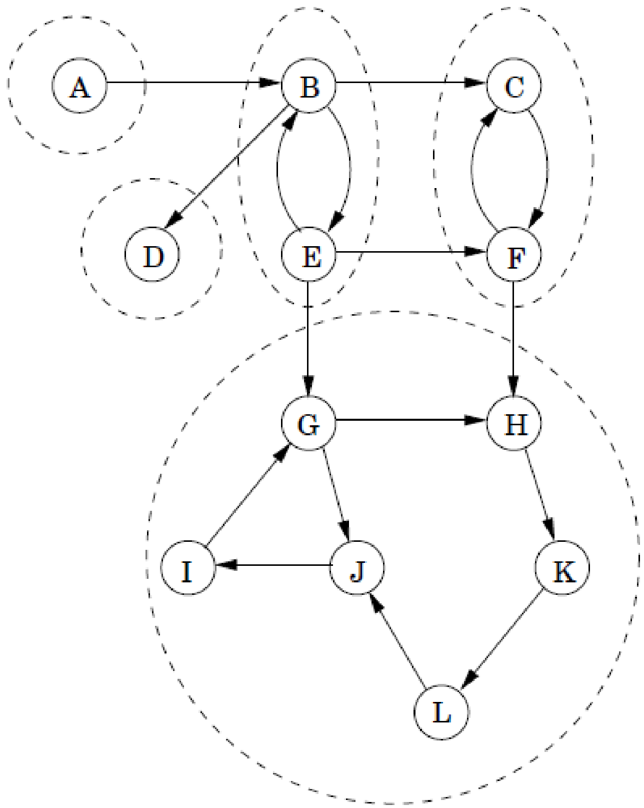


Un esempio



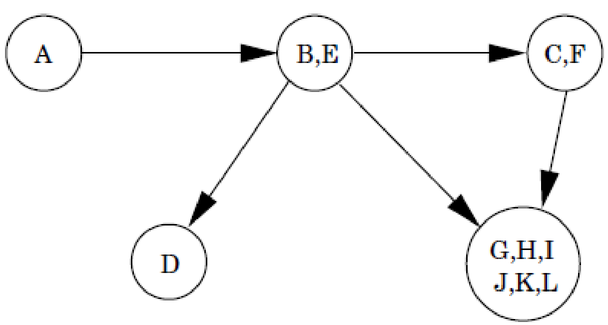


componenti fortemente connesse



una componente fortemente connessa di un grafo $G=(V,E)$ è un insieme massimale di vertici $C \subseteq V$ tale che per ogni coppia di nodi u e v in C, u è raggiungibile da v e v è raggiungibile da u

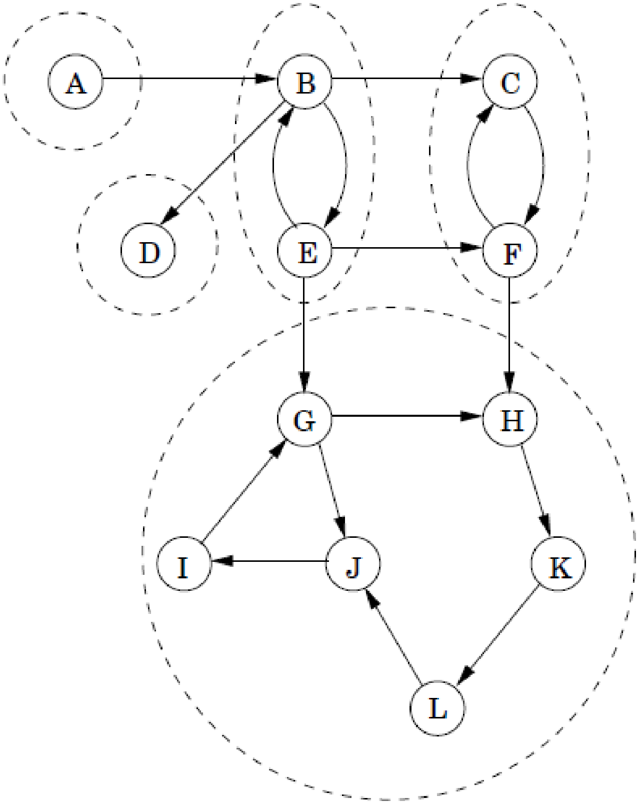
massimale: se si aggiunge un qualsiasi vertice a C la proprietà non è più vera



grafo delle componenti fortemente connesse di G

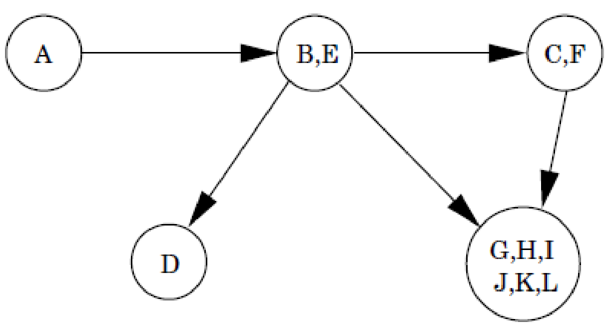
è sempre un DAG!

come si possono calcolare le componenti fortemente connesse di un grafo diretto?

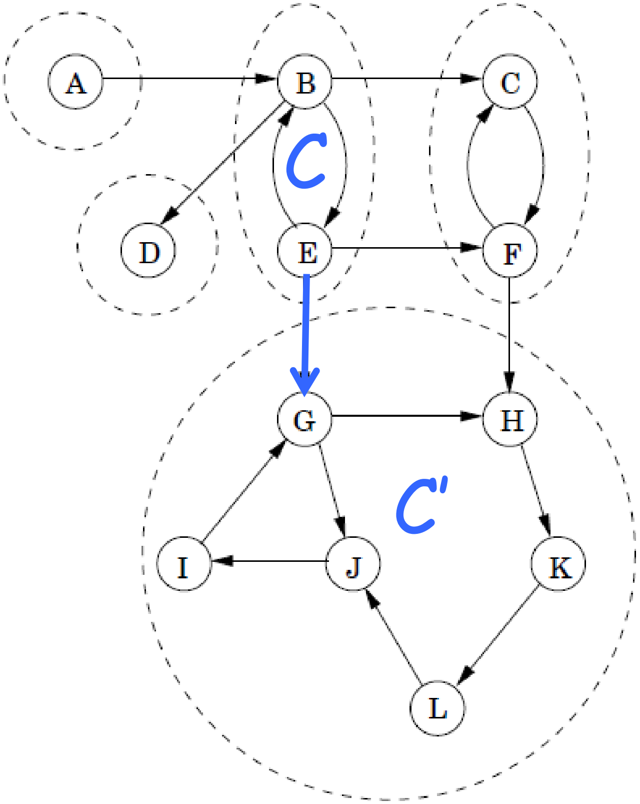


Proprietà 1: se si esegue la procedura visitaDFSricorsiva a partire da un nodo u la procedura termina dopo che tutti i nodi raggiungibili da u sono stati visitati

Idea: eseguire una visita a partire da un nodo di una componente *pozzo*, "eliminare" la componente e ripetere

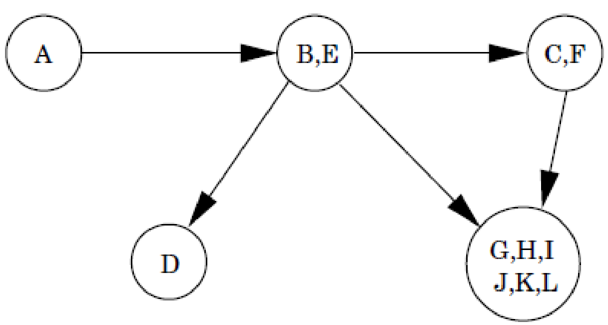


come trovo una componente pozzo?



Proprietà 2: se C e C' sono due componenti e c'è un arco da un nodo in C verso uno in C', allora il più grande valore post() in C è maggiore del più alto valore di post() di C'

dim: se la DFS visita prima C' di C: banale. se visita prima C, allora si ferma dopo che ha raggiunto tutti i nodi di C e C' e termina su un nodo di C.

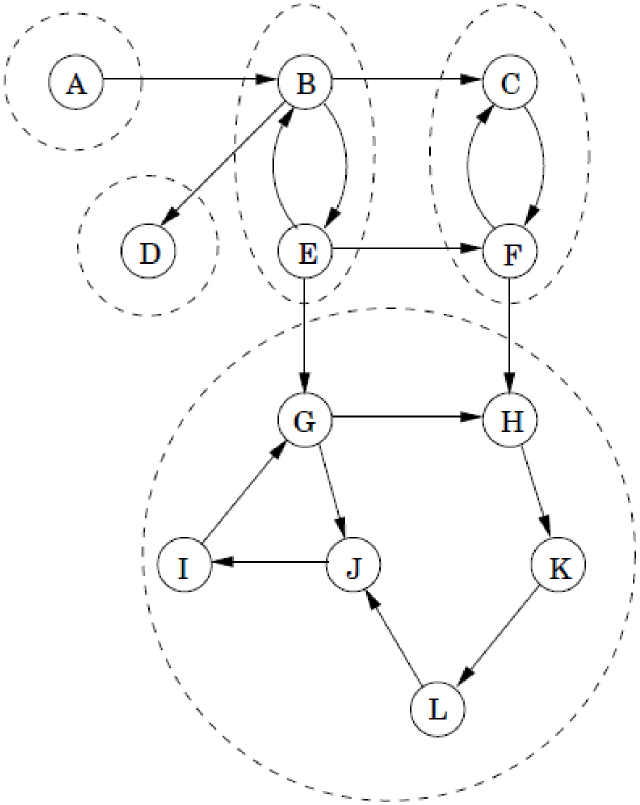


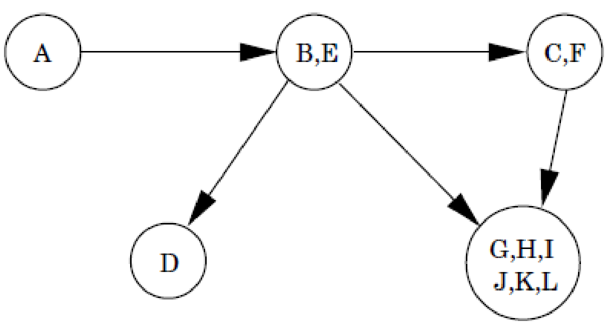
Proprietà 3: il nodo che riceve da una visita DFS il valore più grande di post() appartiene a una componente *sorgente*

ma avevamo bisogno di una componente *pozzo*?

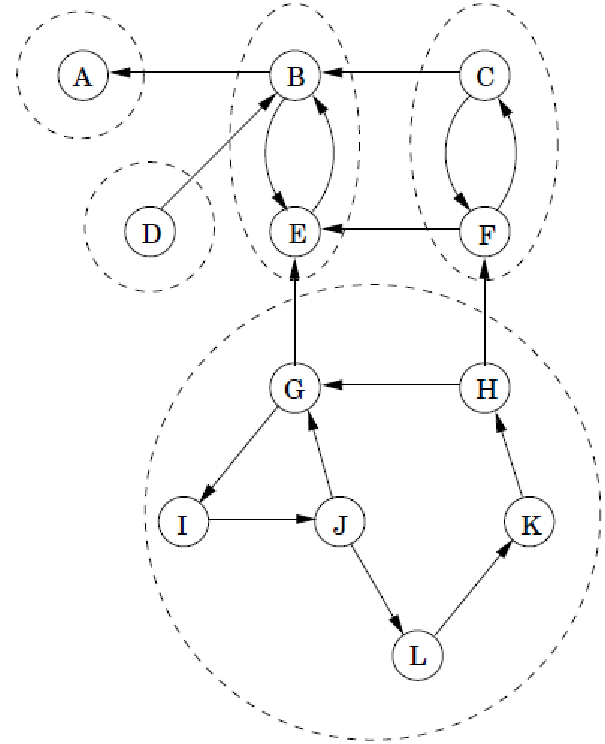
idea: invertiamo gli archi!

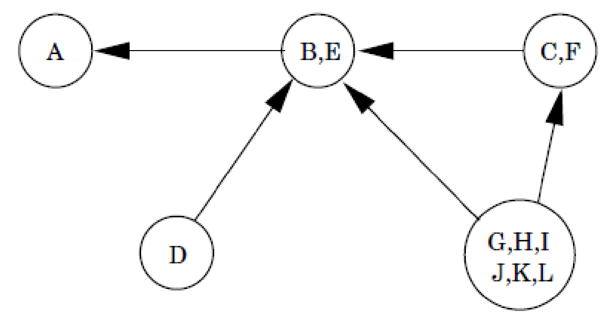
G





G^R





Nota bene: le componenti fortemente connesse sono le stesse!
(perchè?)

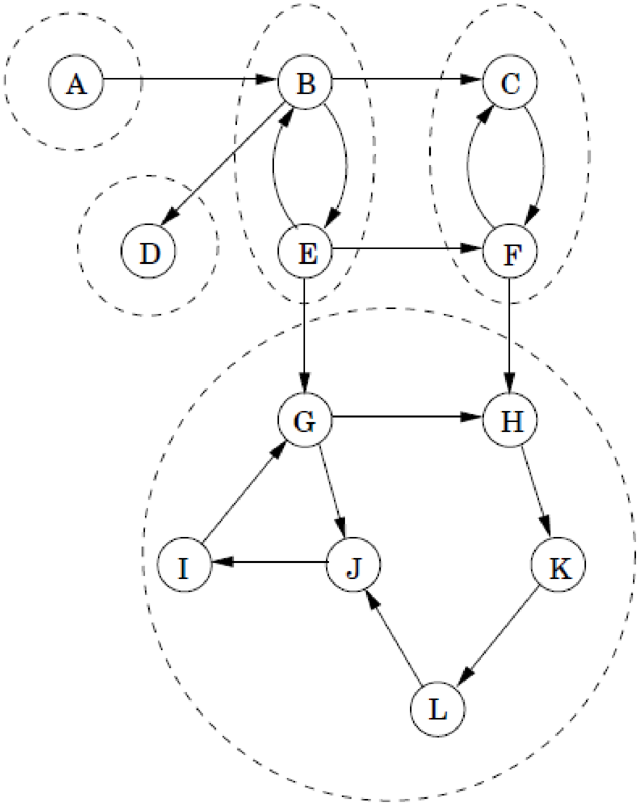
VisitaDFS (grafo *G*)

1. calcola G^R
2. esegui DFS(G^R) per trovare valori post(v)
3. **return** CompConnesse(G)

Complessità temporale:
se *G* è rappresentato con liste di adiacenza
$\Theta(n+m)$

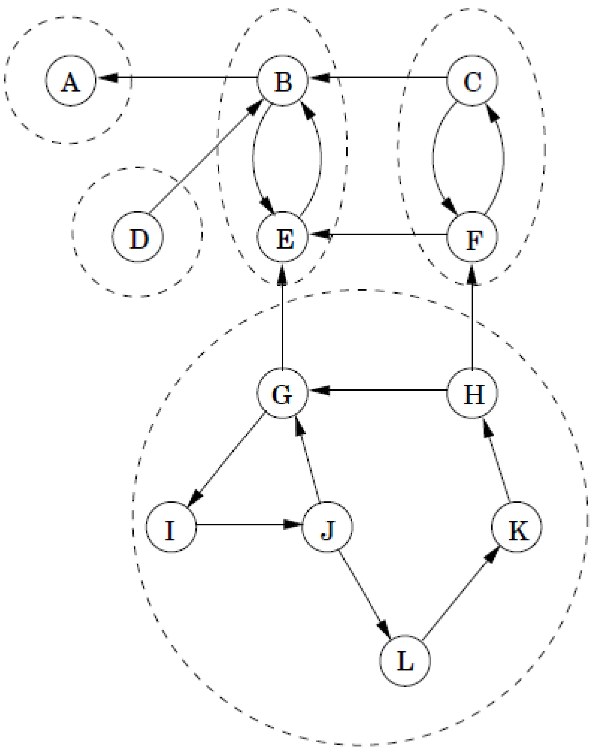
CompConnesse (grafo *G*)

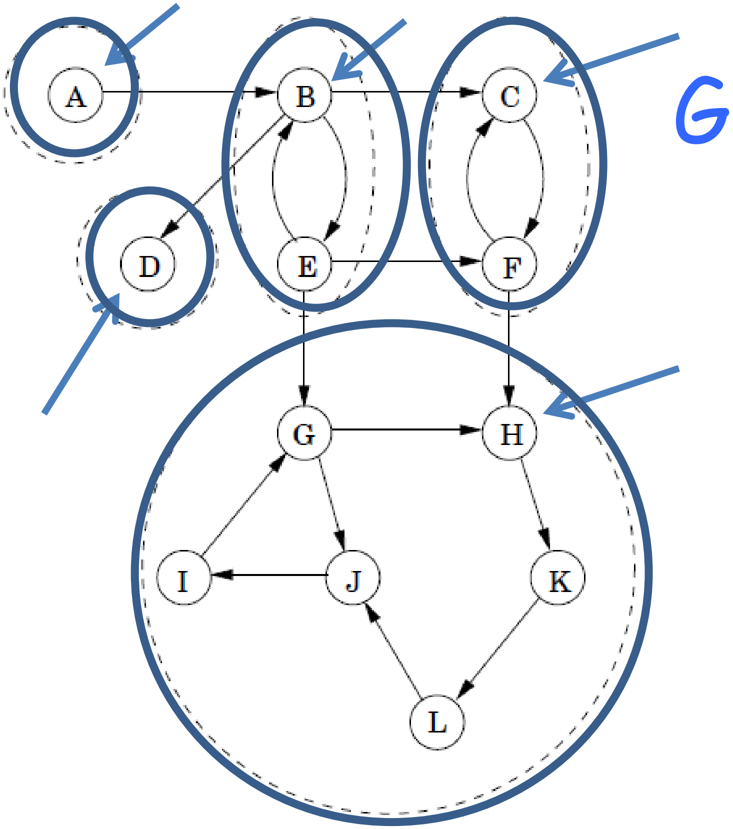
1. **for each** nodo *v* **do** imposta *v* come *non marcato*
2. *Comp* ← ∅
3. **for each** nodo *v* in ordine decrescente di post(v) **do**
4. **if** (*v* è *non marcato*) **then**
5. *T* ← albero vuoto
6. visitaDSFRicorsiva(*v*,*T*)
7. aggiungi *T* a *Comp*
8. **return** *Comp*

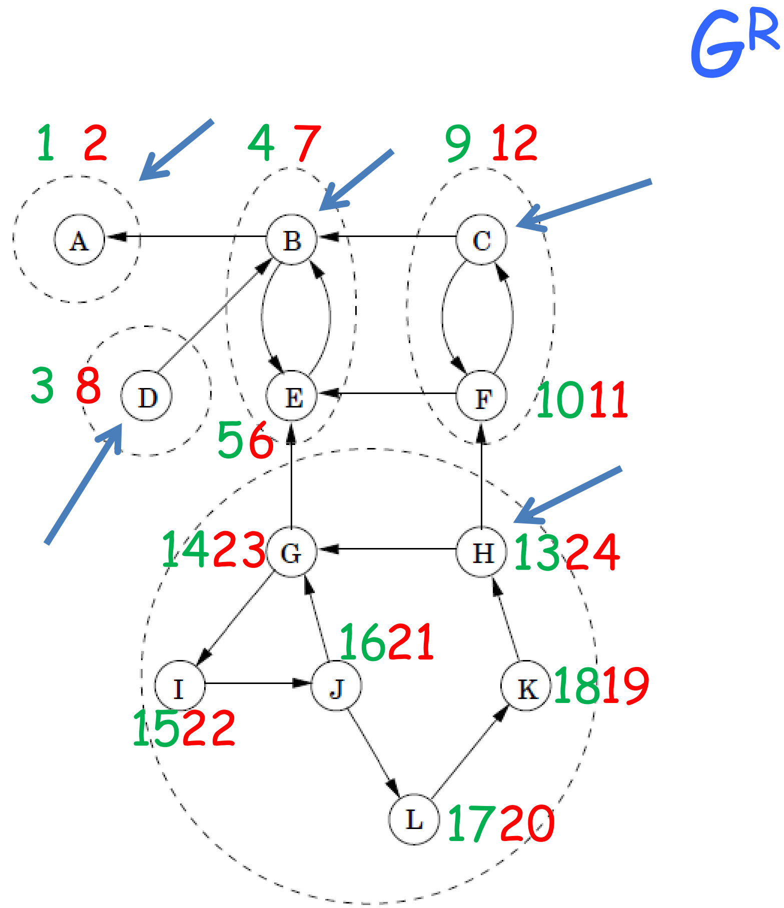


G

G^R




A
B
C
D
E
F
G
H
I
J
K
L
G


G^R
1 2
4 7
9 12
3 8
5 6
10 11
14 23
13 24
16 21
15 22
18 19
17 20
A
B
C
D
E
F
G
H
I
J
K
L