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

1. *marca e visita il vertice* v
2. **for each** (arco (v, w)) **do**
3. **if** (w non è marcato) **then**
4. aggiungi l'arco (v, w) all'albero T
5. `visitaDFSRicorsiva`(w, T)

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

pre(v)=clock
clock=clock+1

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

6. $T \leftarrow$ albero vuoto
7. `visitaDFSRicorsiva`(s, T)
8. **return** T

clock=1

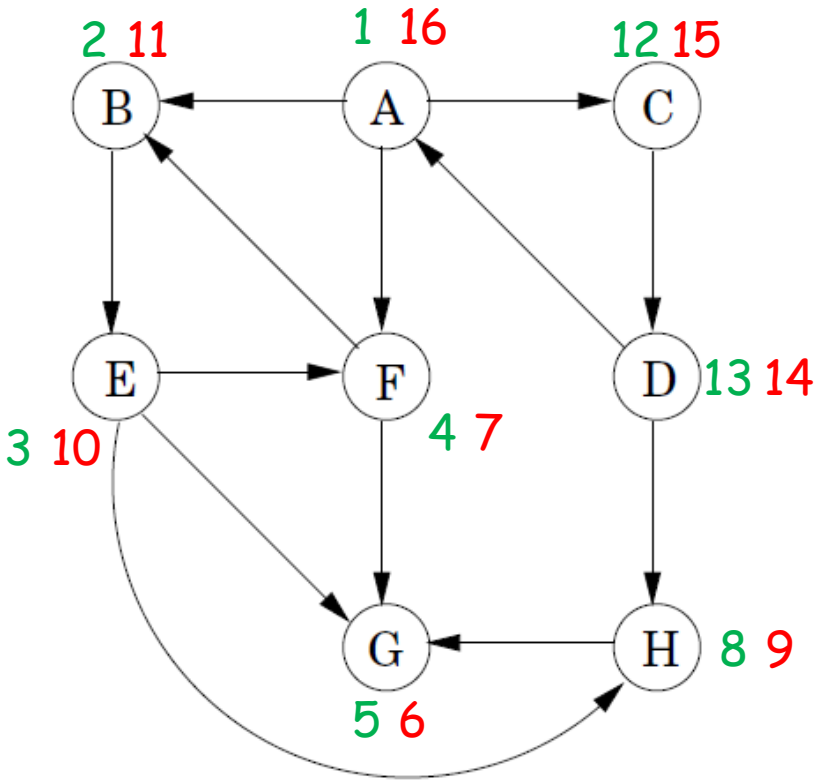
pre(v): tempo in cui viene "scoperto" v
post(v): tempo in cui si "abbandona" v

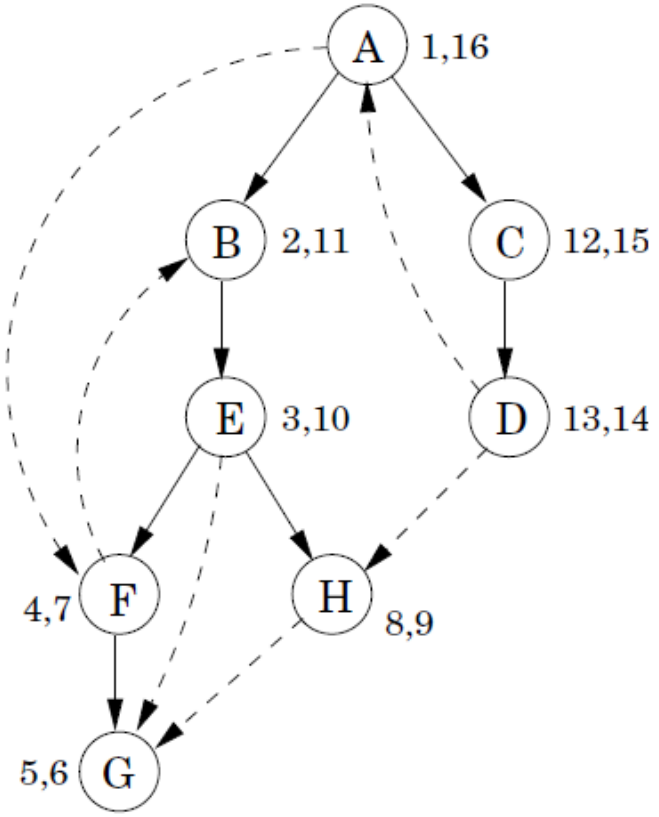
quando non tutti i nodi sono raggiungibili dal punto di partenza

VisitaDFS (grafo *G*)

1. **for each** nodo *v* **do** imposta *v* come *non marcato*
2. clock=1
3. *F* ← foresta vuota
4. **for each** nodo *v* **do**
5. **if** (*v* è *non marcato*) **then**
6. *T* ← albero vuoto
7. visitaDSFRicorsiva(*v*,*T*)
8. aggiungi *T* ad *F*
9. **return** *F*

Un esempio





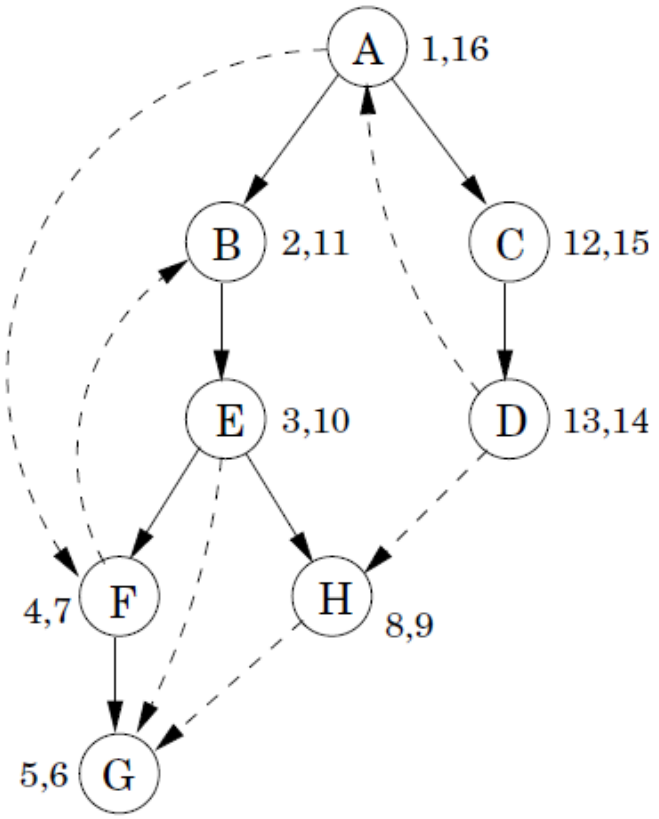
pre(v) post(v)

v

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

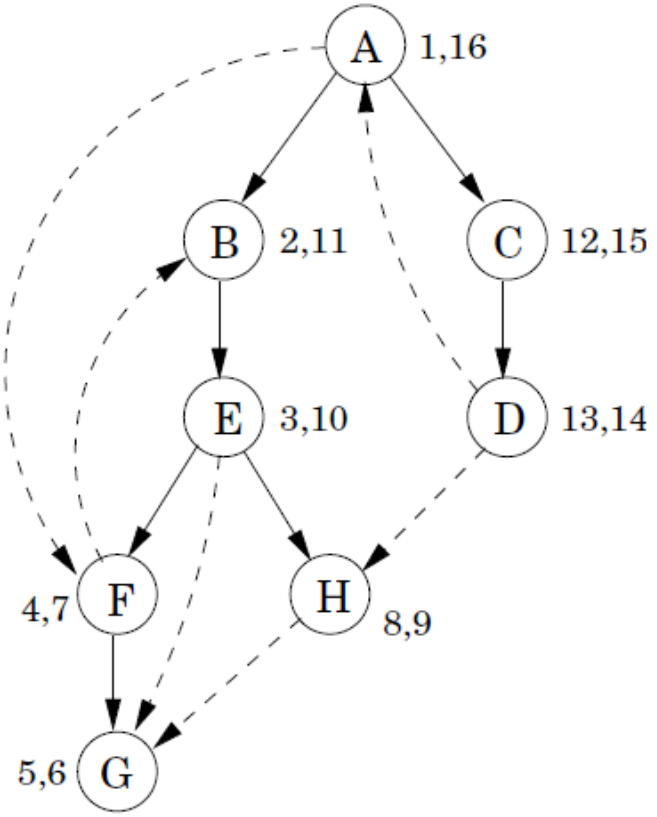
u è antenato di v nell'albero DFS, se pre(u) < pre(v) < post(v) < post(u) condizione che rappresentiamo così:



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
$[_u \; [_v \;]_v \;]_u$	in avanti
$[_v \; [_u \;]_u \;]_v$	all'indietro
$[_v \;]_v \; [_u \;]_u$	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

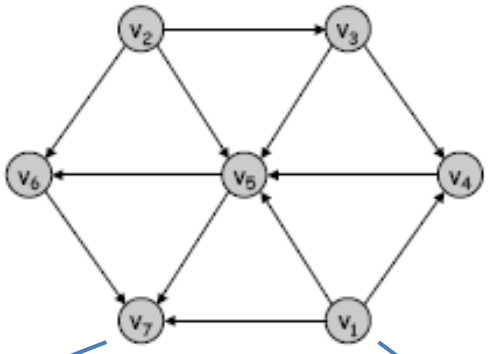
($\Rightarrow$): se c'è ciclo $\langle v_0, v_1, \ldots, v_k = v_0 \rangle$
sia v_i è il primo nodo scoperto nella visita
v_i termina la visita dopo che v_{i+1} ha terminato la sua
v_{i+1} termina la visita dopo che v_{i+2} ha terminato la sua
⋮
quindi, per transitività, v_i termina la visita dopo che v_{i-1} ha terminato la sua
allora (v_{i-1}, v_i) è un arco all'indietro

Definizione

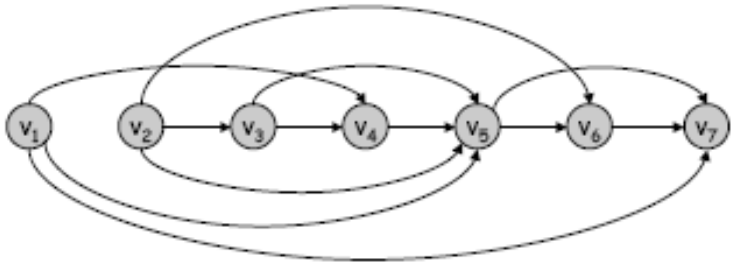
Un grafo diretto aciclico (DAG) è un grafo diretto G che non contiene cicli (diretti).

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

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

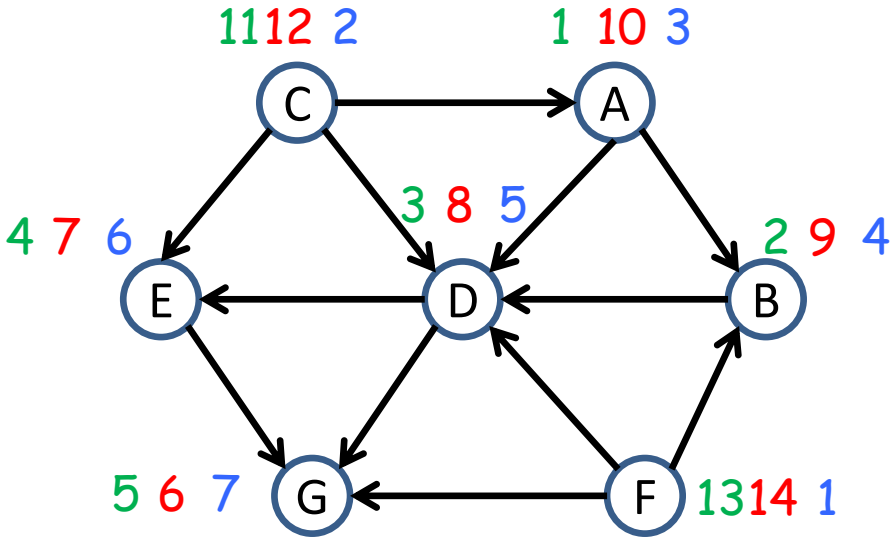
Algoritmo:
fai una visita DFS e restituisci i nodi in ordine decrescente rispetto ai tempi di fine visita post(v)

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

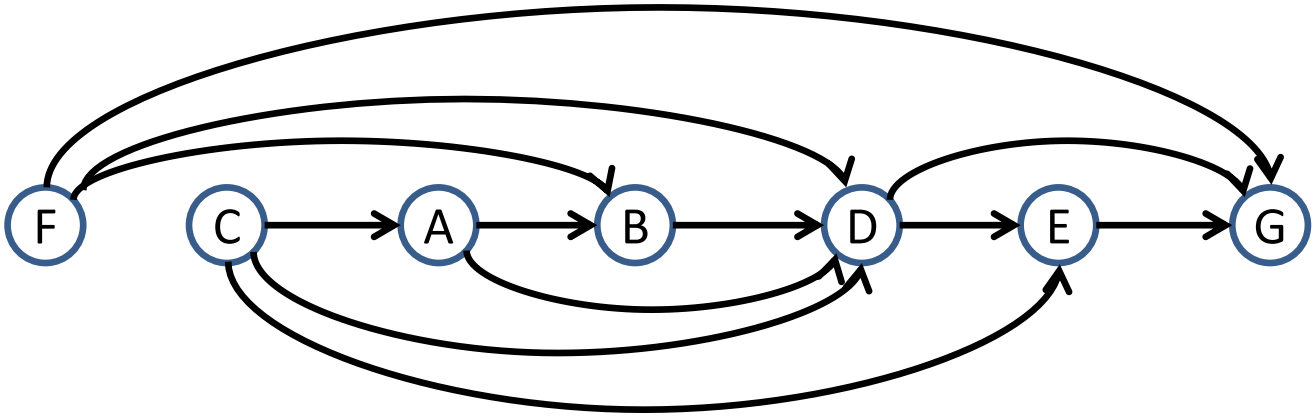
OrdinamentoTopologico (grafo *G*)

1. top=n; *L* ← 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

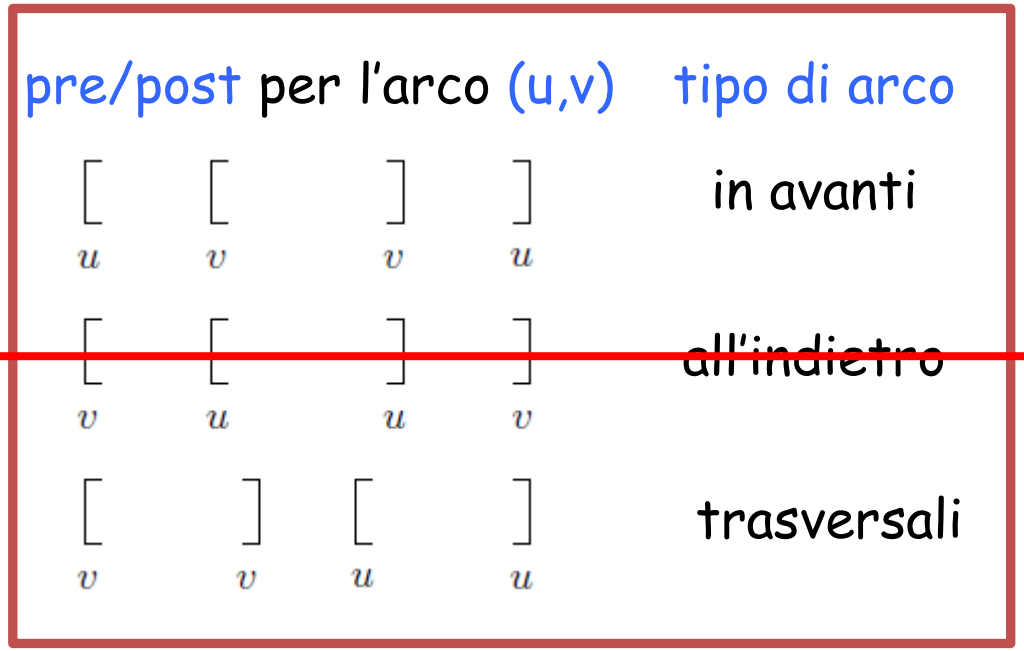


pre(v) post(v) σ(v)



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



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

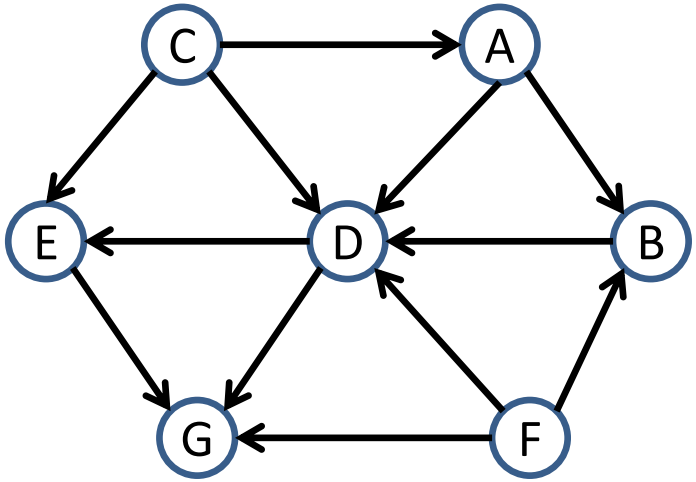
(*) **if** ($\widehat{G}$ non è diventato vuoto) **then errore** il grafo G non è aciclico

return ord

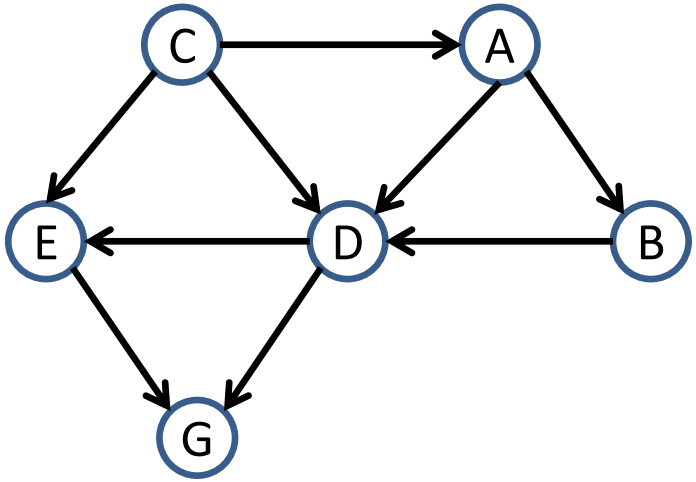
(*) perché altrimenti in $\hat{G}$ ogni vertice deve avere almeno un arco entrante, e quindi posso trovare un ciclo percorrendo archi entranti a ritroso, e quindi G non può essere aciclico)

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

Un esempio

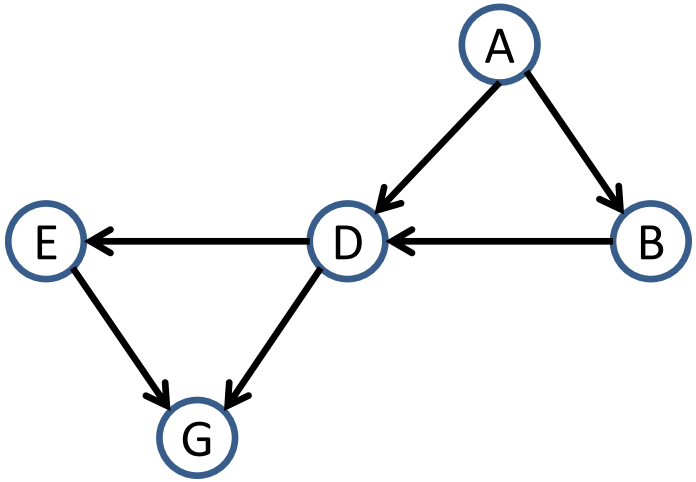


Un esempio



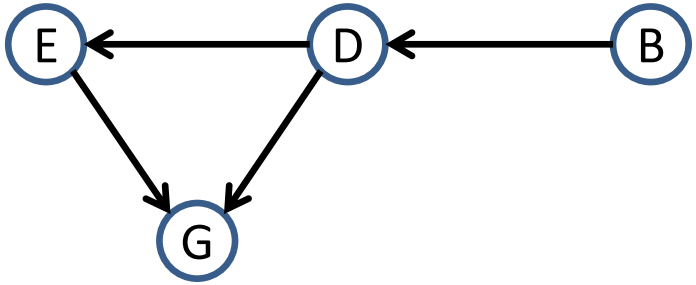
F

Un esempio

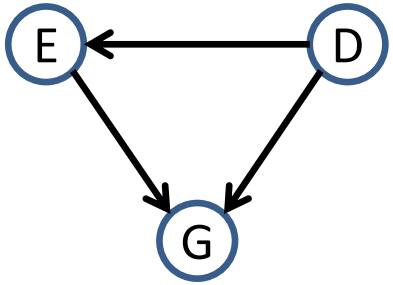


F C

Un esempio

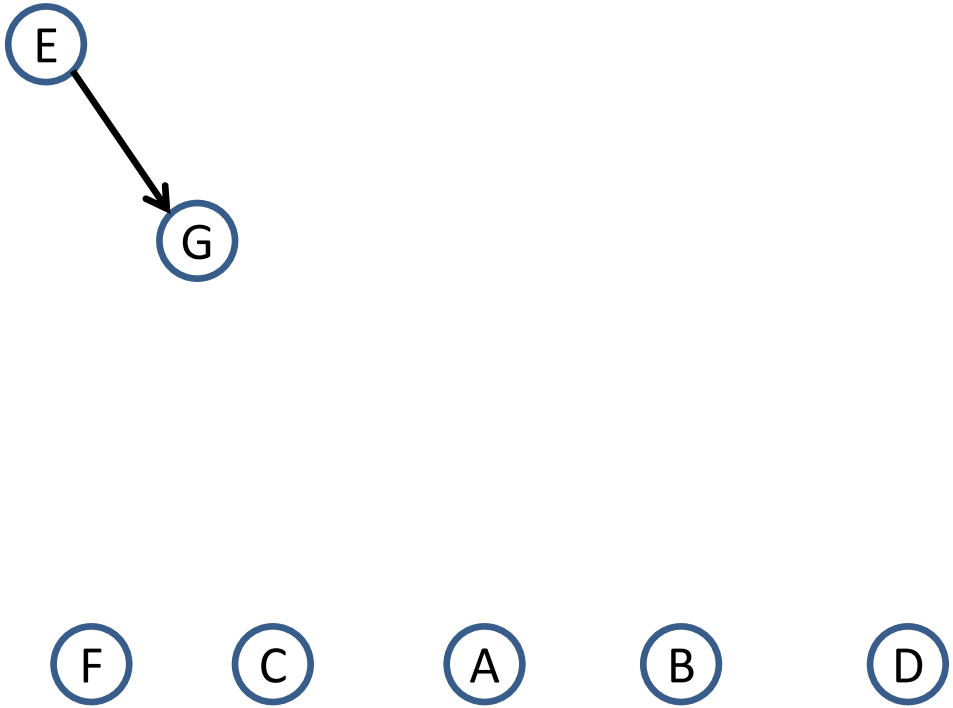


Un esempio

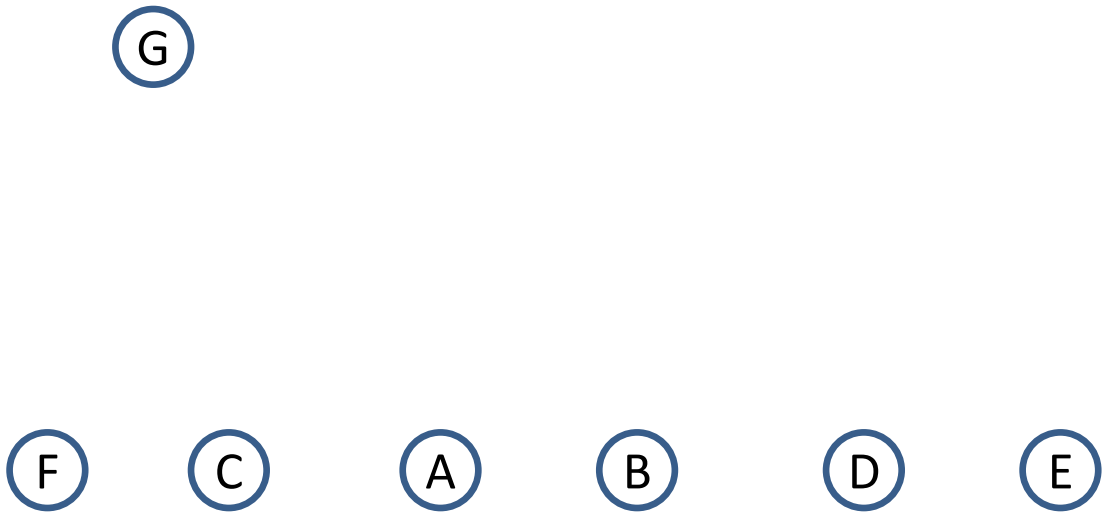


F C A B

Un esempio

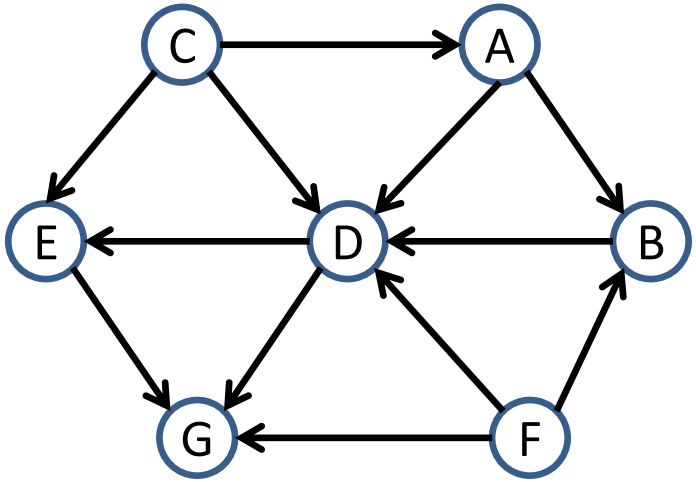


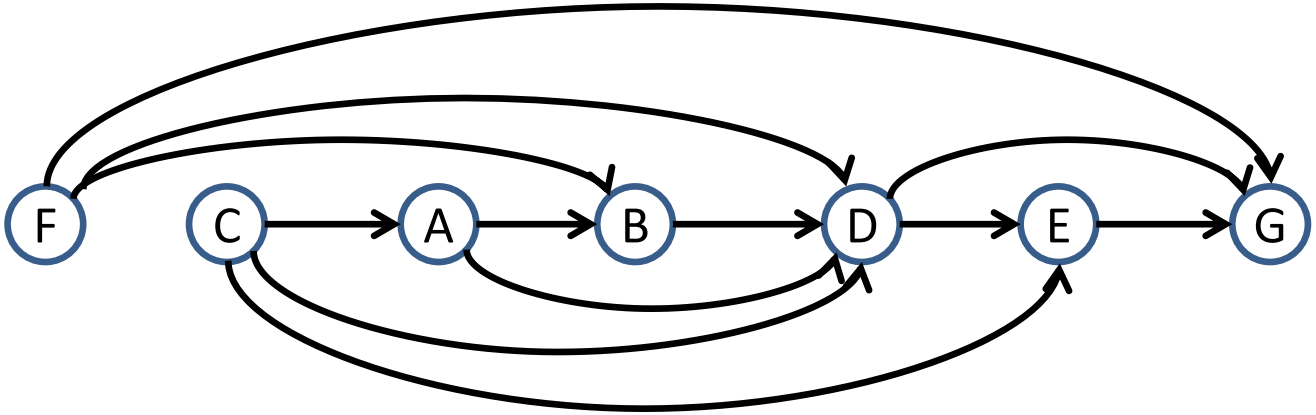
Un esempio



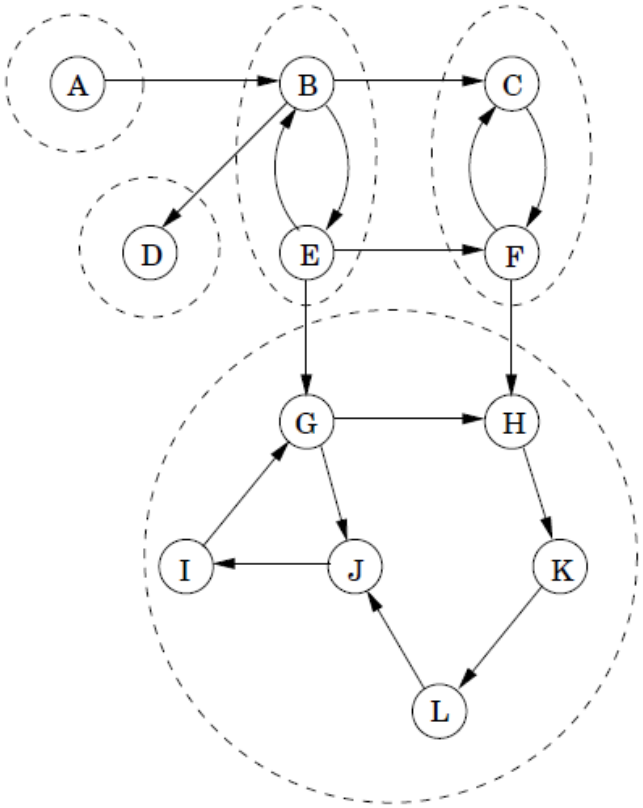
Un esempio

Un esempio



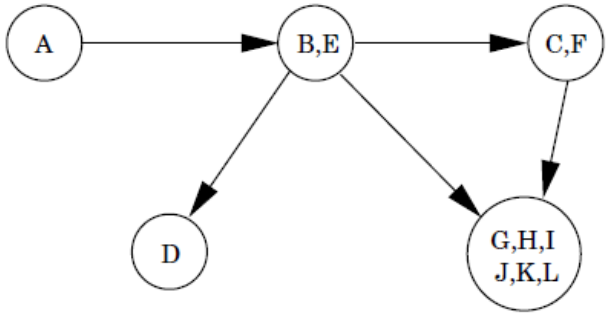


componenti fortemente connesse



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

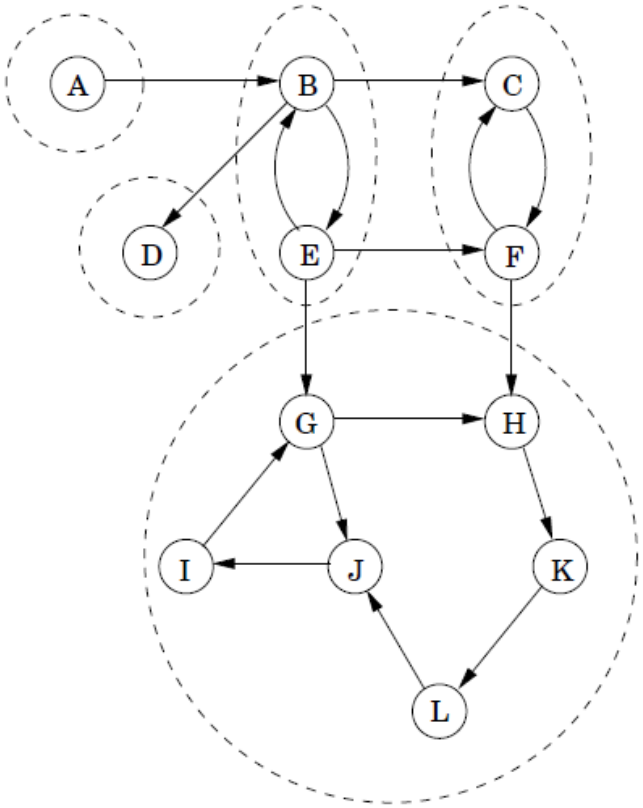
massimale: se si aggiunge un qualsiasi vertice a C la proprietà non è più vera



grafo delle componenti fortemente connesse di G

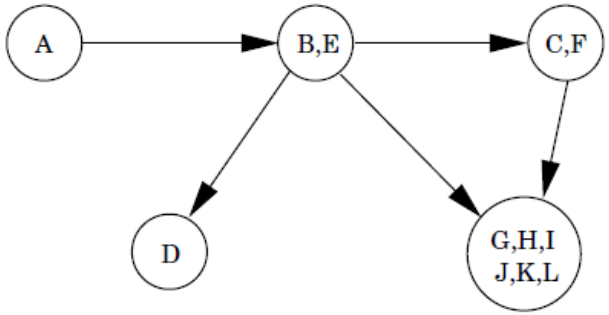
è sempre un DAG!

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

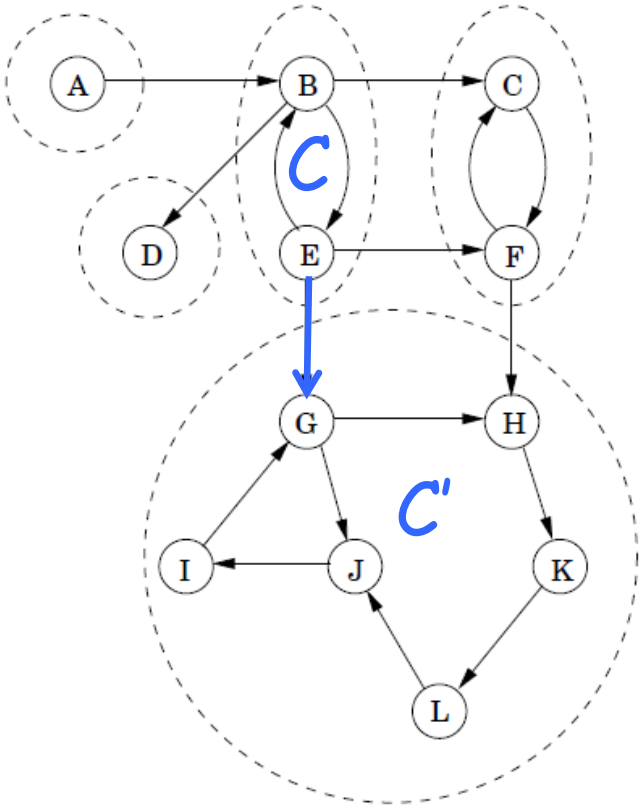


Proprietà 1: se si esegue la procedura visitaDFSricorsiva a partire da un nodo u la procedura termina dopo che tutti i nodi raggiungibili da u sono stati visitati

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



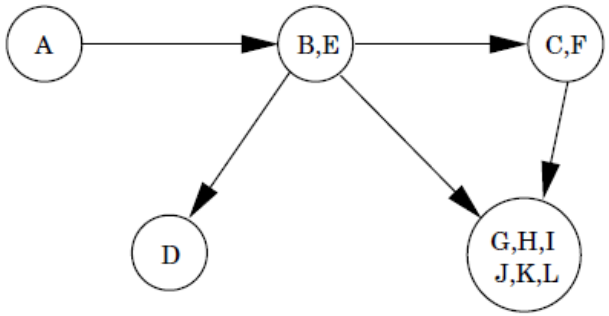
come trovo una componente pozzo?



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'

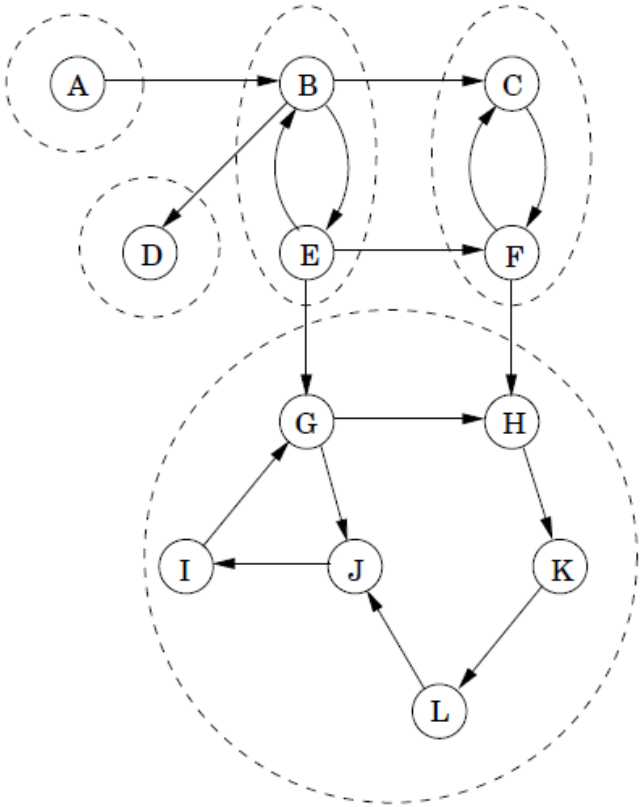
dim: se la DFS visita prima C' di C: banale. se visita prima C, allora si ferma dopo che ha raggiunto tutti i nodi di C e C' e termina su un nodo di C.

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

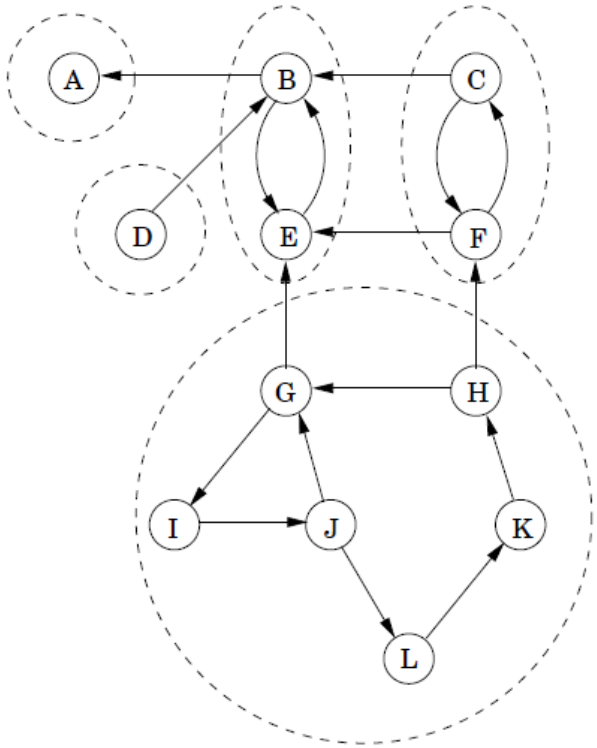


ma avevamo bisogno di una componente *pozzo*?

idea: invertiamo gli archi!

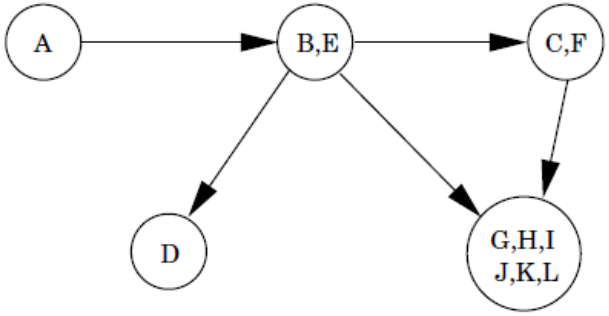


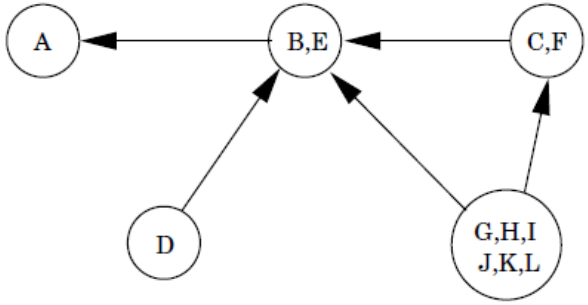
G



G^R

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





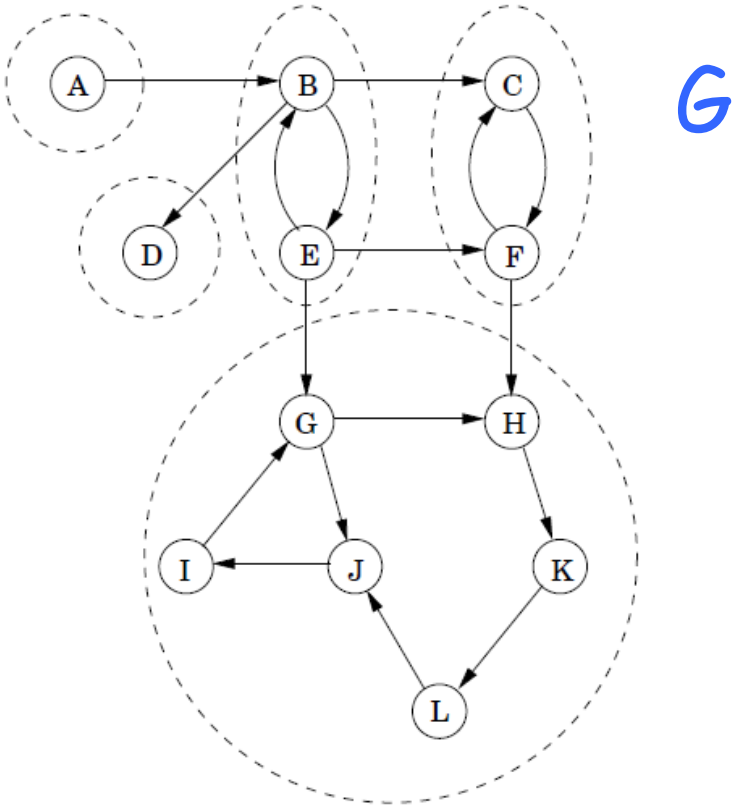
VisitaDFS (grafo *G*)

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

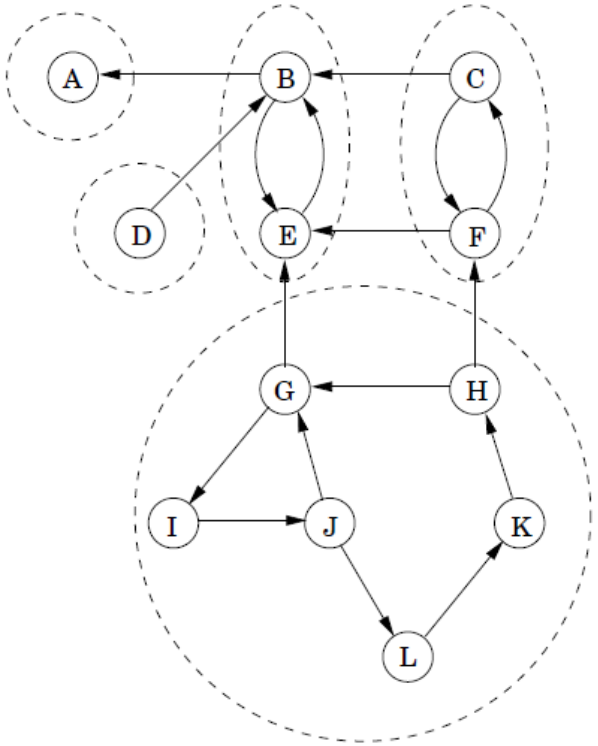
Complessità temporale:
se G è rappresentato
con liste di adiacenza
$\Theta(n+m)$

CompConnesse (grafo *G*)

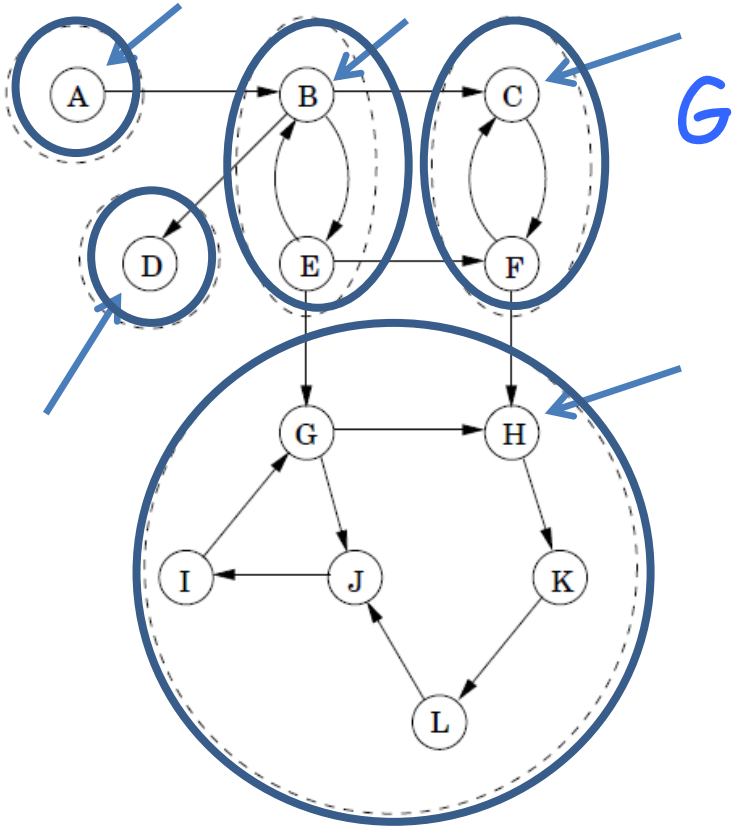
1. **for each** nodo *v* **do** imposta *v* come *non marcato*
2. *Comp* ← ∅
3. **for each** nodo *v* in ordine decrescente di post(v) **do**
4. **if** (*v* è *non marcato*) **then**
5. *T* ← albero vuoto
6. visitaDSFRicorsiva(*v*,*T*)
7. aggiungi *T* a *Comp*
8. **return** *Comp*

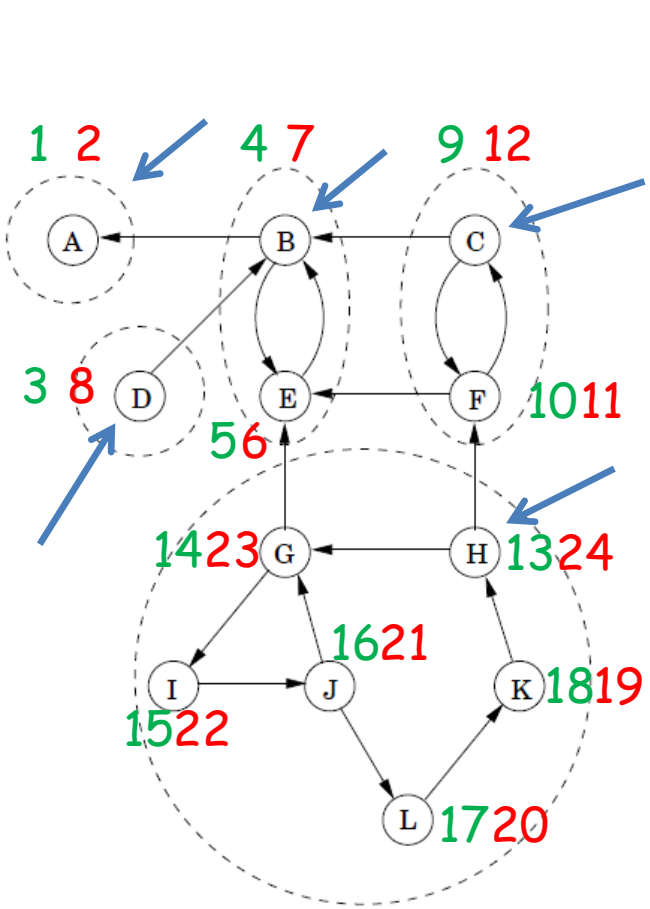


G



G^R


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


1 2
4 7
9 12
A
B
C
3 8
D
E
F
1011
56
1423
G
H
1324
1621
I
J
K
1819
1522
L
1720
G^R

Problema 1

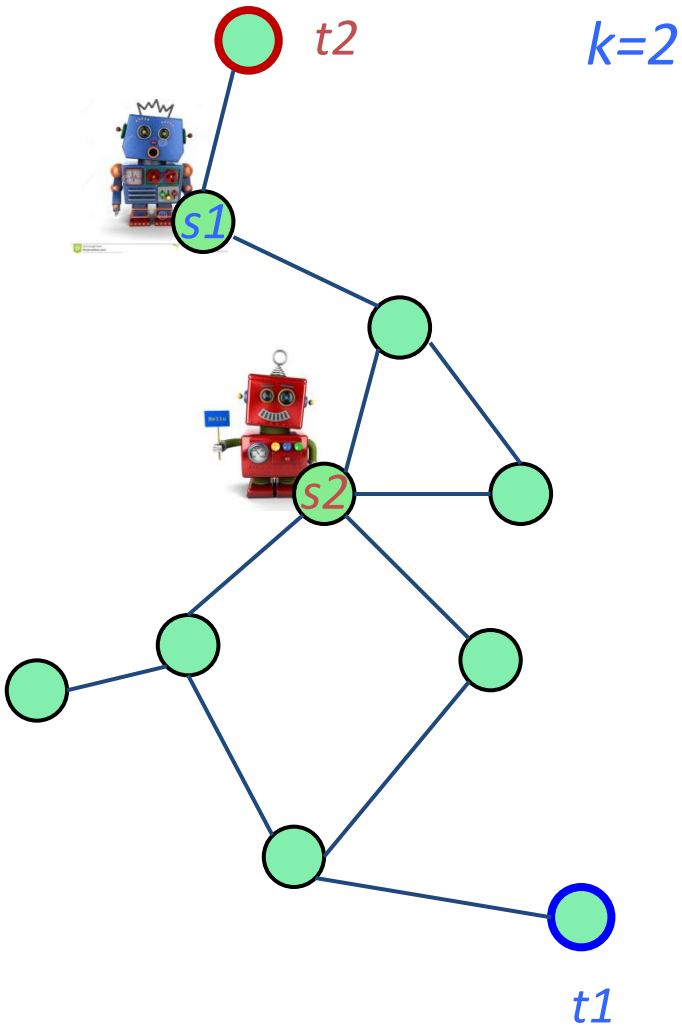
In un corso di laurea sono previsti un certo numero di esami obbligatori. Esistono inoltre dei vincoli di propedeuticità: se un esame A è propedeutico ad un esame B allora B deve essere sostenuto in una sessione d'esami successiva a quella in cui è stato sostenuto A. Se due o più esami non violano alcun vincolo di propedeuticità allora possono essere sostenuti tutti all'interno della stessa sessione. Supponendo di superare ogni esame al primo tentativo fornire un algoritmo lineare che calcoli il minimo numero di sessioni necessarie per sostenere tutti gli esami. Per ogni sessione d'esami, fornire inoltre l'insieme di esami da sostenere.
Suggerimento: Considerare il grafo delle propedeuticità e le sue proprietà strutturali.

Problema 2

Su un grafo non orientato e non pesato $G = (V, E)$ si possono muovere due robot telecomandabili a distanza. All'inizio i due robot sono posizionati su due nodi del grafo, diciamo s_1 e s_2. In ogni istante di tempo è possibile effettuare la seguente *mossa*: ordinare a uno dei due robot di spostarsi dal nodo su cui è in un nodo adiacente. L'obiettivo è portare il robot che si trova su s_1 in un certo nodo t_1 e il robot che si trova su s_2 in un certo nodo t_2. Le antenne dei robot, però, suffrono di problemi di interferenze: se i robot finiscono troppo vicini l'uno con l'altro non riescono più a ricevere il segnale e quindi a muoversi. Per questo motivo si vuole che i robot in ogni istante di tempo sono sempre a distanza reciproca (nel grafo) di almeno k, dove k è un parametro del problema. Progettare un algoritmo che trovi il minimo numero di mosse che porta i robot nelle posizioni desiderate.

trovare la sequenza minima di mosse per portare il robot 1 in t1 e robot 2 in t2

con il vincolo che i due robot devono essere sempre a distanaza almeno k



Problema 3

È dato un grafo non orientato $G = (V, E)$. Si vogliono posizionare delle monete sui vertici di G. Per posizionare una moneta, questa deve dapprima essere piazzata su di un vertice x di G non ancora occupato e successivamente spostata su un vertice adiacente ad x, anch'esso non occupato. Progettare un algoritmo lineare che trovi il modo di posizionare il maggior numero di monete possibili sui vertici di G.

Suggerimento: Pensare al caso in cui G è un albero ed estendere la soluzione trovata a grafi generici.