

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 pre(v)=clock
2. **for each** (arco (v, w)) **do** clock=clock+1
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

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

6. $T \leftarrow$ albero vuoto clock=1
7. visitaDFSRicorsiva(s, T)
8. **return** T

pre(v): tempo in cui viene "scoperto" v

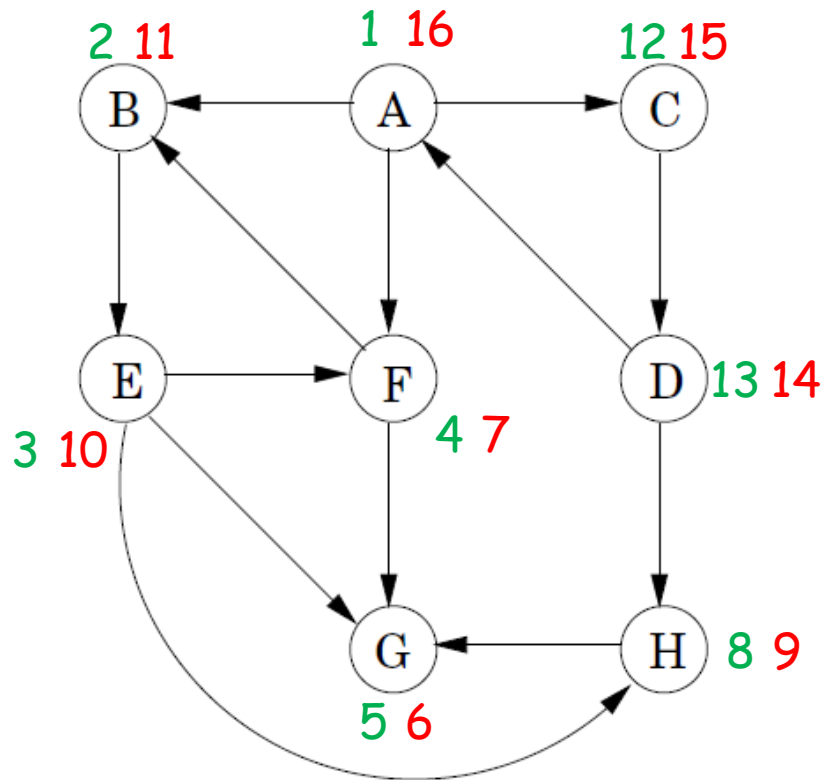
post(v): tempo in cui si "abbandona" v

quando non tutti i nodi sono raggiungibili dal
punto di partenza

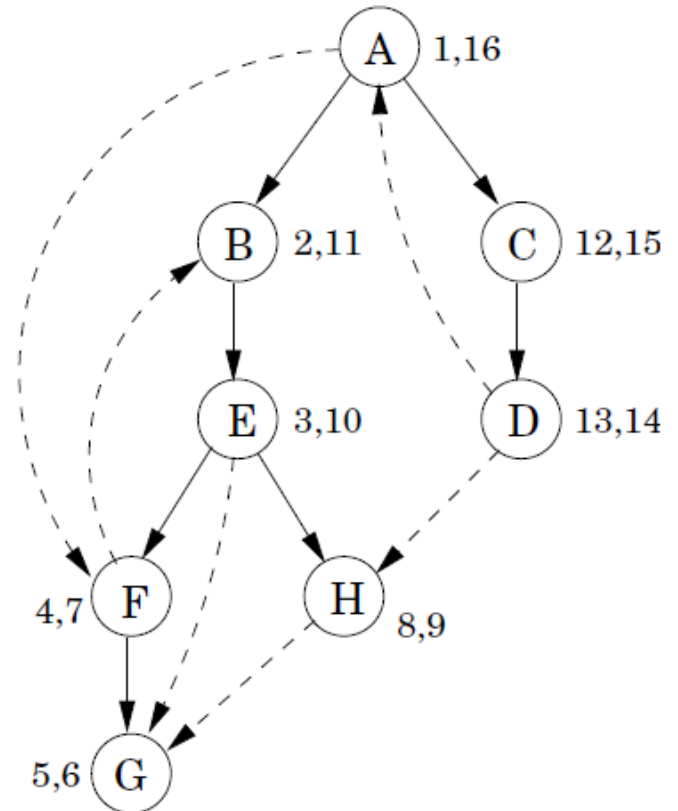
VisitaDFS (grafo G)

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

Un esempio



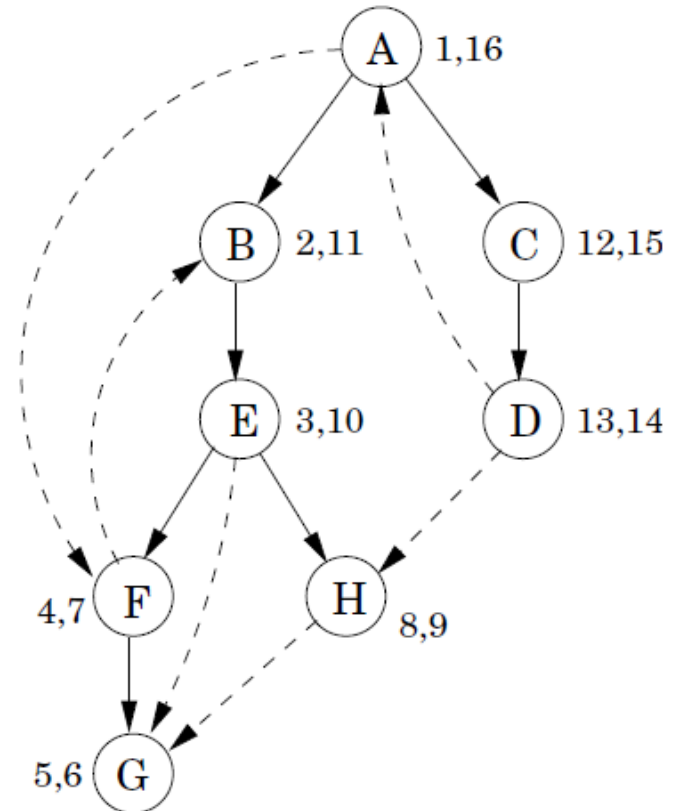
pre(v) post(v)



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

[	[	]	]
<i>u</i>	<i>v</i>	<i>v</i>	<i>u</i>

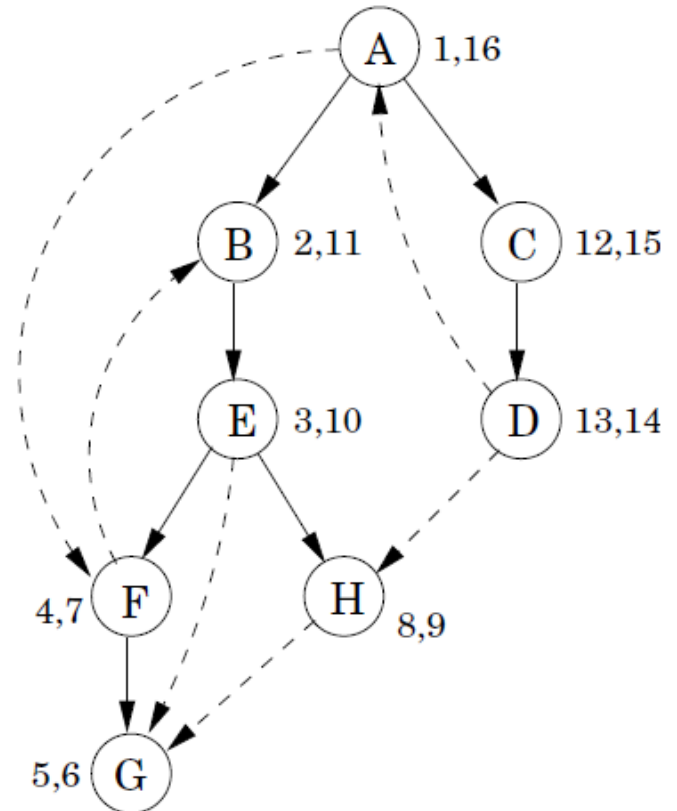
in avanti

[	[	]	]
<i>v</i>	<i>u</i>	<i>u</i>	<i>v</i>

all'indietro

[	]	[	]
<i>v</i>	<i>v</i>	<i>u</i>	<i>u</i>

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

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

$\vdots$

quindi, per transitività, v_i termina la visita dopo che v_{i-1} ha terminato la sua

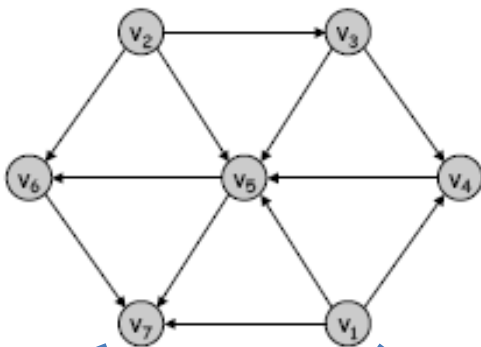
allora (v_{i-1}, v_i) è un arco all'indietro

Definizione

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

Definizione

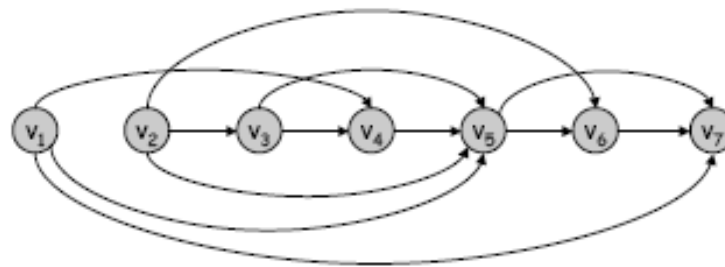
Un **ordinamento topologico** di un grafo diretto $G=(V,E)$ è una funzione biettiva $\sigma:V \rightarrow \{1,2,\dots,n\}$ tale che per ogni arco $(u,v) \in E$, $\sigma(u) < \sigma(v)$.



a DAG

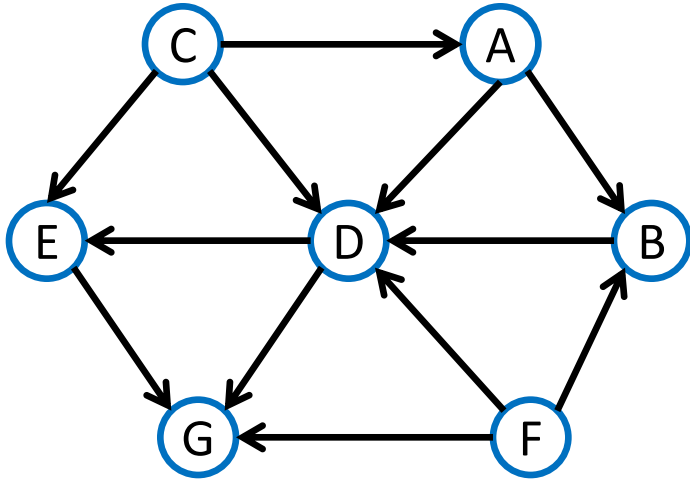
pozzo: solo
archi entranti

sorgente: solo
archi uscenti



a topological ordering

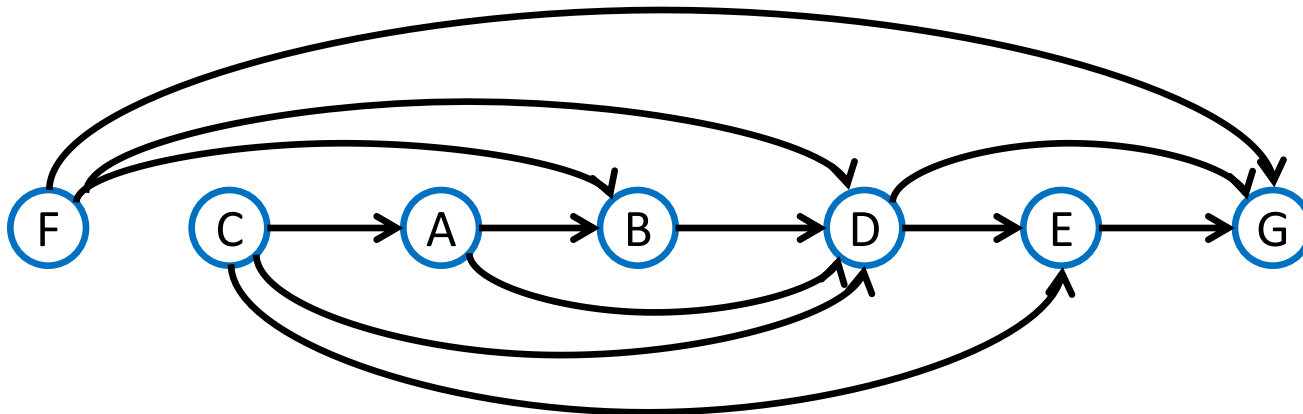
reti "delle dipendenze"



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 $\langle v_0, v_1, \dots, v_k = v_0 \rangle$ un ciclo

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

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

calcolare ordinamento topologico

Algoritmo:

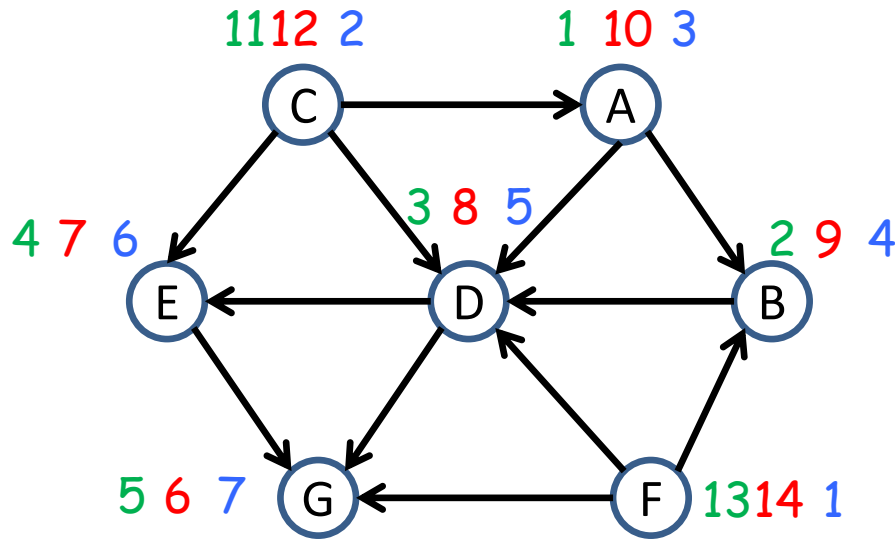
fai una visita DFS e restituisci i nodi in ordine decrescente rispetto ai tempi di fine visita $\text{post}(v)$

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

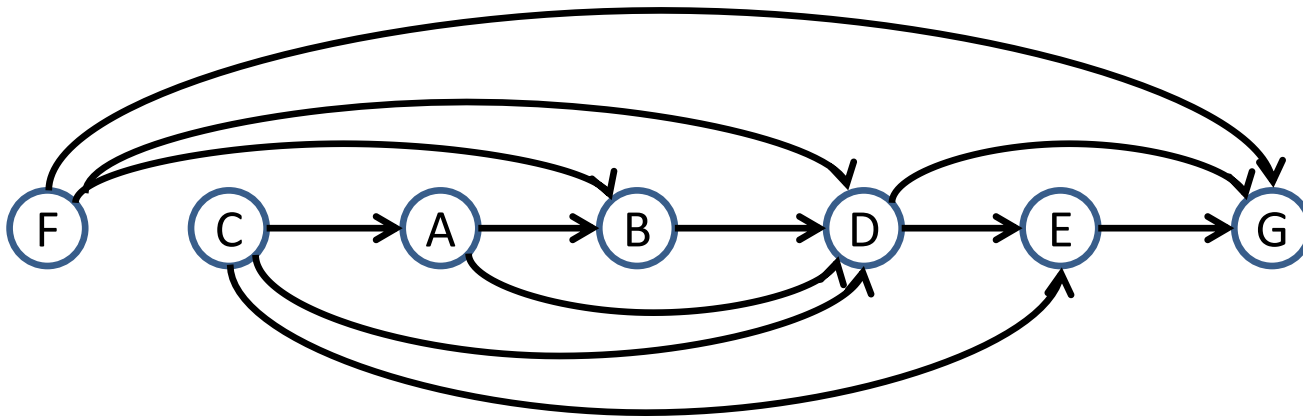
OrdinamentoTopologico (grafo G)

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

Un esempio



$pre(v)$ $post(v)$ $\sigma(v)$



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

$[$ $[$ $]$ $]$
 u v v u

in avanti

$[$ $[$ $]$ $]$
 v u u v

~~all'indietro~~

$[$ $]$ $[$ $]$
 v v u u

trasversali

non ci possono
essere archi
all'indietro

Un algoritmo alternativo

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

$\hat{G} \leftarrow G$

ord $\leftarrow$ lista vuota di vertici

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

 appendi u come ultimo elemento di *ord*

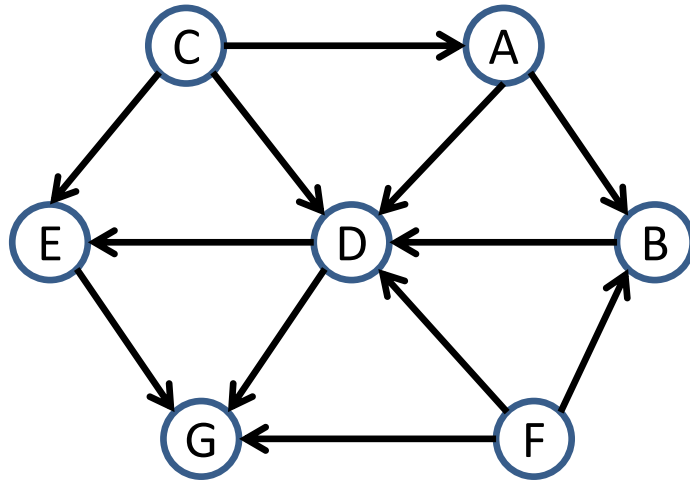
 rimuovi da $\hat{G}$ il vertice u e tutti i suoi archi uscenti

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

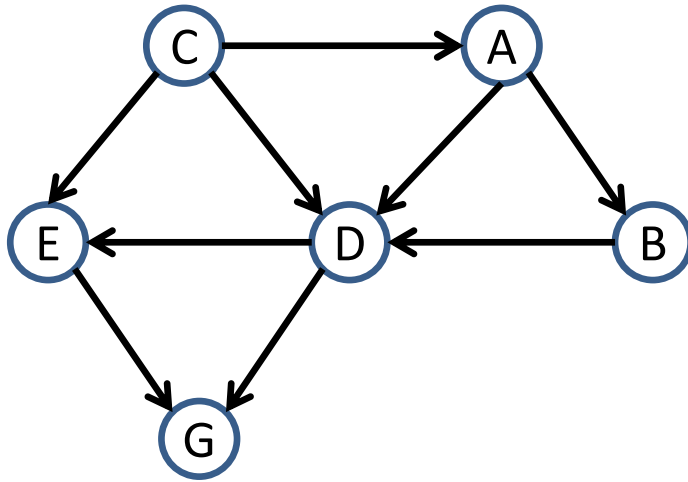
(*) perché altrimenti in $\hat{G}$ ogni vertice deve avere almeno un arco entrante, e quindi posso trovare un ciclo percorrendo archi entranti a ritroso, e quindi G non può essere aciclico)

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

Un esempio

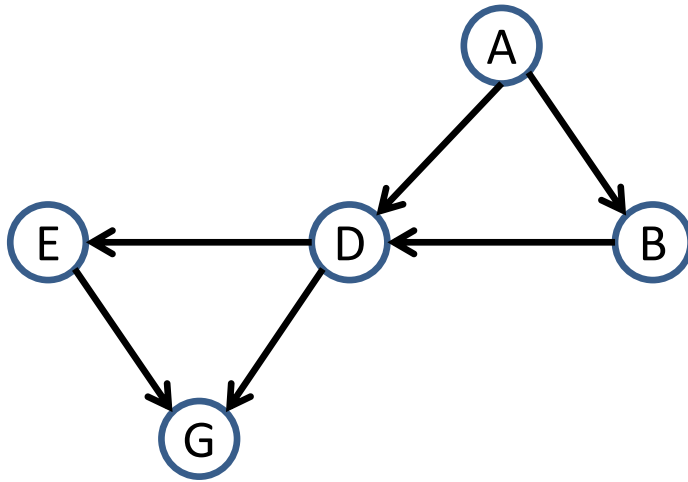


Un esempio



F

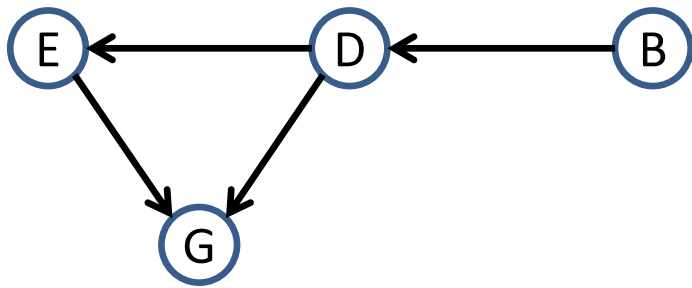
Un esempio



F

C

Un esempio

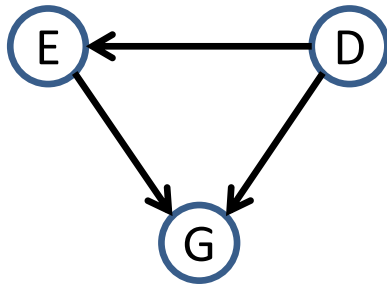


F

C

A

Un esempio



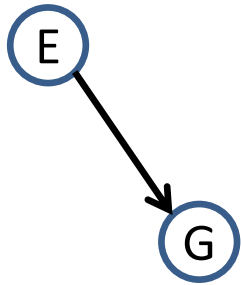
F

C

A

B

Un esempio



F

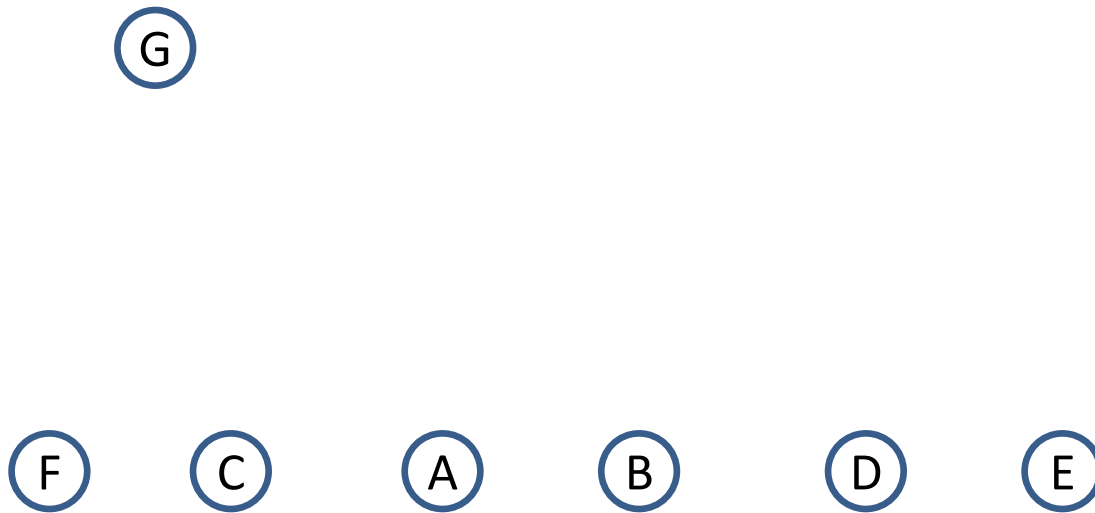
C

A

B

D

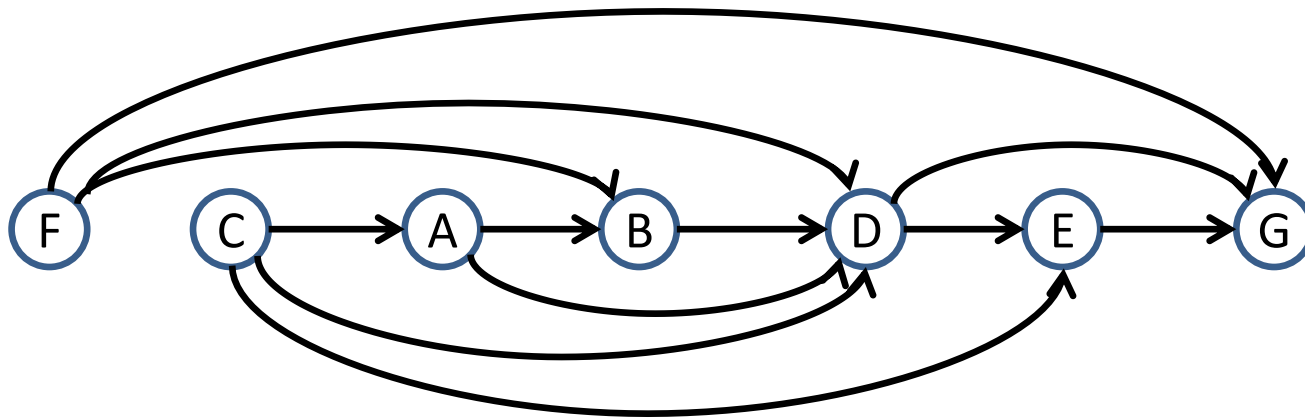
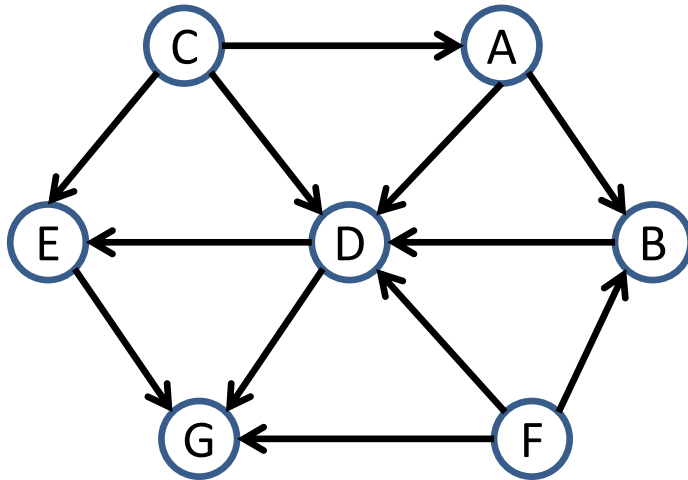
Un esempio



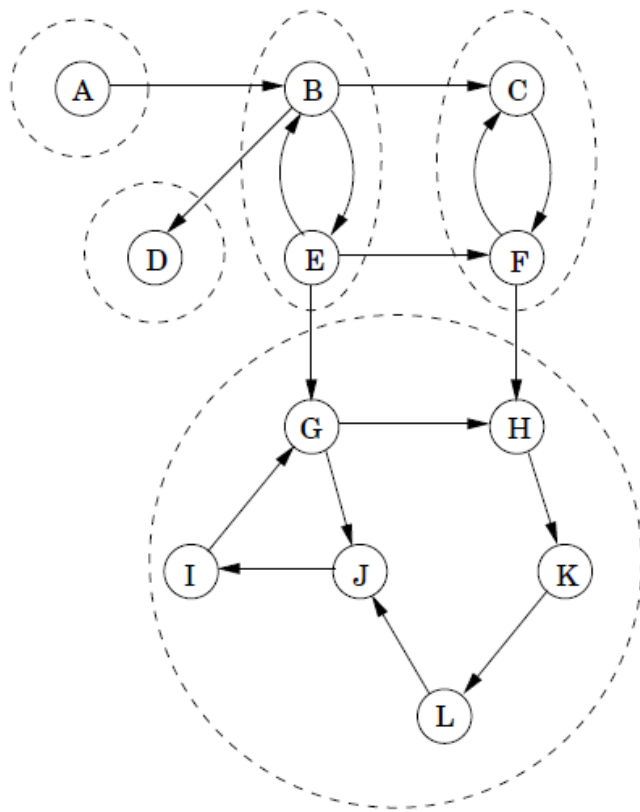
Un esempio



Un esempio



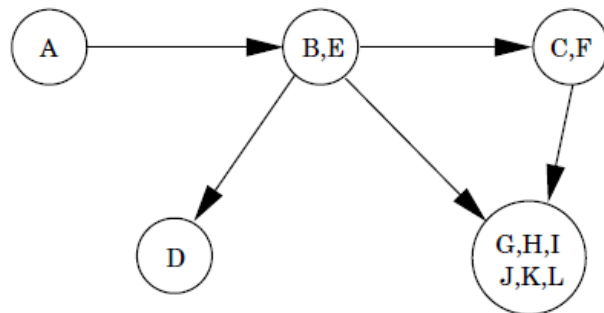
componenti fortemente
connesse



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

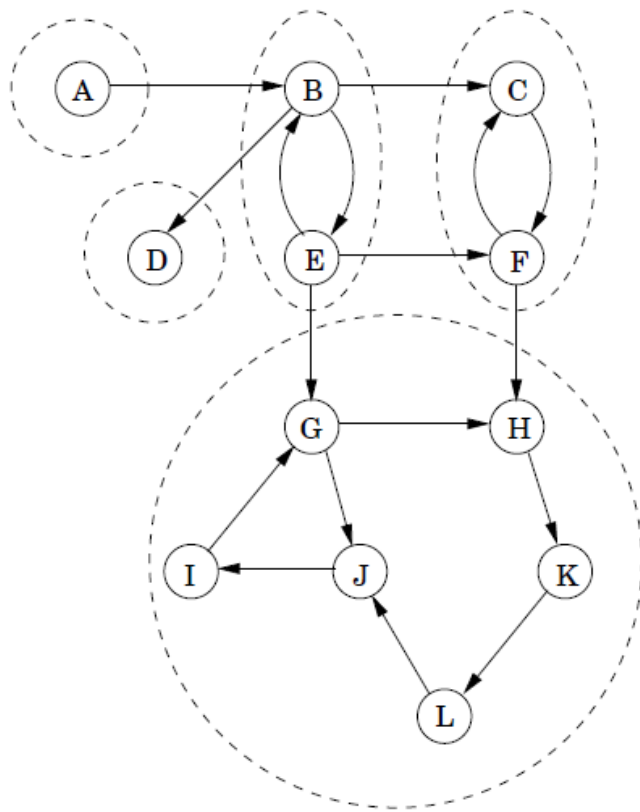
massimale: se si aggiunge un qualsiasi vertice a C la proprietà non è più vera

grafo delle componenti
fortemente connesse di G



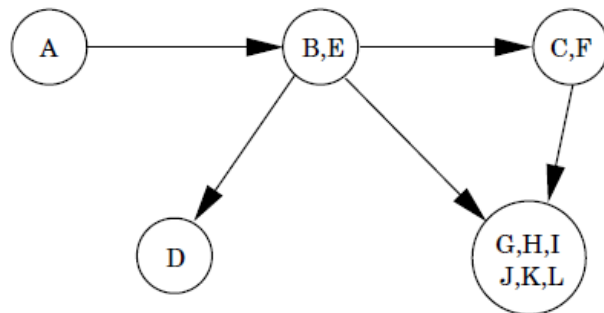
è sempre un **DAG**!

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

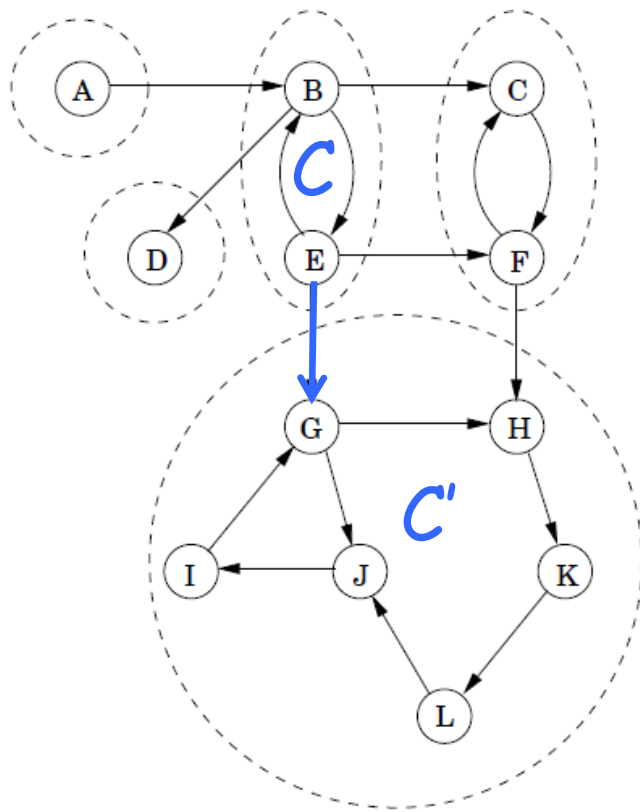


Proprietà 1: se si esegue la procedura visitaDFSricorsiva a partire da un nodo **u** la procedura termina dopo che tutti i nodi raggiungibili da **u** sono stati visitati

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



come trovo una
componente pozzo?

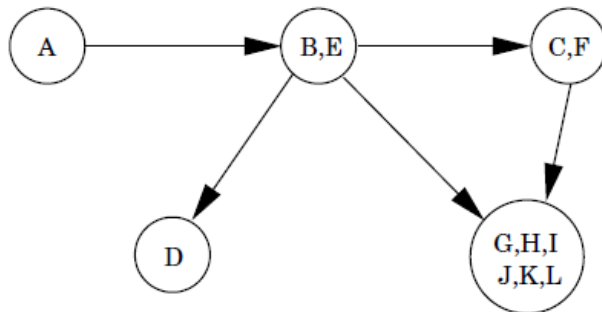


Proprietà 2: se C e C' sono due componenti e c'è un arco da un nodo in C verso uno in C' , allora il più grande valore $\text{post}()$ in C è maggiore del più alto valore di $\text{post}()$ di C'

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 .

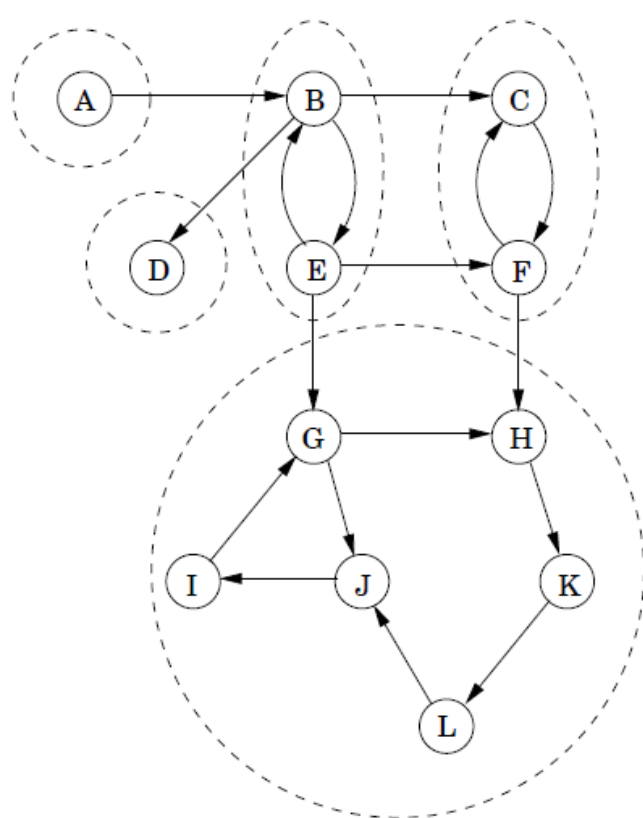


Proprietà 3: il nodo che riceve da una visita DFS il valore più grande di $\text{post}()$ appartiene a una componente *sorgente*

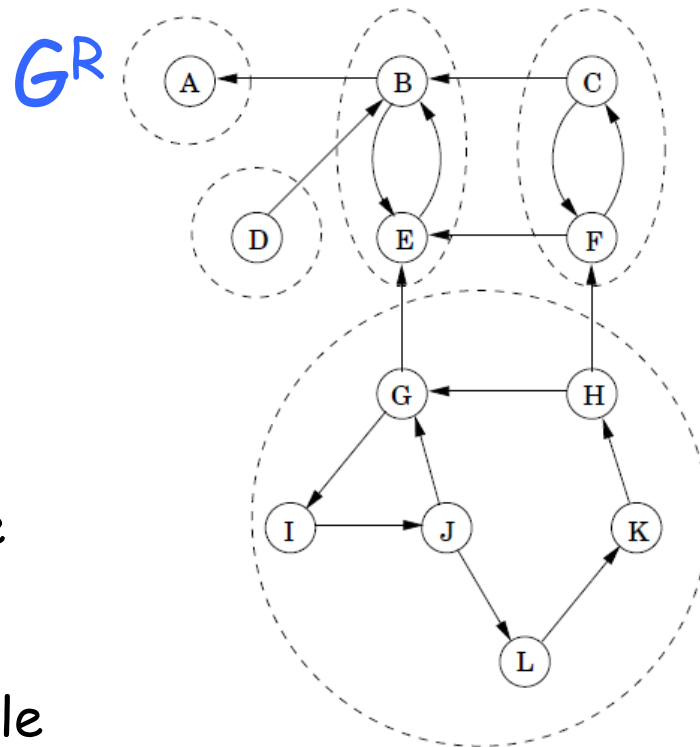


ma avevamo bisogno di una componente *pozzo*?

idea: invertiamo gli archi!

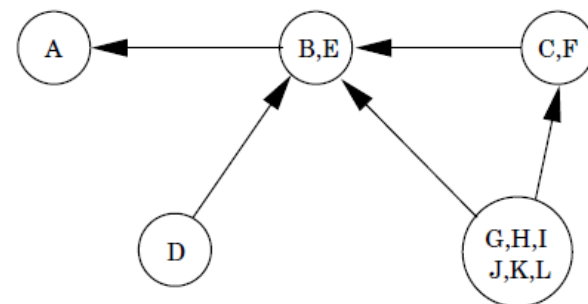
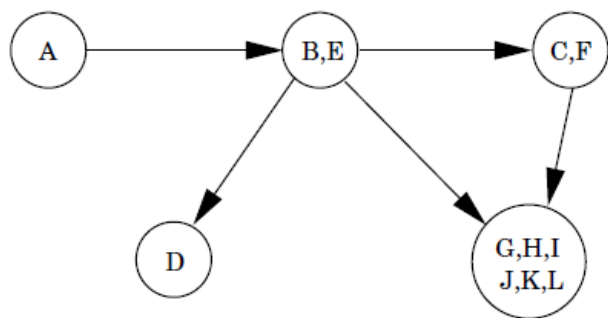


G



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)

CompConnesse (grafo G)

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

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

