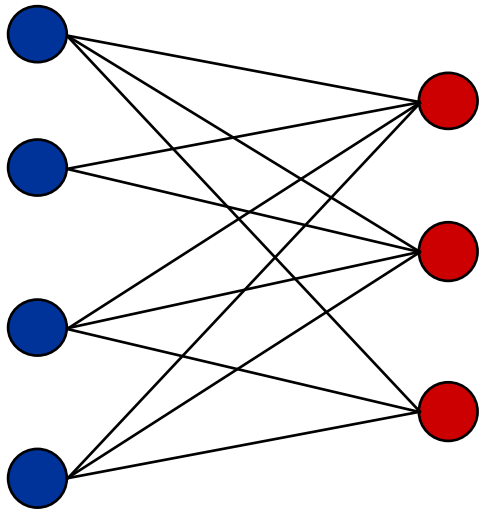
Grafi bipartiti

- Def. Un grafo non direzionato è **bipartito** se l'insieme di nodi può essere partizionato in due sottoinsiemi X e Y tali che ciascun arco del grafo ha una delle due estremità in X e l'altra in Y
  - Possiamo colorare i nodi con due colori (ad esempio, rosso e blu) in modo tale che ogni arco ha un'estremita rossa e l'altra blu.

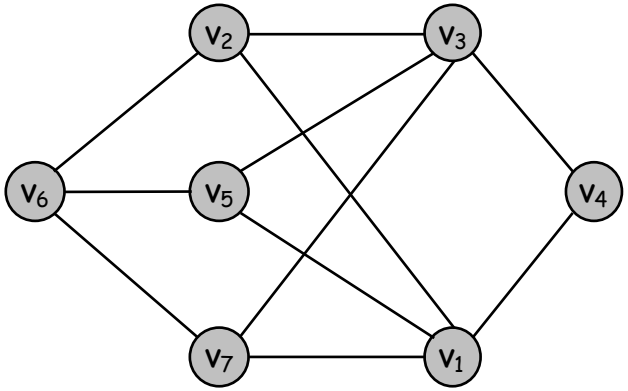
- Applicazioni.
  - Matrimoni stabili: uomini = rosso, donna = blu.
  - Scheduling: macchine = rosso, job = blu.



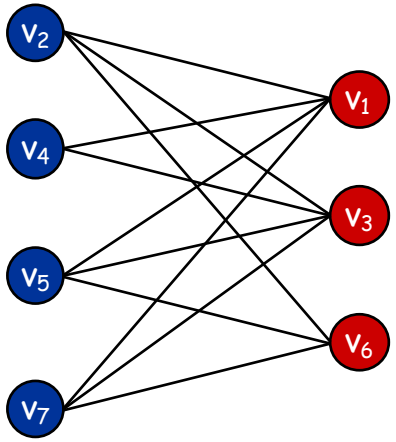
Un grafo bipartito

Testare se un grafo è bipartito

- **Testare se un grafo è bipartito.** Dato un grafo G, vogliamo scoprire se è bipartito.
  - Molti problemi su grafi diventano:
    - Più facili se il grafo sottostante è bipartito (matching: sottoinsieme di archi tali che non hanno estremità in comune)
    - Trattabili se il grafo è bipartito (max insieme indipendente)



Un grafo bipartito G

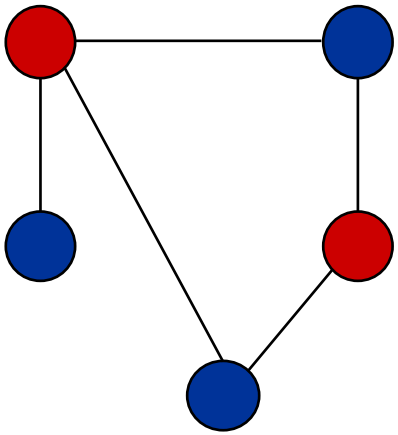


Modo alternativo di disegnare G

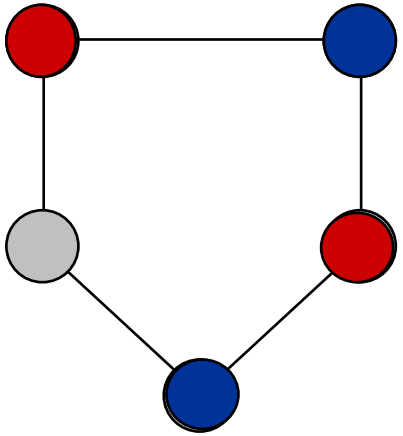
Grafi bipartiti

- Lemma. Se un grafo G è bipartito, non può contenere un ciclo dispari (formato da un numero dispari di archi)

- Dim. Non è possibile colorare di rosso e blu i nodi su un ciclo dispari in modo che ogni arco abbia le estremità di diverso colore.



bipartito
(2-colorabile)



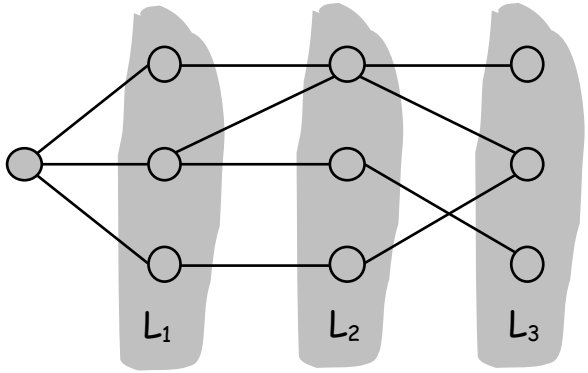
Non bipartito
(non 2-colorabile)

Grafi bipartiti

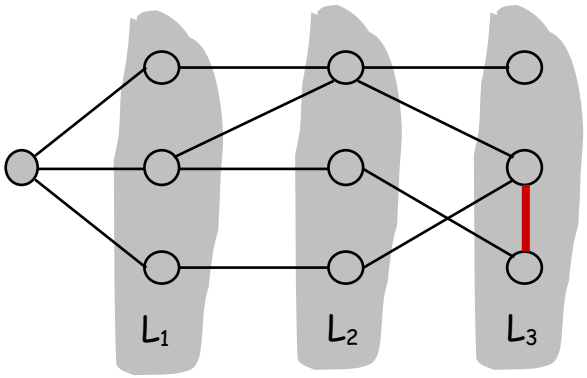
- Osservazione. Sia G un grafo connesso e siano $L_0, \ldots, L_k$ i livelli prodotti da un'esecuzione di BFS a partire dal nodo s. Può avvenire o che si verifichi la (i) o la (ii)

(i) Nessun arco di G collega due nodi sullo stesso livello

(ii) Un arco di G collega due nodi sullo stesso



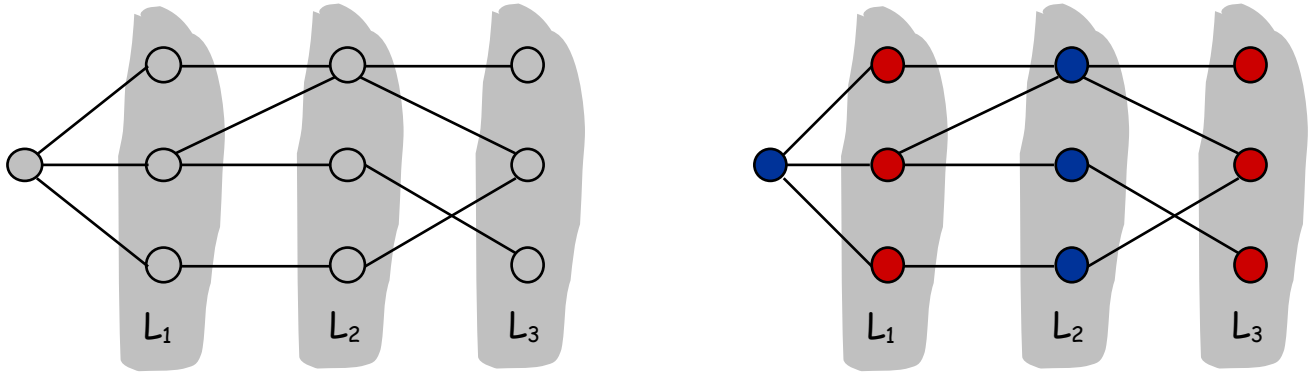
Case (i)



Case (ii)

Grafi Bipartiti

- Nel caso (i) il grafo è bipartito.
- Dim. Tutti gli archi collegano nodi in livelli consecutivi (per la proprietà sulla distanza dei nodi adiacenti nel BFS tree). Quindi se coloro i livelli di indice pari di rosso e quelli di indice dispari di blu, ho che le estremità di ogni arco sono di colore diverso.

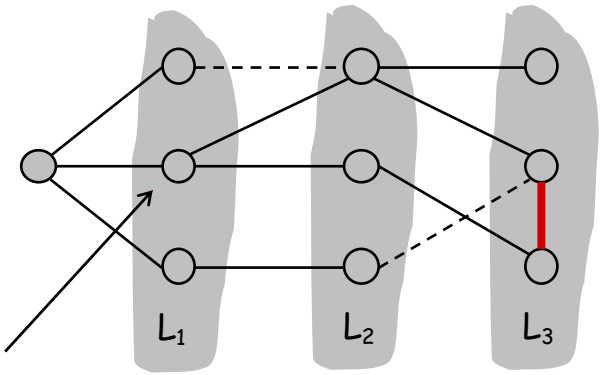


Caso (i)

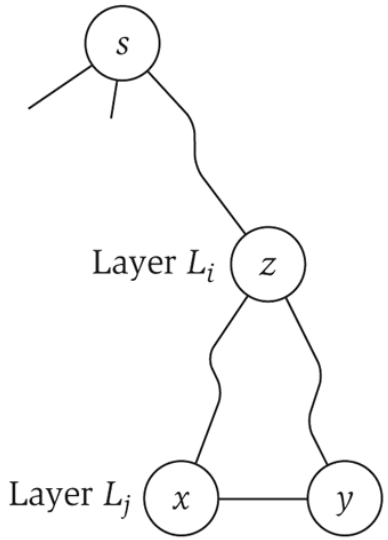
Grafi Bipartiti

Nel caso (ii) il grafo non è bipartito.

Dim. Il grafo contiene un ciclo dispari: supponiamo che esiste l'arco (u,v) tra i vertici u e v di L_i. Indichiamo con z l'antenato comune ad u e v nell'albero BFS che si trova più vicino a u e v. Sia L_j il livello in cui si trova z. Possiamo ottenere un ciclo dispari del grafo prendendo il percorso seguito dalla BFS da z a u (i-j archi), quello da z a v (i-j archi) e l'arco (u,v). In totale il ciclo contiene 2(i-j)+1 archi.



Antenato comune più vicino (lowest common ancestor)



Caso (ii)

Algoritmo che usa BFS per determinare se un grafo è bipartito

- Modifichiamo BFS come segue:

- Usiamo un array Color per assegnare i colori ai nodi
- Ogni volta che aggiungiamo un nodo v alla lista L[i+1] poniamo Color[v] uguale a rosso se i+1 è pari e uguale a blu altrimenti
- Alla fine esaminiamo tutti gli archi per vedere se c'è ne è uno con le estremità dello stesso colore. Se c'è concludiamo che G non è bipartito; altrimenti concludiamo che G è bipartito.

- Tempo: $O(n+m)$

Visita di grafi direzionati

- Raggiungibilità con direzione. Dato un nodo s, trova tutti i nodi raggiungibili da s.

- Il problema del più corto percorso diretto da s a t. Dati due nodi s e t, qual è la lunghezza del percorso più corto da s a t?

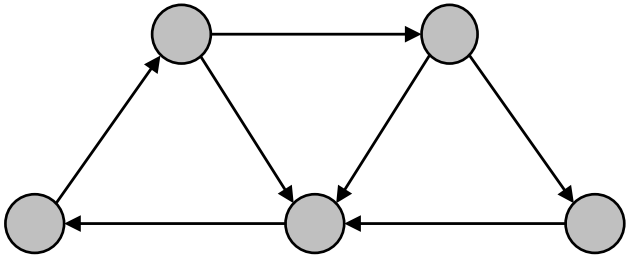
- Visita di un grafo. La BFS si estende naturalmente ai grafi direzionati.
  - Quando si esaminano gli archi incidenti su un certo vertice u, si considerano solo quelli uscenti da u.

- Web crawler. Comincia dalla pagina web s. Trova tutte le pagine raggiungibili a partire da s, sia direttamente che indirettamente.

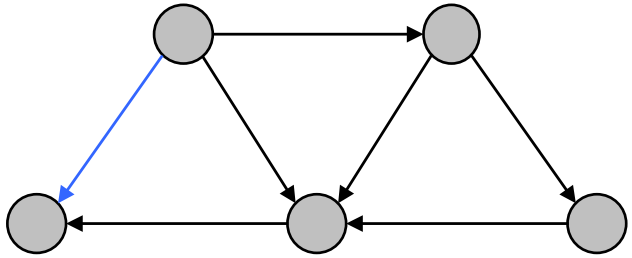
Connettività forte

- Def. I nodi u e v sono mutualmente raggiungibili se c'è un percorso da u a v e anche un percorso da v a u.

- Def. Un grafo è fortemente connesso se per coppia di nodi u e v , si ha che i nodi u e v sono mutualmente raggiungibili



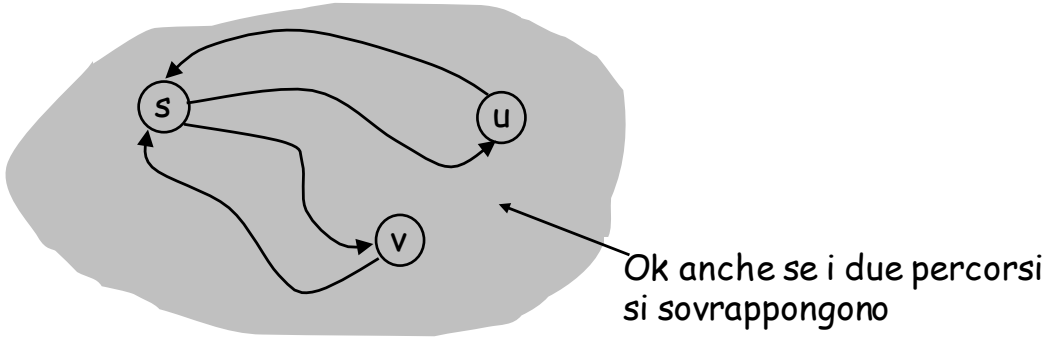
Fortemente connesso



Non fortemente connessog

Connettività forte

- Lemma. Sia s un qualsiasi nodo di G. G è fortemente connesso se e solo se ogni nodo è raggiungibile da s ed s è raggiungibile da ogni nodo.
- Dim. $\Rightarrow$ Segue dalla definizione.
- Dim. $\Leftarrow$ Dati due qualsiasi nodi u e v di G, un percorso da u a v si ottiene concatenando il percorso da u ad s con il percorso da s a v. Un percorso da v ad u si ottiene concatenando il percorso da v ad s con il percorso da s ad u.



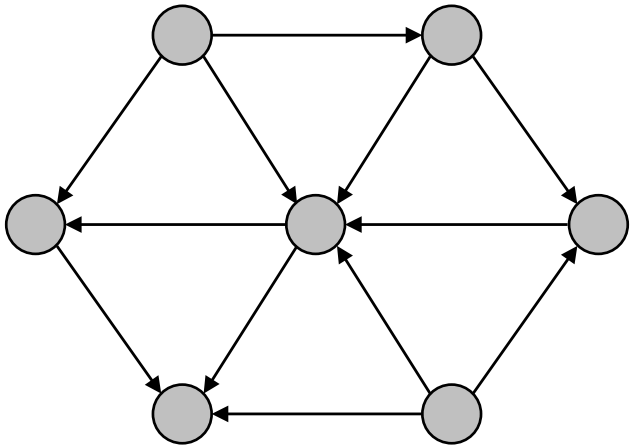
Algoritmo per la connettività forte

- Teorema. Si può determinare se G è fortemente connesso in tempo O(m + n) time.
- Dim.
  - Prendi un qualsiasi nodo s.
  - Esegui la BFS con sorgente s in G.
  - Crea il grafo G^{rev} invertendo la direzione di ogni arco in G
  - Esegui la BFS con sorgente s in G^{rev}.
  - Restituisci true se e solo se tutti i nodi di G vengono raggiunti in entrambe le esecuzioni della BFS.
  - La correttezza segue dal lemma precedente: basta osservare che
    - La prima esecuzione trova i percorsi da s a tutti gli altri nodi
    - La seconda esecuzione trova i percorsi da tutti gli altri nodi ad s perchè avendo invertito gli archi un percorso da s a u è di fatto un percorso da u ad s nel grafo di partenza.

Grafi direzionati aciclici (DAG)

- Def. Un DAG è un grafo direzionato che non contiene cicli direzionati

- Possono essere usati per esprimere vincoli di precedenza o dipendenza: l'arco (v_i, v_j) indica che v_i deve precedere v_j o che v_j dipende da v_i
- Infatti generalmente i grafi usati per esprimere i suddetti vincoli sono privi di clicli
- Esempio. Vincoli di precedenza: grafo delle propedeuticità degli esami

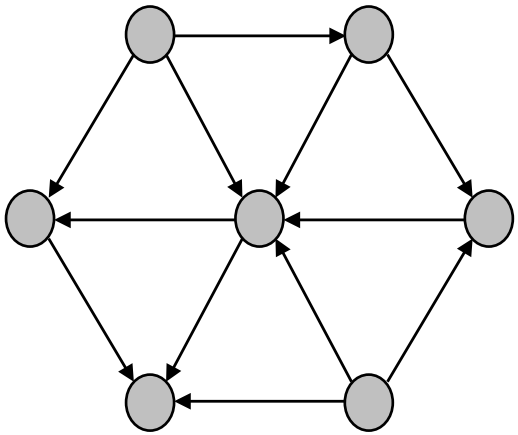


Un DAG G

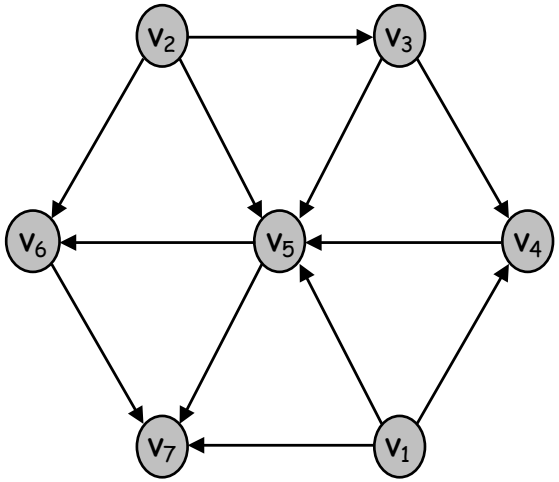
Ordine topologico

- Def. Un ordinamento topologico di un grafo direzionato $G = (V, E)$ è una etichettatura dei suoi nodi $v_1, v_2, \ldots, v_n$ tale che per ogni arco (v_i, v_j) si ha $i < j$. Detto in un altro modo, se c'è l'arco (u,w) in G, allora il vertice u precede il vertice w nell'ordinamento (tutti gli archi puntano in avanti nell'ordinamento).

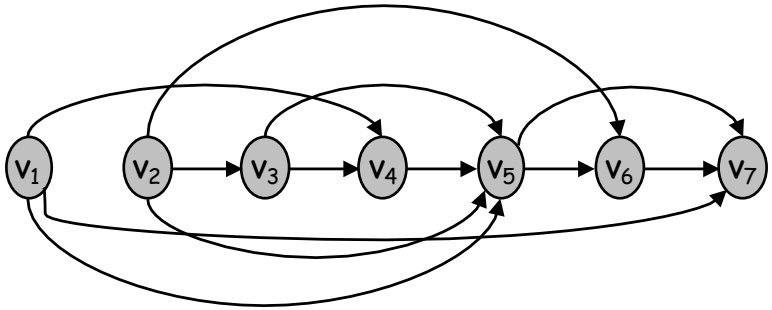
- Esempio. Nel caso in cui un grafo direzionato G rappresenti le propedeuticità degli esami, un ordinamento topologico indica un possibile ordine in cui gli esami possono essere sostenuti dallo studente.



Un DAG G



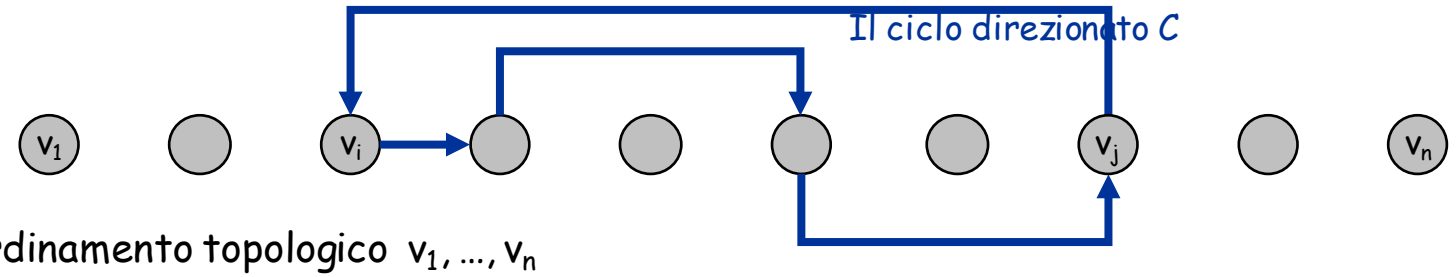
Un ordinamento topologico di G



Un modo diverso di ridisegnare G in modo da evidenziare l' ordinamento topologico di G

DAG e ordinamento topologico

- **Lemma.** Sia G un grafo direzionato. Se G ha un ordinamento topologico allora G è un DAG.
- **Dim.** (per assurdo)
  - Supponiamo che G sia un grafo direzionato e che abbia un ordinamento $v_1, \ldots, v_n$. Supponiamo per assurdo che G non sia un DAG ovvero che abbia un ciclo direzionato C. Vediamo cosa accade.
  - Consideriamo i nodi che appartengono a C e tra questi sia v_i quello con indice più piccolo e sia v_j il vertice che precede v_i nel ciclo C. Ciò ovviamente implica che (v_j, v_i) è un arco.
  - Per come abbiamo scelto i, abbiamo **i < j**.
  - D'altra parte, siccome (v_j, v_i) è un arco e $v_1, \ldots, v_n$ è un ordinamento topologico allora, deve essere **j < i**, che è una contraddizione al fatto che G contiene un ciclo.



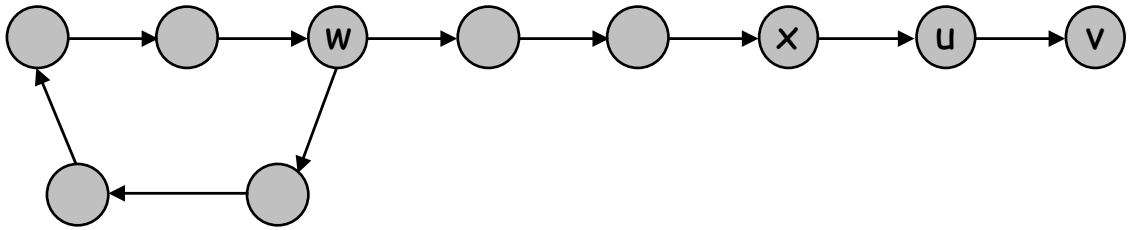
L'ordinamento topologico $v_1, \ldots, v_n$

DAG e ordinamento topologico

- Abbiamo visto che se G ha un ordinamento topologico allora G è un DAG.
- Domanda. E' vera anche l'implicazione inversa? Cioè dato un DAG, è sempre possibile trovare un suo ordinamento topologico?
- E se sì, come trovarlo?

DAG e ordinamento topologico

- Lemma. Se G è un DAG allora G ha un nodo senza archi entranti
- Dim. (per assurdo)
  - Supponiamo che G sia un DAG e che ogni nodo di G abbia almeno un arco entrante. Vediamo cosa succede.
  - Prendiamo un qualsiasi nodo v e cominciamo a seguire gli archi in senso contrario alla loro direzione a partire da v. Possiamo farlo perchè ogni nodo ha un arco entrante: v ha un arco entrante (u,v), il nodo u ha un arco (x,u) e così via.
  - Possiamo continuare in questo modo per quante volte vogliamo. Immaginiamo di farlo per n o più volte. Così facendo attraversiamo a ritroso almeno n archi e di conseguenza passiamo per almeno n+1 vertici. Ciò vuol dire che c'è un vertice w che viene incontrato almeno due volte e quindi deve esistere un ciclo direzionato C che comincia e finisce in w



DAG e ordinamento topologico

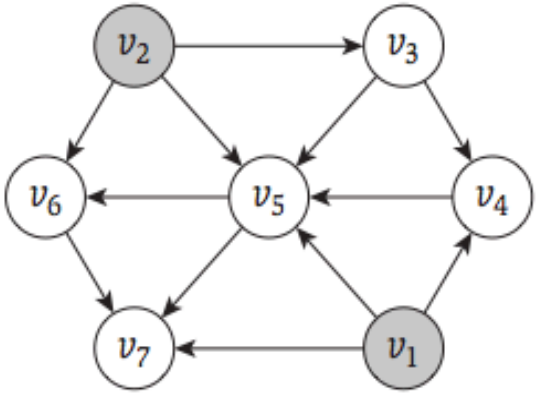
- Lemma. Se G è un DAG, G ha un ordinamento topologico.
- Dim. (induzione su n)
  - Caso base: vero banalmente se n = 1.
  - Passo induttivo: supponiamo asserto del lemma vero per DAG con n≥1 nodi
  - Dato un DAG con n+1 > 1 nodi, prendiamo un nodo v senza archi entranti (abbiamo dimostrato che un tale nodo deve esistere).
  - G - { v } è un DAG, in quanto cancellare un nodo non introduce clicli nel grafo.
  - Poiché G - { v } è un DAG con n nodi allora, per ipotesi induttiva, G - { v } ha un ordinamento topologico.
  - Consideriamo l'ordinamento dei nodi di G che si ottiene mettendo v all'inizio dell'ordinamento e aggiungendo gli altri nodi nell'ordine in cui appaiono nell'ordinamento topologico di G - { v }.
  - Siccome v non ha archi entranti quello che si ottiene è un ordinamento topologico (tutti gli archi puntano in avanti).

Algoritmo per l'ordinamento topologico

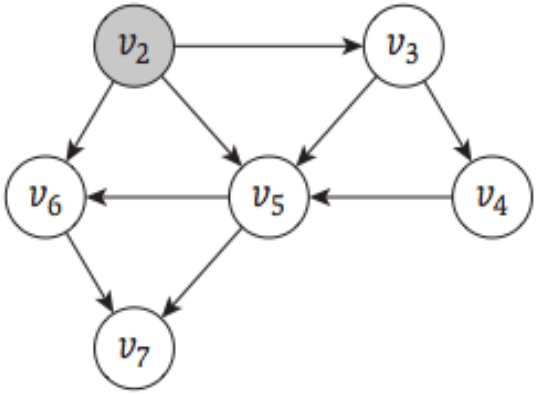
- La dimostrazione per induzione che abbiamo appena visto suggerisce un algorimo ricorsivo per trovare l' ordinamento topologico di un DAG.

To compute a topological ordering of G:
Find a node v with no incoming edges and order it first
Delete v from G
Recursively compute a topological ordering of $G-\{v\}$
  and append this order after v

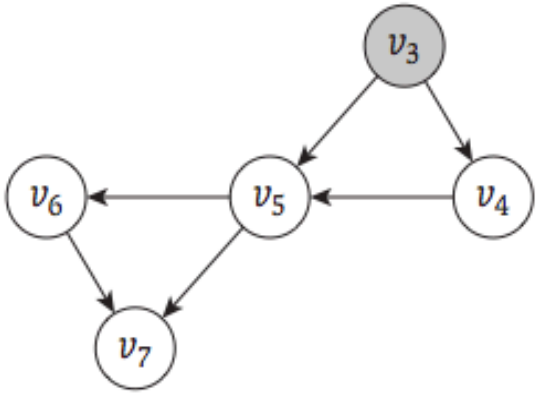
Algoritmo per l'ordinamento topologico



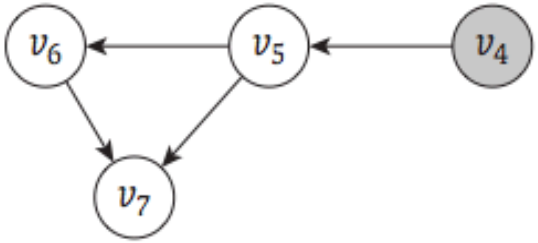
(a)



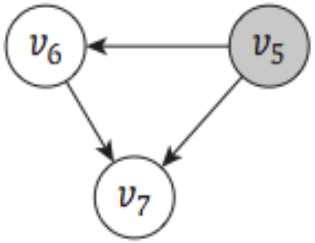
(b)



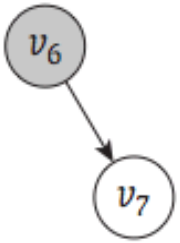
(c)



(d)



(e)



(f)

Algoritmo per l'ordinamento topologico : analisi dell'algoritmo

- Trovare un nodo senza archi entranti richiede $O(n)$
- Cancellare un nodo v da G richiede tempo proporzionale al numero di archi uscenti da v che è **al più** deg(v)=$O(n)$
- Se consideriamo tutte le n chiamate ricorsive il tempo è $O(n^2)$

To compute a topological ordering of G:
Find a node v with no incoming edges and order it first
Delete v from G
Recursively compute a topological ordering of $G-\{v\}$
  and append this order after v

Algoritmo per l'ordinamento topologico : analisi dell'algoritmo

- Possiamo anche scrivere la relazione di ricorrenza

$$T(n) \leq \begin{cases} c \quad \text{per } n=1 \\ T(n-1)+c'n \text{ per } n>1 \end{cases}$$

Lavoro ad ogni chiamata ricorsiva è $O(n+dev(v))=O(n)$, dove v è il nodo rimosso da G

che ha soluzione $T(n)=O(n^2)$

Metodo iterativo

$T(n) \leq T(n-1)+c'n \leq T(n-2)+c'(n-1)+c'n \leq T(n-3)+c'(n-2)+c'(n-1)+c'n \leq \ldots$

$\leq c+c'2+\ldots+nc'=c+c'n(n+1)/2 -c' = O(n^2)$

Metodo di sostituzione. Ipotizziamo $T(n) \leq Cn^2$ per $n \geq n_0$, dove C ed n_0 sono costanti positive da determinare

Base induzione: $T(1)) \leq c \leq 1^2C$ se $C \geq c$

Passo induttivo. $T(n) \leq T(n-1)+c'n \leq C(n-1)^2+c'n = Cn^2+C-2Cn+c'n \leq Cn^2$ se $C \geq c'$

Basta prendere $C=\max\{c,c'\}$ e $n_0=1$

Algoritmo per l'ordinamento topologico con informazioni aggiuntive

- Il bound $O(n^2)$ non è molto buono se il grafo è sparso, cioè se il numero di archi è molto più piccolo di n^2
- Possiamo ottenere un bound migliore?

- Per ottenere un bound migliore occorre usare un modo efficiente per individuare un nodo senza archi entranti ad ogni chiamata ricorsiva

- Si procede nel modo seguente:
- Un nodo si dice attivo se non è stato ancora cancellato
- Occorre mantenere le seguenti informazioni:
- per ciascun vertice attivo w
  - count[w] = numero di archi entranti in w provenienti da nodi attivi.
  - S = insieme dei nodi attivi che non hanno archi entranti provenienti da altri nodi attivi.

Algoritmo per l'ordinamento topologico con informazioni aggiuntive: analisi

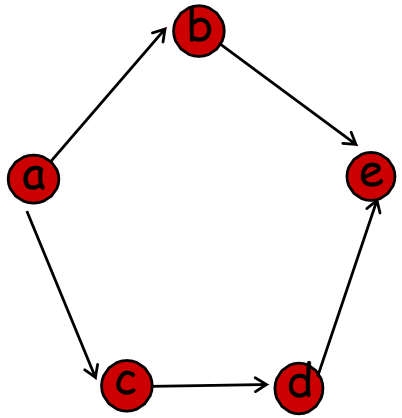
- Teorema. L'algoritmo trova l'ordinamento topologico di un DAG in tempo $O(m + n)$.
- Dim.
  - Inizializzazione. Richiede tempo $O(m + n)$ in quanto
    - Inizialmente tutti i nodi sono attivi per cui S consiste dei nodi di G senza archi entranti➔ basta scandire tutti i nodi una sola volta per inizializzare S.
  - I valori di count[w] vengono inizializzati scandendo tutti gli archi e incrementando count[w] per ogni arco entrante in w ➔ basta scandire tutti gli archi una sola volta.
  - Aggiornamento. Per trovare il nodo v da cancellare basta prendere un nodo da S. Per cancellare v occorre
    1. Cancellare v da S. Tempo $O(1)$
    2. Decrementare count[w] per ogni arco (v,w). Se count[w]=0 allora occorre aggiungere w a S. Tempo $O(\deg(v))$.

    I passi 1. e 2. vengono eseguiti una volta per ogni vertice ➔tutti gli aggiornamenti vengono fatti in

$$\sum_{u \in V} O(1) + \sum_{u \in V} O(\deg(u)) = O(n) + O(m) = O(n+m)$$

Esercizio 1 su grafi

- Fornire tutti gli ordinamenti topologici del grafo sottostante



- Soluzione. Potremmo esaminare le 5!= 120 possibili permutazioni….
- Ragioniamo: il primo nodo dell'ordinamento non deve avere archi entranti, l'ultimo non deve avere archi uscenti. Gli unici nodi che rispettivamente soddisfano questi requisiti sono a ed e. Quindi ogni ordinamento topologico deve cominciare con a e finire con e. In quanti modi possono essere sistemati gli altri nodi? Osserviamo che l'arco (c,d) implica che c precede d in quasiasi ordinamento topologico mentre b può trovarsi in una qualsiasi posizione tra a ed e. In totale, ci sono quindi 3 ordinamenti topologici
- a b c d e , a c b d e, a c d b e

Esercizio 2 su Grafi

1. Due robot, il primo parte dal punto a e deve arrivare nel punto c mentre il secondo parte dal punto b e deve arrivare nel punto d.
2. In ogni momento i due robot devono trovarsi a distanza maggiore di r per evitare interferenze tra i trasmettitori che utilizzano per comunicare con la stazione base.

- La pianta del piano in cui si muovono i robot può essere rappresentata da un grafo G in cui i nodi rappresentano i vari punti del piano e gli archi uniscono due punti adiacenti
- Se non vi fosse il punto 2., il problema consisterebbe semplicemente nell'assegnare a ciascun robot degli istanti in cui il robot si muove da una certa posizione ad una adiacente e l'altro sta fermo, in modo tale che alla fine il primo robot venga a trovarsi in c e il secondo in d.

Continua nella prossima slide

Esercizio 2 su Grafi

- Il punto 2 impone che le rispettive posizioni dei robot siano sempre a distanza maggiore di r
- Per poter risolvere il problema costruiamo un grafo H come segue:
- I nodi di H sono coppie (u,v), dove u è la posizione del primo robot e v quella del secondo robot
- Due nodi (u,v) e (u',v') di H sono collegati se u=u' e (v,v') è un arco in G oppure v=v' e (u,u') è un arco in G. Questo perché ad ogni passo, uno dei due robot si sposta in una posizione adiacente a quella in cui si trovava prima mentre l'altro robot rimane fermo.
- Un percorso in H dalla configurazione (a,b) alla configurazione (c,d) è una sequenza di passi che porta i robot nelle posizioni desiderate

Continua nella prossima slide

Esercizio 2 su Grafi

- Un percorso in H dalla configurazione (a,b) alla configurazione (c,d) però non rispetta necessariamente il vincolo 2. che impone che i due robot possono trovarsi nella configurazione (u,v) solo se la distanza tra u e v è maggiore di r.
- Per essere certi di trovare un percorso in H che soddisfi il vincolo 2., eliminiamo da H tutti i nodi (u,v) con u e v che si trovano a distanza al più r. Chiamiamo H' il grafo risultante.
- Una volta costruito H', eseguiamo una visita BFS o DFS a partire dal nodo sorgente (a,b). Se nel visitare H' raggiungiamo (c,d) allora vuol dire che è possibile far arrivare i due robot nei punti desiderati senza che vi sia mai interferenza tra i loro trasmettitori.

Continua nella prossima slide

Esercizio 2 su Grafi

- Analisi dell'algoritmo:
- La creazione di H richiede tempo $O(n^2)$ per creare i nodi e $O(n^3)$ per creare gli archi:
- Vi sono al più n^2 nodi
- Ogni nodo (u,v) ha un numero di archi uscenti che arrivano in un nodo della forma (u,v') pari al numero di archi uscenti da v in G e ha un numero di archi uscenti che arrivano in un nodo della forma (u',v) pari al numero di archi uscenti da u in G. Ne consegue che per ogni nodo (u,v) vi è un a numero di archi uscenti pari al più deg(u)+deg(v) ≤ n. In totale vi sono quindi n^3 archi in H.
- Per cancellare da H i nodi (u,v) con u e v a distanza al più r, eseguiamo BFS(u) in G per ogni nodo u e creiamo la lista di tutti i nodi a distanza al più r da u. Per ogni nodo v in questa lista, rimuoviamo il nodo (u,v) da H. La cancellazione di ciascun nodo (u,v) richiede tempo O(n) perché ci sono al più n archi incidenti (entranti e uscenti) su (u,v). In totale la costruzione di H' a partire da H, richiede $O(n(n+m))+O(n^3)$
- Il tempo per la visita di H' richiede $O(n^2+n^3)= O(n^3)$

Esercizio 2 Cap 3

- Fornire un algoritmo che, dato un grafo non direzionato G, scopre se G contiene cicli e in caso affermativo produce in output uno dei cicli. L'algoritmo deve avere tempo di esecuzione O(n+m)

- Soluzione. Si esegua una visita BFS sul grafo. Se il grafo non è connesso si eseguono più visite, una per componente connessa. Se al termine gli alberi BFS contengono tutti gli archi allora G non contiene cicli. In caso contrario, c'è almeno un arco (x,y) che non fa parte degli alberi BFS. Consideriamo l'albero BFS T in cui si trovano x e y e sia z l'antenato comune più vicino a x e y (LCA di x e y). L'arco (x,y) insieme ai percorsi tra z e x e quello tra z e y forma un ciclo

Esercizio 3 Cap. 3

- Modificare l'algoritmo per l'ordinamento topologico di un DAG in modo tale che se il grafo direzionato input non è un DAG l'algoritmo riporta in output un ciclo che fa parte del grafo.

- Soluzione.
- Caso 1. All'inizio di ogni chiamata ricorsiva dell'algoritmo per l'ordinamento topologico, l'insieme S contiene almeno un nodo. In questo caso riusciamo ad ottenere un ordinamento topologico perché ogni nodo cancellato v non ha archi entranti che provengono dai nodi che sono ancora attivi e che quindi saranno posizionati nell'ordinamento dopo v. Il Lemma ci dice che se il grafo ha un ordinamento topologico allora il grafo è un DAG.
- Caso 2. All'inizio di una certa chiamata ricorsiva, S è vuoto. In questo caso il grafo formato dai nodi attivi non è un DAG per il lemma che dice che un DAG ha almeno un nodo senza archi entranti. Il ciclo è ottenuto percorrendo a ritroso gli archi a partire da un qualsiasi nodo attivo v fino a che non incontriamo uno stesso nodo w due volte.

Continua nella prossima slide

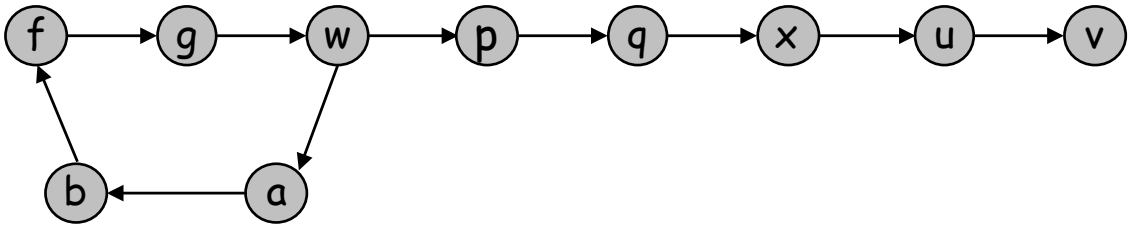
Esercizio 3 Cap. 3

- Basta quindi modificare l'algoritmo in modo che se all'inizio di una chiamata ricorsiva si ha che S è vuoto allora l'algoritmo sceglie un nodo attivo v e comincia a percorrere gli archi a ritroso a partire da v: si sceglie un arco (x,v) nella lista degli archi entranti in v, poi si sceglie un arco (y,x) nella lista degli archi entranti in x e così via.
- Ogni volta che viene attraversato un arco (p,q) a ritroso, il nodo p raggiunto viene inserito all'inizio di una lista a doppi puntatori ed etichettato come visitato.
- Se ad un certo punto si raggiunge un nodo w già etichettato come visitato, l'algoritmo interrompe questo percorso all'indietro e cancella dalla lista tutti i nodi a partire dalla fine della lista fino a che incontra per la prima volta w.
- I nodi restanti nella lista formano un ciclo direzionato che comincia e finisce in w.
- Tempo $O(n+m)$ in quanto l'algoritmo per l'ordinamento ha costo $O(n+m)$ e il costo aggiuntivo per trovare il ciclo è $O(n)$.

Continua nella prossima slide

Esercizio 3 Cap. 3

- Esempio di grafo con ciclo



- Se cominciamo il cammino a ritroso a partire da v, la lista dei nodi attraversati è w a b f g w p q x u v (aggiungiamo ogni nodo attraversato all'inizio della lista). Non appena incontriamo la seconda occorrenza di w, ci fermiamo e cancelliamo gli ultimi 5 nodi della lista scandendo la lista a partire dalla fine. I nodi che rimangono nella lista formano il ciclo w a b f g w.

Esercizio 9 Cap. 3

- Sia G un grafo non direzionato con n nodi. Dimostriamo che se G contiene due nodi s e t a distanza maggiore di n/2 allora esiste un nodo v diverso da s e t tale che se eliminiamo v da G allora non è più possibile andare da s a t. In altre parole tutti i percorsi tra s e t passano per v.

- **Applicazione**: vogliamo capire quanto una rete sia vulnerabile ai guasti. Il risultato che l'esercizio ci chiede di dimostrare suggerisce che se in una rete ci sono nodi s e t molto distanti tra di loro allora il malfunzionamento di un solo nodo compromette la comunicazione tra s e t.

- **Soluzione**: Immaginiamo di eseguire una BFS con s come sorgente e sia L_j il livello in cui si trova t. Per ipotesi s e t sono a distanza > n/2 e quindi j>n/2.

  - Deduciamo che esiste un livello L_i , con $0 < i < j$, tale che L_i contiene un solo nodo. Infatti i livelli con indice in [1,j-1] sono almeno n/2 e se ognuno di essi contenesse almeno due nodi allora il numero di nodi di G sarebbe maggiore di n. Chiamiamo v l'unico nodo di L_i.

Esercizio 9 Cap. 3

- Osserviamo che

1) Per raggiungere t a partire da s occorre ad un certo punto raggiungere un nodo che si trova fuori da $L_0, L_1, \ldots, L_{i-1}$ in quanto t non si trova in nessuno dei livelli $L_0, L_1, \ldots, L_{i-1}$.
2) Ogni arco che incide su un nodo u che si trova in uno dei livelli $L_0, L_1, \ldots, L_{i-1}$ connette u ad un nodo che si trova o in uno dei livelli $L_0, L_1, \ldots, L_{i-1}$ o in L_i. Abbiamo infatti dimostrato che ogni arco di G connette due nodi che si trovano o nello stesso livello o in livelli consecutivi.

La 1) e la 2) implicano che qualsiasi percorso che da s va a t deve passare per un nodo di L_i. Siccome L_i contiene solo v allora tutti i percorsi da s a t passano per v.