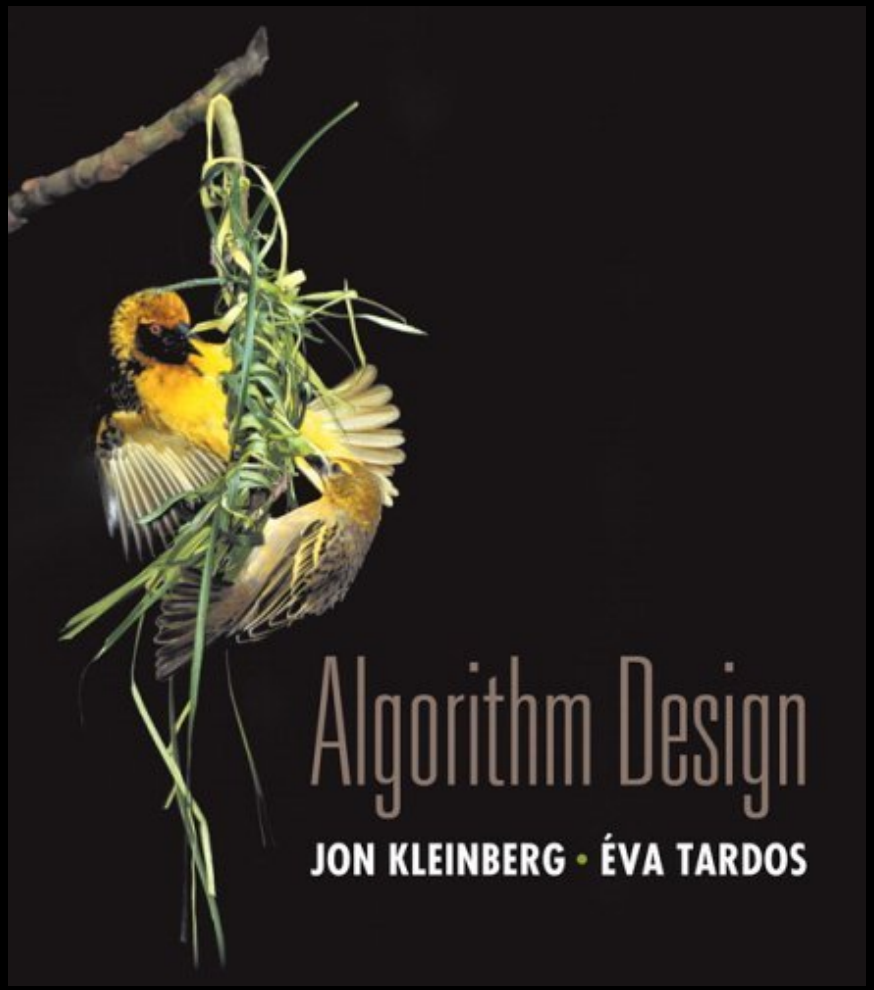


Chapter 3

Graphs

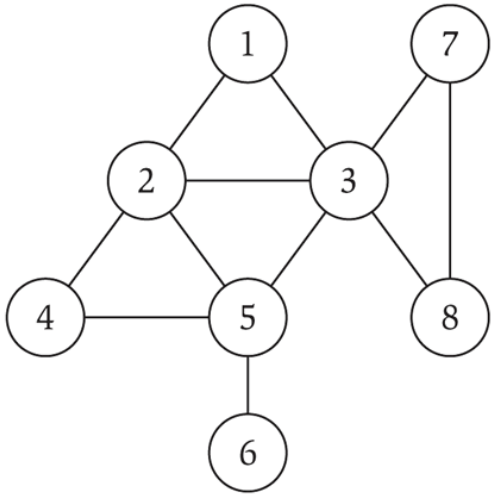
PEARSON
Addison Wesley

3.1 Basic Definitions and Applications

Undirected Graphs

Undirected graph. $G = (V, E)$

- V = nodi (anche vertici).
- E = archi tra coppie di nodi.
- Modella relazioni tra coppie di oggetti.
- Parametri della grandezza di n grafo: $n = |V|$, $m = |E|$.



V = { 1, 2, 3, 4, 5, 6, 7, 8 }

E = { {1,2}, {1,3}, {2,3}, {2,4}, {2,5}, {3,5}, {3,7}, {3,8}, {4,5}, {5,6} }

n = 8

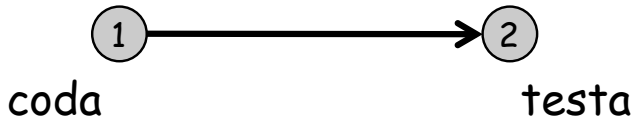
m = 11

Grafi Diretti

Grafo diretto. $G = (V, E)$

- V = nodi (anche vertici).
- E = archi tra coppie di nodi.
- Modella relazioni tra coppie di oggetti.
- Parametri della grandezza di n grafo: $n = |V|$, $m = |E|$.

(1,2)



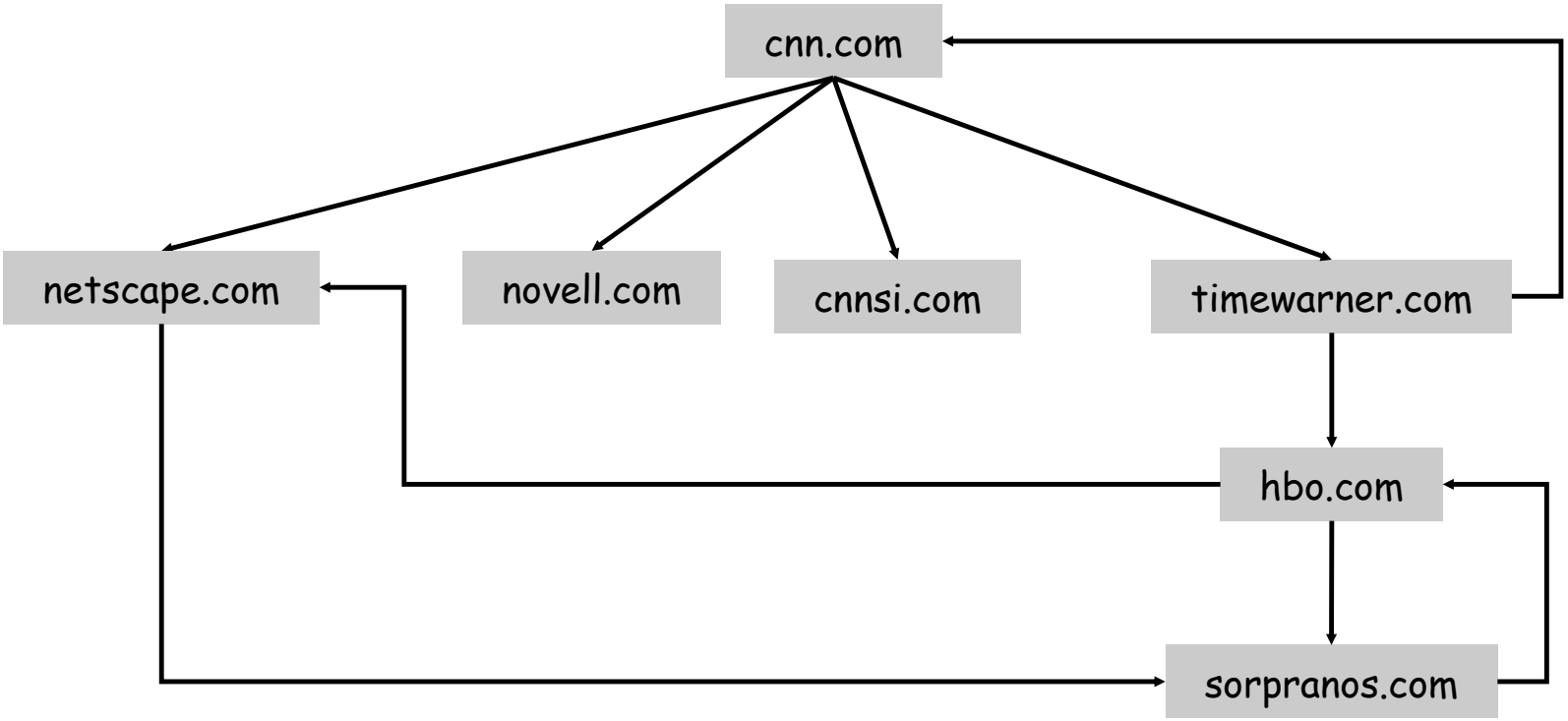
Esempi Grafi

Grafo	Nodi	Archi
trasporti	città, aeroporti	strade, rotte
comunicazioni	computer	cavi fibra ottica
World Wide Web	pagine web	hyperlink
social network	persone	relazioni
food web	specie	predatore-preda
sistemi software	funzioni	chiamate funzioni
scheduling	task	vincoli precedenza
circuiti	porte	fili

World Wide Web

Grafo del Web.

- Nodo: pagina web.
- Arco: hyperlink da una pagina ad un' altra.

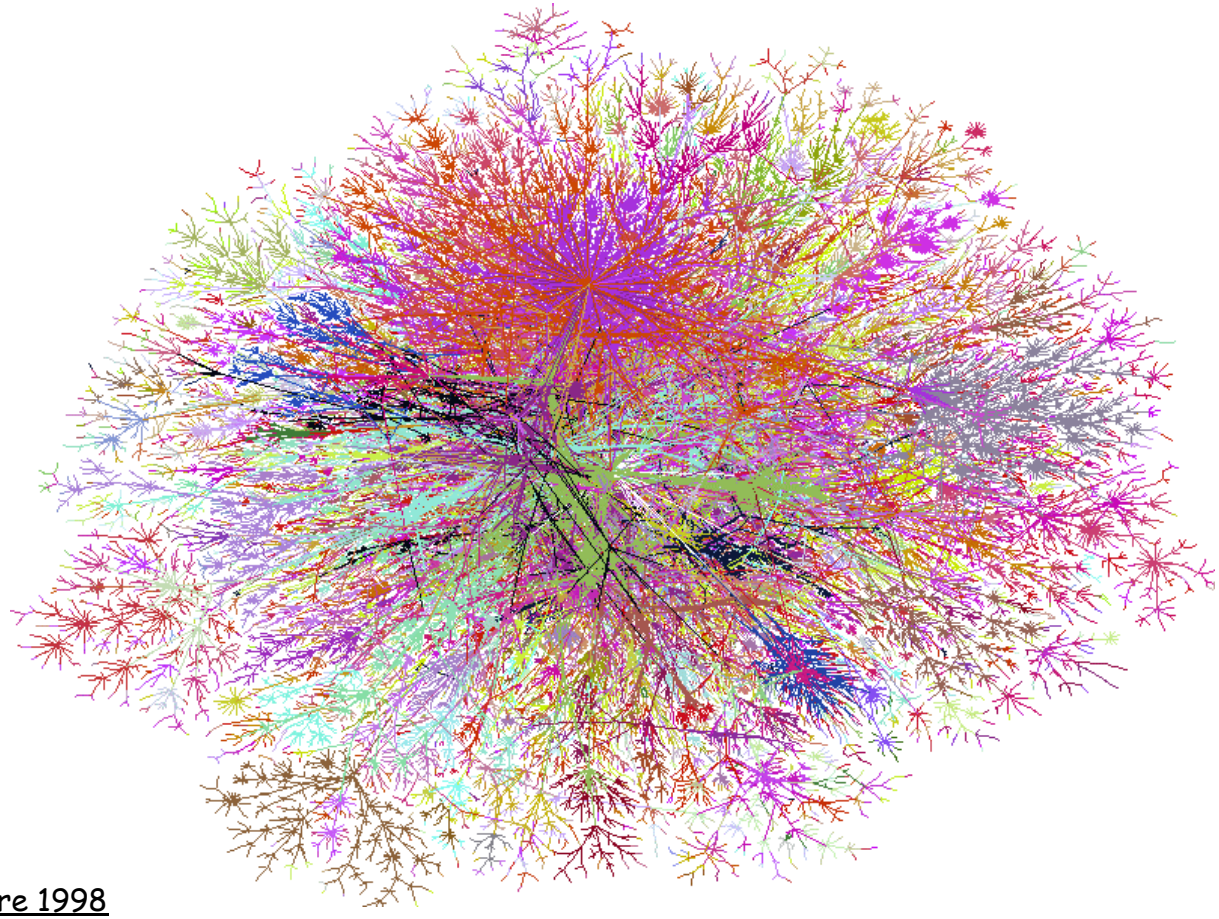


World Wide Web

Albero con 100.000 nodi

traceroute-style path probes

Colori: distanza, indirizzo IP, regione geografica



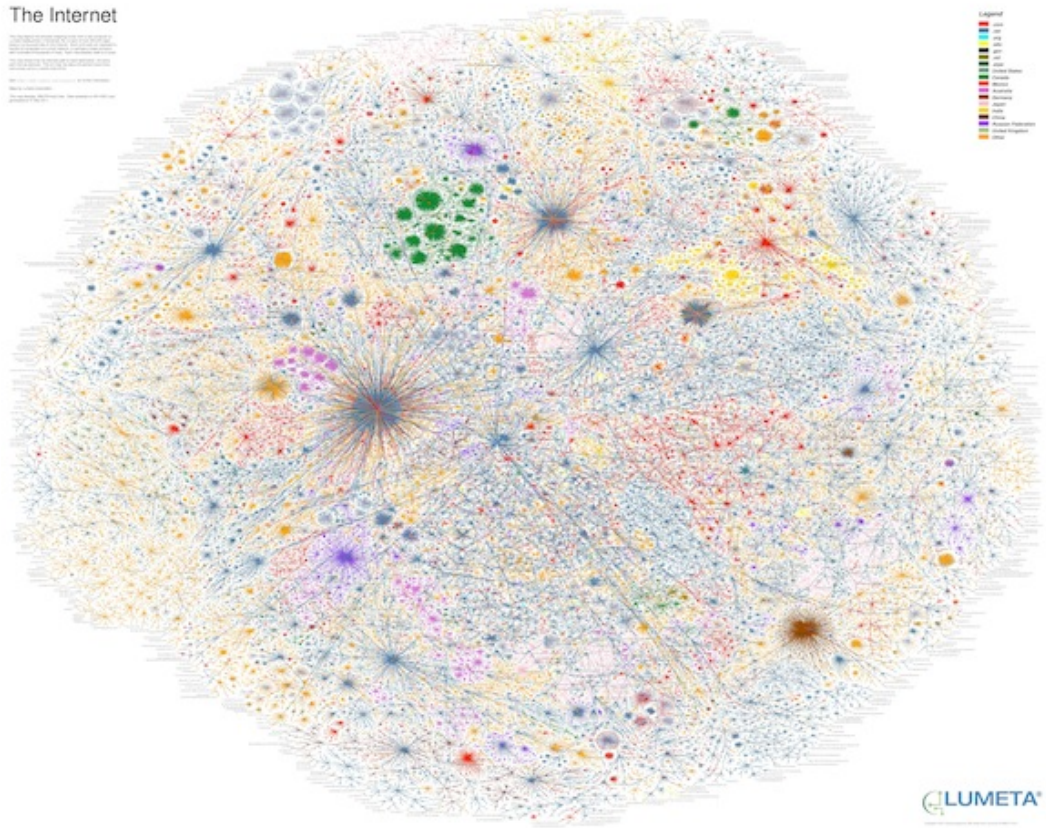
Wired Magazine, dicembre 1998
http://www.cheswick.com/ches/map/

World Wide Web

Albero con 359.075 nodi

traceroute-style path probes

Colori: distanza, indirizzo IP, regione geografica



21 marzo 2011

http://www.lumeta.com/

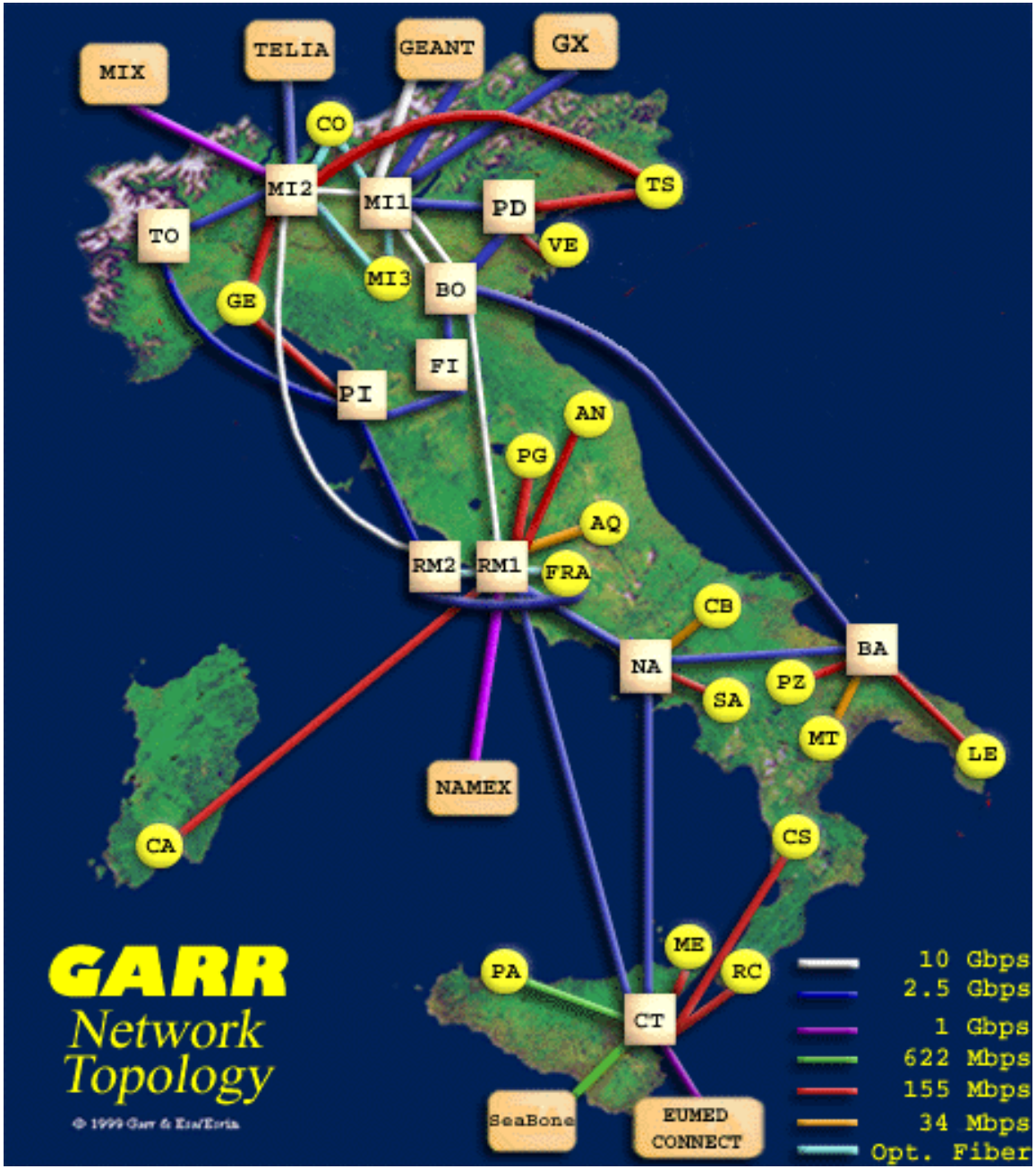
Rete terroristi dell' 11 settembre

Grafo di una social network.

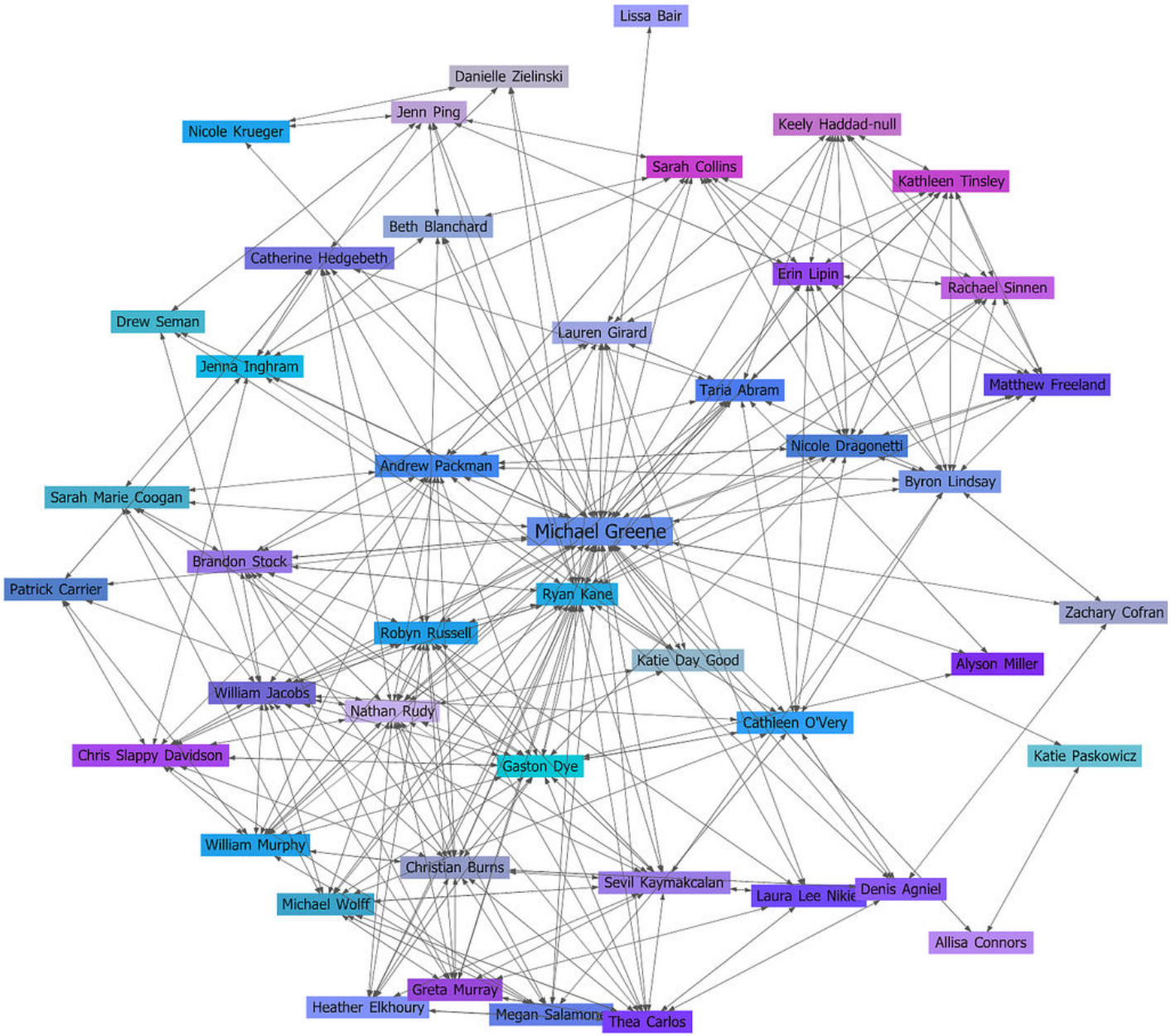
- Nodo: persona.
- Arco: relazioni tra 2 pesone.

Reference: Valdis Krebs, http://www.firstmonday.org/issues/issue7_4/krebs

Rete GARR


MIX
TELIA
GEANT
GX
CO
MI2
MI1
PD
TS
TO
VE
MI3
BO
GE
FI
PI
AN
PG
AQ
RM2
RM1
FRA
CB
BA
NA
PZ
SA
MT
LE
NAMEX
CA
CS
ME
PA
RC
CT
SeaBone
EUMED
CONNECT
GARR
Network
Topology
10 Gbps
2.5 Gbps
1 Gbps
622 Mbps
155 Mbps
34 Mbps
Opt. Fiber

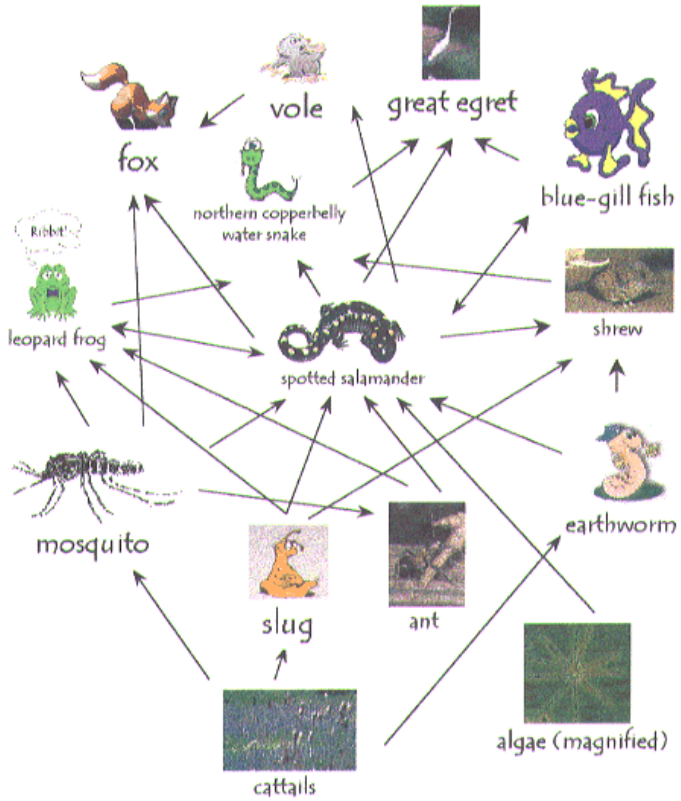
Facebook network graph



Ecological Food Web

Food web.

- Nodo = specie.
- Arco = dalla preda al predatore.



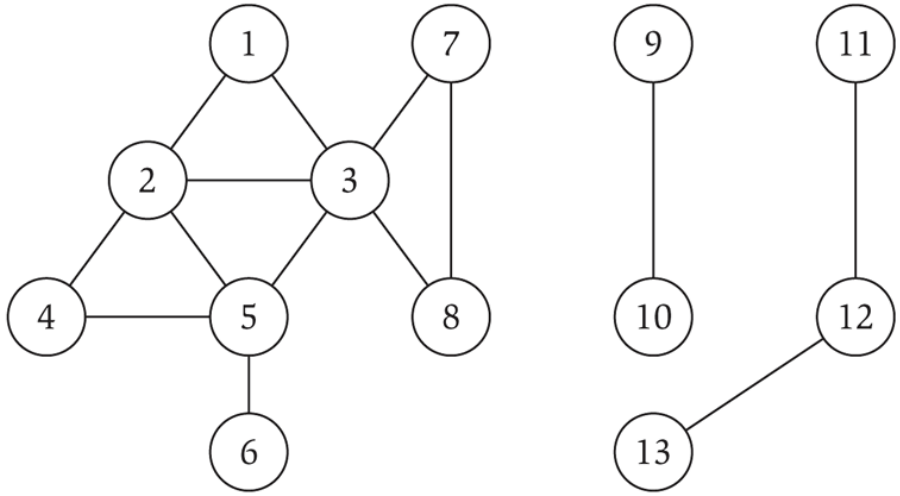
Reference: http://www.twingroves.district96.k12.il.us/Wetlands/Salamander/SalGraphics/salfoodweb.giff

Cammini e Connettività

Def. Un cammino in un grafo nondiretto $G = (V, E)$ è una sequenza P di nodi $v_1, v_2, \ldots, v_{k-1}, v_k$ con la proprietà che ogni coppia consecutiva v_i, v_{i+1} è collegata da un arco in E.

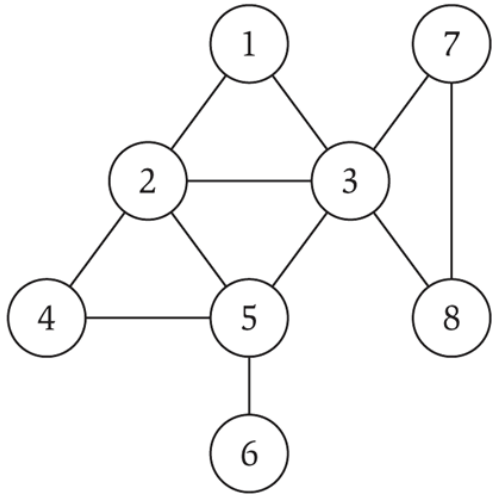
Def. Un cammino è semplice se tutti i nodi sono distinti.

Def. Un grafo nondiretto è connesso se per ogni coppia di nodi u and v, vi è un cammino tra u e v.



Distanza

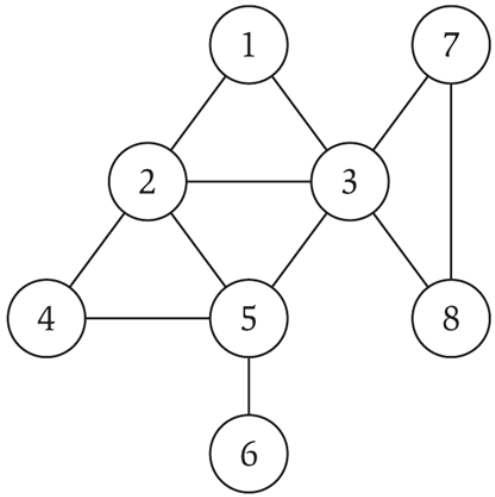
Def. La distanza tra due nodi u e v è il minimo numero di archi in un cammino u-v.



Distanza tra 4 e 3 = 2
Distanza tra 6 e 1 = 3

Cicli

Def. Un ciclo è un cammino $v_1, v_2, \ldots, v_{k-1}, v_k$ in cui $v_1 = v_k$, $k > 2$, e i primi $k-1$ nodi sono tutti distinti.



ciclo C = 1-2-4-5-3-1

Alberi

Def. Un grafo nondiretto è un albero se è connesso e non contiene cicli.

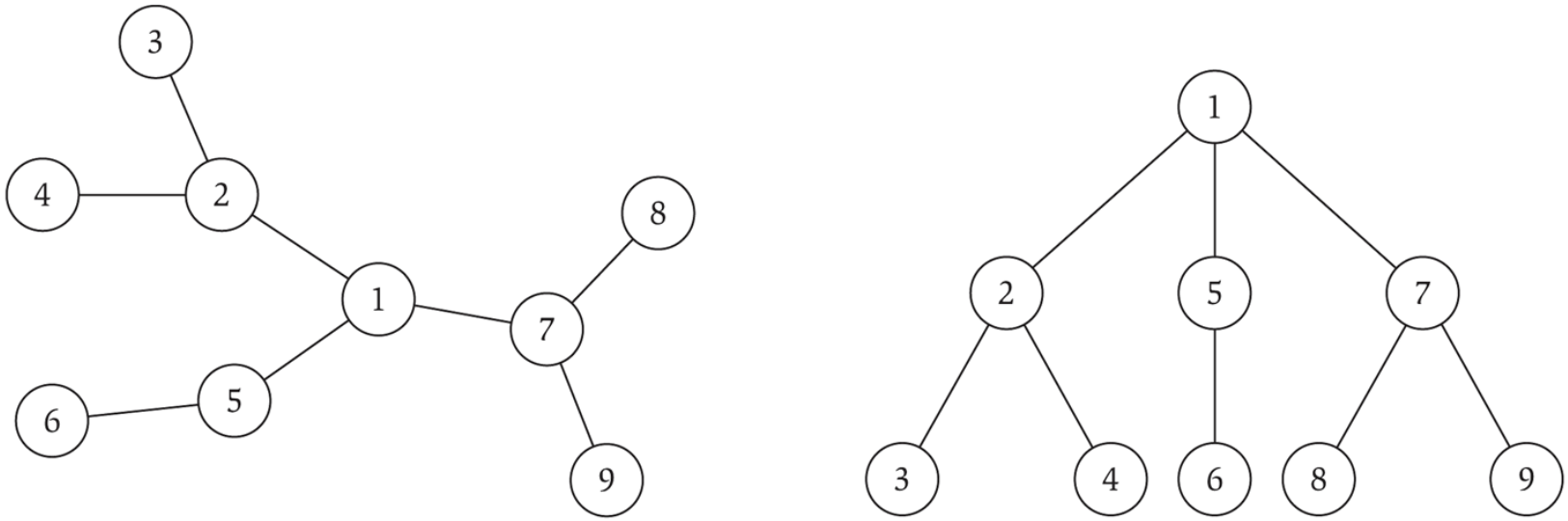


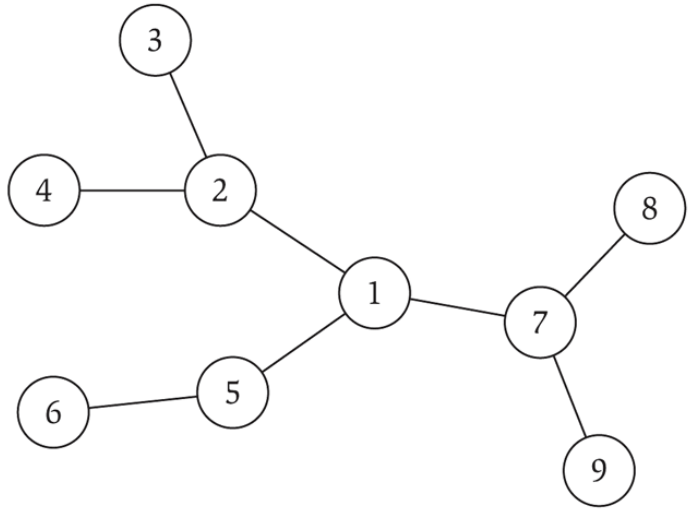
Figure 3.1 Two drawings of the same tree. On the right, the tree is rooted at node 1.

Alberi

Def. Un grafo nondiretto è un albero se è connesso e non contiene cicli.

Teorema. Sia G un grafo non diretto con n nodi. Qualunque coppia delle seguenti affermazioni implica la terza.

- G è connesso.
- G non contiene cicli.
- G ha n-1 archi.

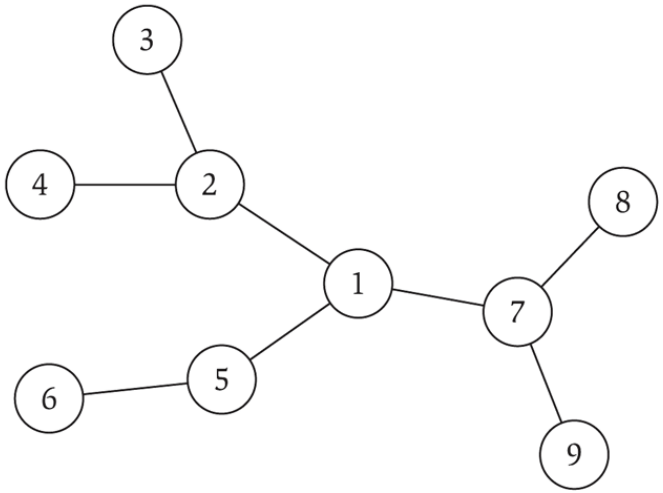


Non c'è la prova sul libro di testo

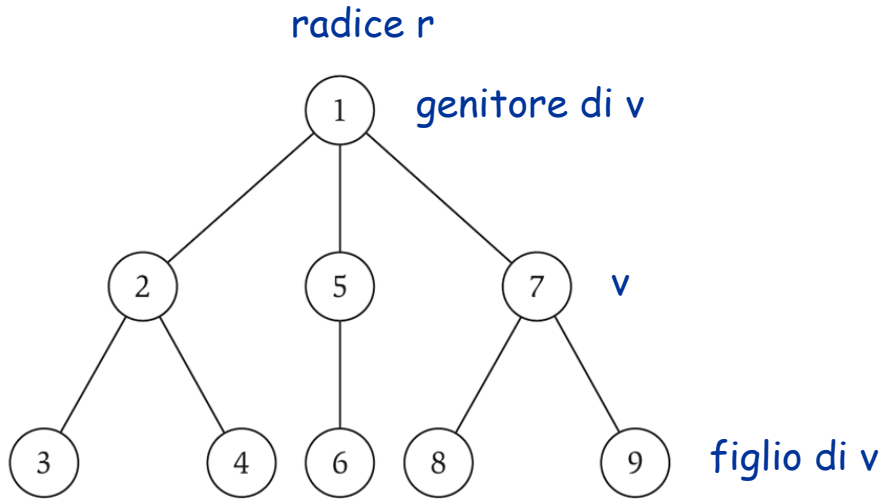
Alberi radicati

Albero radicato. Dato un albero T, scegliere un nodo r come radice e orientare ogni arco rispetto ad r.

Importanza. Modella strutture gerarchiche.



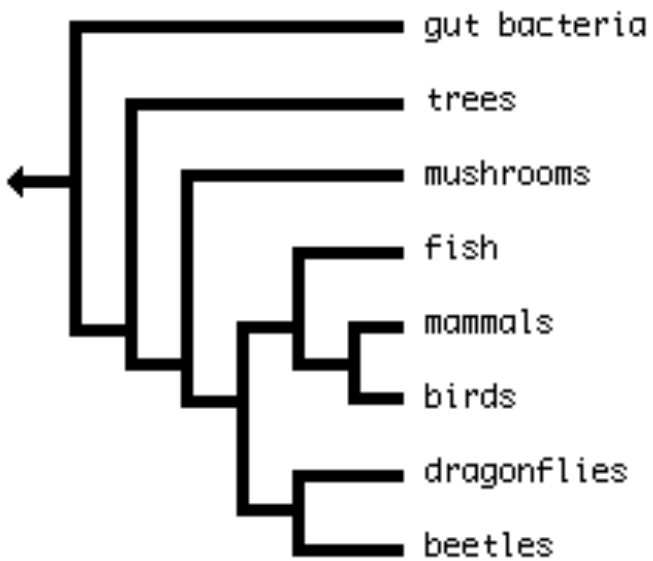
albero



stesso albero radicato in 1

Alberi Filogenetici

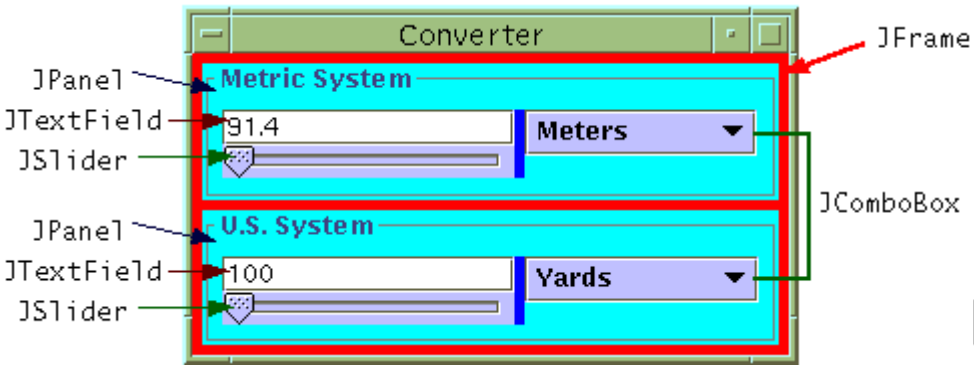
Alberi filogenetici. Descrivono la storia evolutiva delle specie.

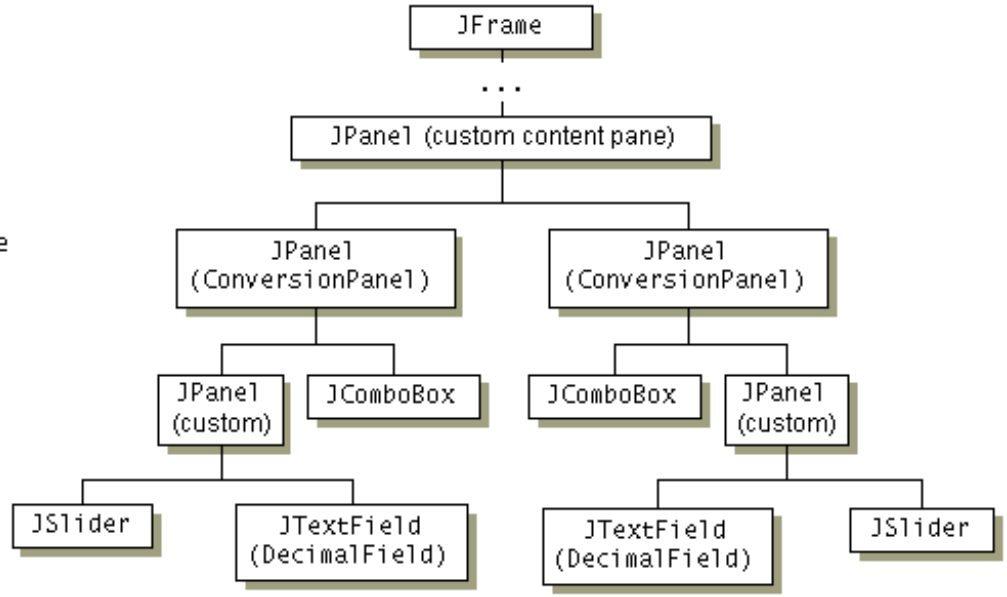


GUI Containment Hierarchy

GUI (Graphical User Interface) containment hierarchy.

Describe organization of GUI widgets.





Reference: http://java.sun.com/docs/books/tutorial/uiswing/overview/anatomy.html

3.2 Graph Traversal

Connettività

Problema della connetività s-t. Dati due nodi s e t, esiste un cammino tra s e t?

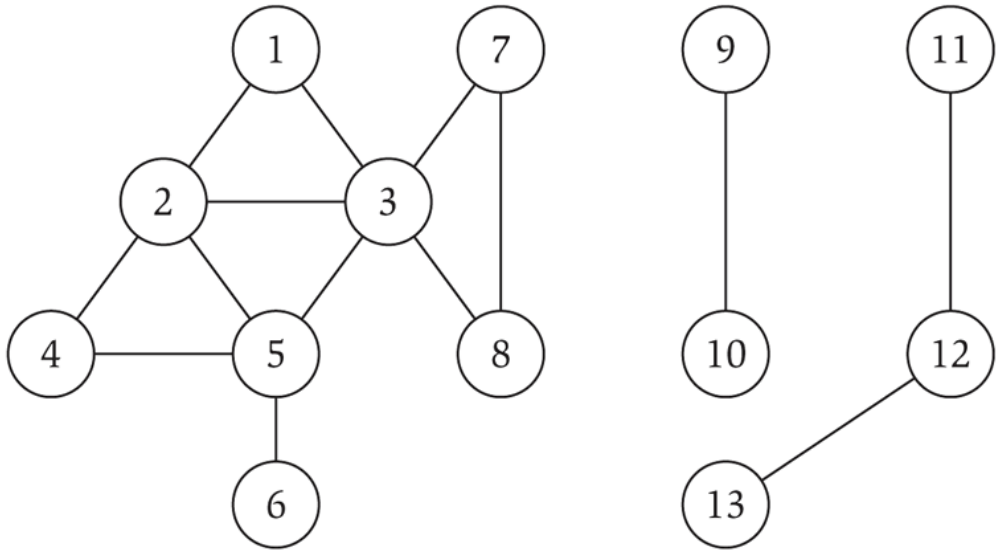


Figure 3.2 In this graph, node 1 has paths to nodes 2 through 8, but not to nodes 9 through 13.

Connettività

Problema della connetività s-t. Dati due nodi s e t, esiste n cammino tra s e t?

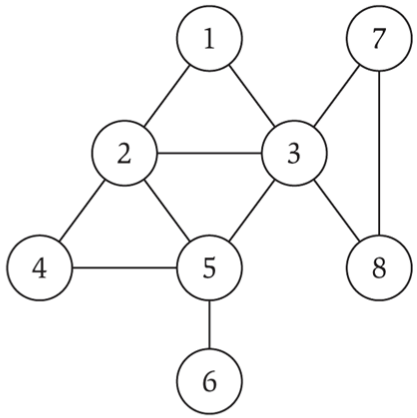
Problema del cammino minimo s-t. Dati due nodi s e t, qual'è la lunghezza del cammino minimo tra s e t?

Applicazioni.

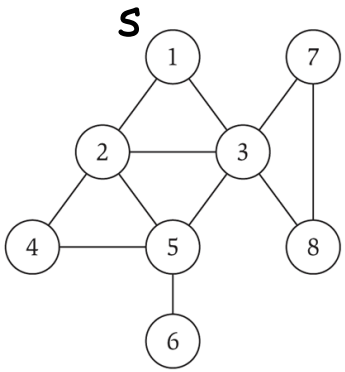
- Numero minimo di hop in una rete di comunicazione.

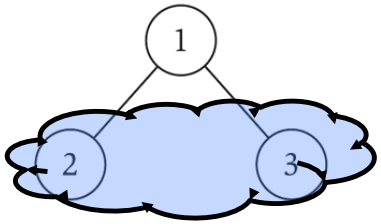
Vedremo:

- Breadth First Search.
- Depth First Search.



Breadth First Search





(a)

Figure 3.3 The construction of a breadth-first search tree T for the graph in Figure 3.2, with (a), (b), and (c) depicting the successive layers that are added. The solid edges are the edges of T; the dotted edges are in the connected component of G containing node 1, but do not belong to T.

Breadth First Search

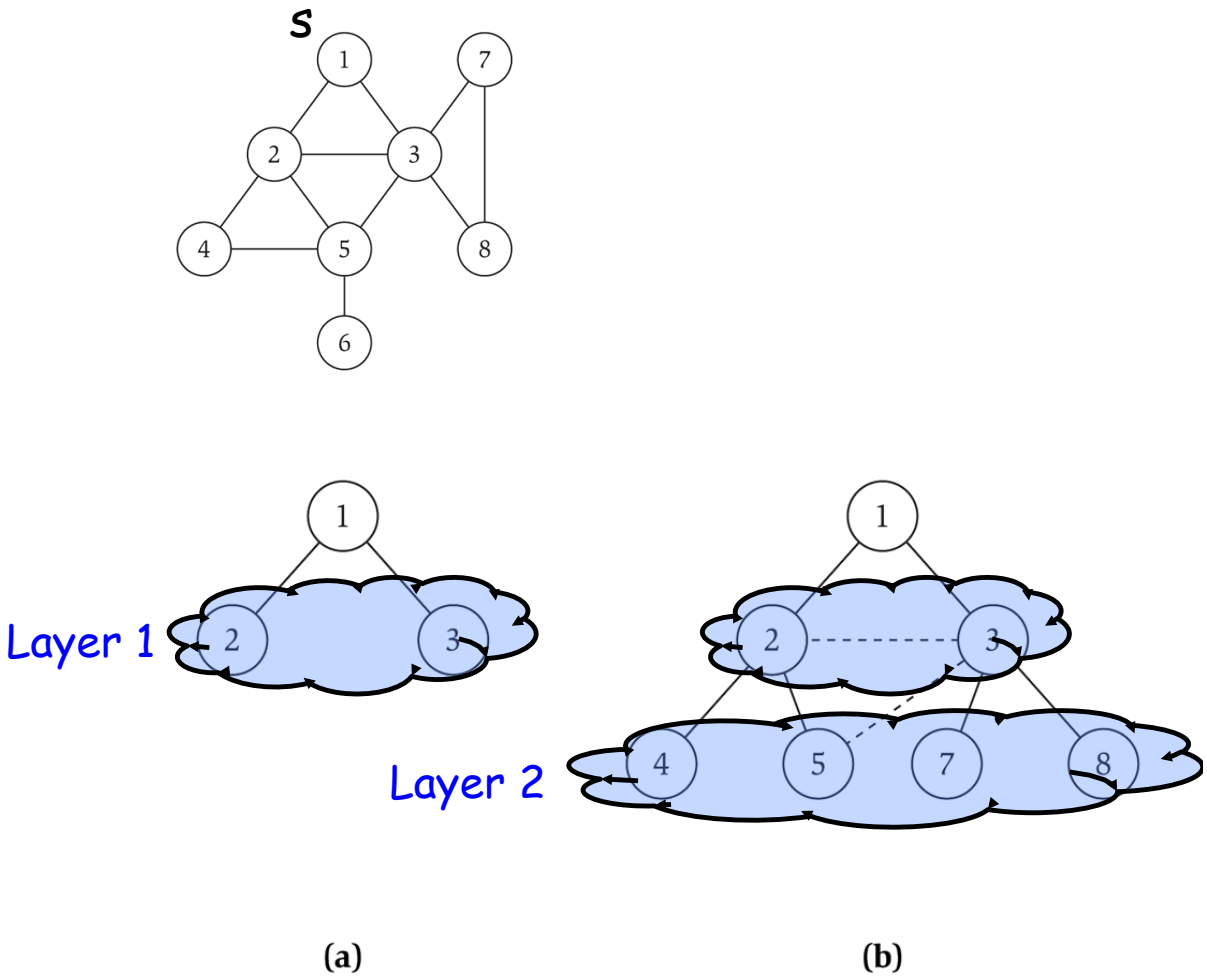
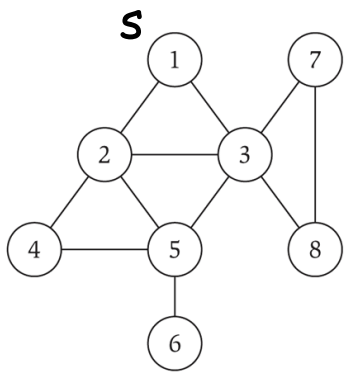


Figure 3.3 The construction of a breadth-first search tree T for the graph in Figure 3.2, with (a), (b), and (c) depicting the successive layers that are added. The solid edges are the edges of T; the dotted edges are in the connected component of G containing node 1, but do not belong to T.

Breadth First Search



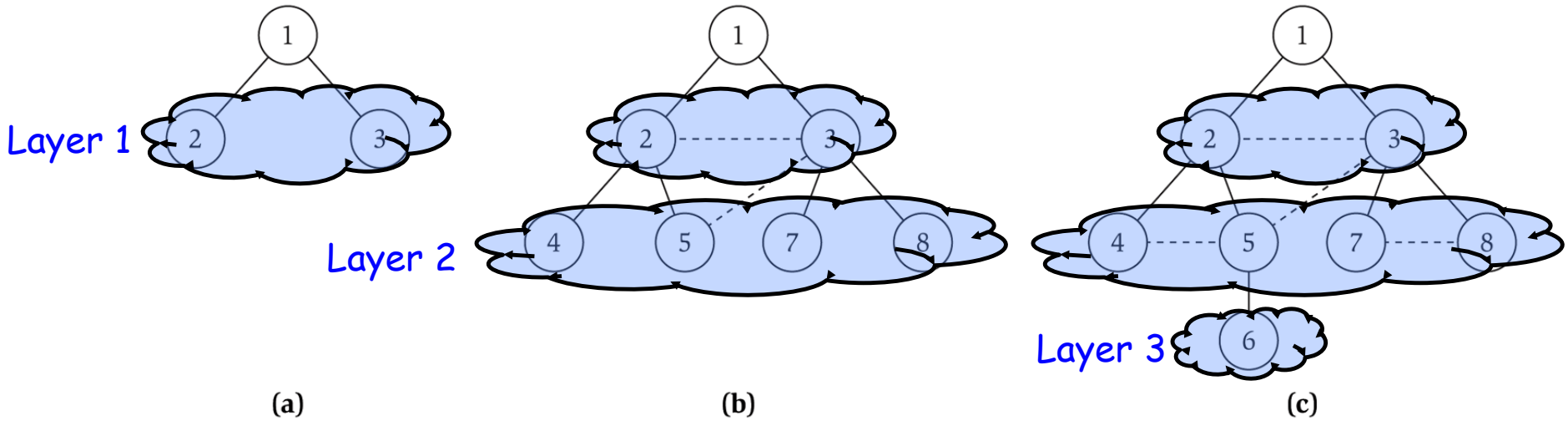
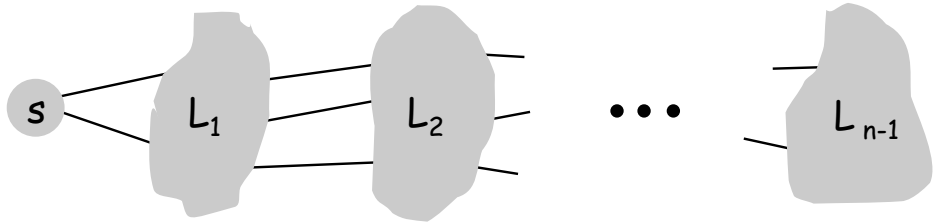


Figure 3.3 The construction of a breadth-first search tree T for the graph in Figure 3.2, with (a), (b), and (c) depicting the successive layers that are added. The solid edges are the edges of T; the dotted edges are in the connected component of G containing node 1, but do not belong to T.

Breadth First Search

BFS intuizione. Esplorare a partire da s in tutte le possibili direzioni, aggiungendo nodi un "layer" alla volta.

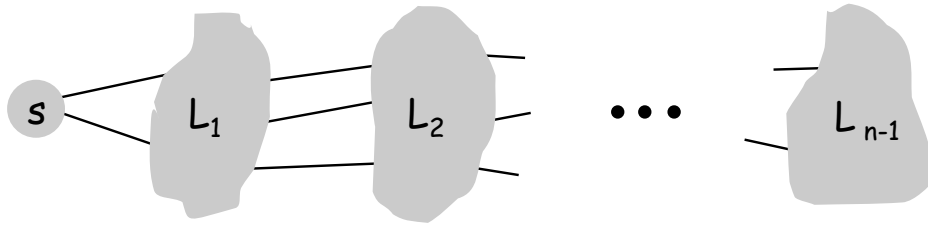


BFS algorithm.

- $L_0 = \{ s \}$.
- L_1 = tutti i vicini di L_0.
- L_2 = tutti i nodi che non sono in L_0 o L_1, e che hanno un arco con un nodo in L_1.
- ...
- L_{i+1} = tutti i nodi che non sono in un layer precedente (cioè $L_0, L_1, \ldots, L_i$), e che hanno un arco con un nodo in L_i.

Breadth First Search

BFS intuizione. Esplorare a partire da s in tutte le possibili direzioni, aggiungendo nodi un "layer" alla volta.



Algoritmo BFS.

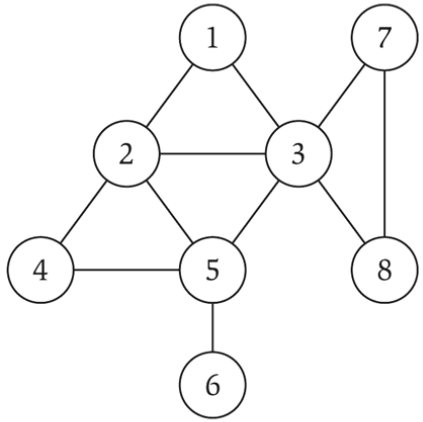
- $L_0 = \{ s \}$.
- L_1 = tutti i vicini di L_0.
- L_2 = tutti i nodi che non sono in L_0 o L_1, e che hanno un arco con un nodo in L_1.
- ...
- L_{i+1} = tutti i nodi che non sono in un layer precedente (cioè $L_0, L_1, \ldots, L_i$), e che hanno un arco con un nodo in L_i.

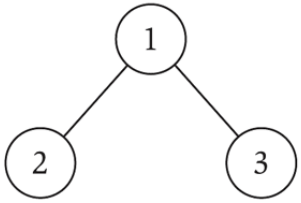
Teorema. Per ogni i, layer L_i consiste di tutti i nodi a distanza i da s. Esiste un cammino da s a t *se e solo se* t è in qualche layer.

Breadth First Search

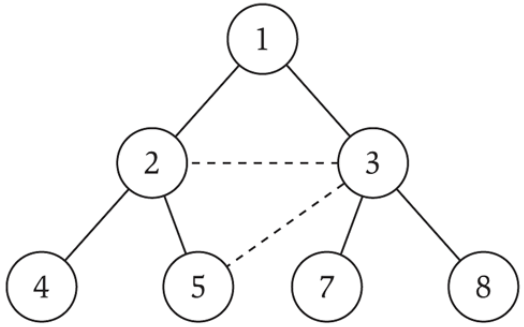
Proprietà. Sia T un albero BFS di $G = (V, E)$, e sia (x, y) un arco di G con x nel layer L_i e y nel layer L_j. Allora $|i-j| \leq 1$.

Prima della prova vediamo un esempio.

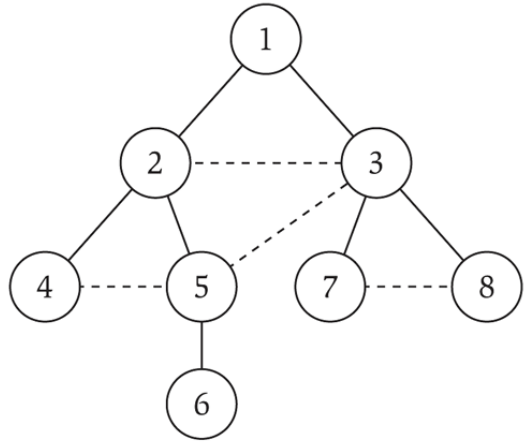




(a)



(b)



(c)

L_0

L_1

L_2

L_3

Breadth First Search

Proprietà. Sia T un albero BFS di $G = (V, E)$, e sia (x, y) un arco di G con x nel layer L_i e y nel layer L_j. Allora $|i-j| \leq 1$.

Prova. Per assurdo.

Supponiamo che $i < j-1$.

Quando l'algoritmo BFS esamina gli archi incidenti ad x nel layer L_i trova (x,y)

- se y è stato esaminato prima allora è in un layer precedente
- se y non è stato esaminato prima allora è nel layer L_{i+1}

Breadth First Search: implementazione

Abbiamo visto la primitiva algoritmica della BFS senza parlare di dettagli implementativi

Prima di vedere lo pseudocodice della BFS vediamo come rappresentare i grafi

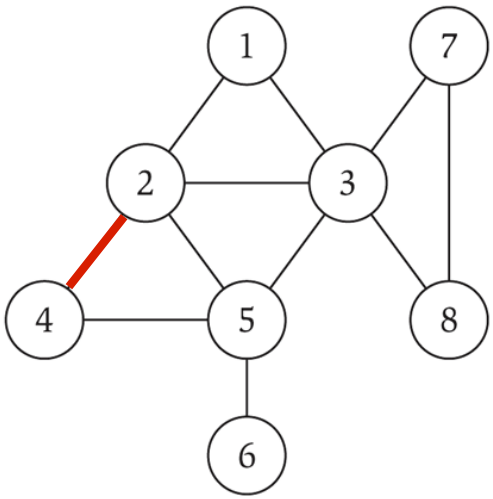
Due rappresentazioni di un grafo:

- Matrice Adiacenze
- Liste adiacenze

Rappresentazione Grafo: Matrice Adiacenze

Matrice adiacenze. Matrice n x n con $A_{uv} = 1$ se (u,v) è un arco.

- Due rappresentazioni di un arco.
- Spazio proporzionale ad n^2.
- Verificare se esiste l'arco (u,v) richiede tempo $\Theta(1)$.
- Listare tutti gli archi richiede tempo $\Theta(n^2)$.
- Listare tutti gli archi incidenti su un fissato nodo richiede tempo $\Theta(n)$.



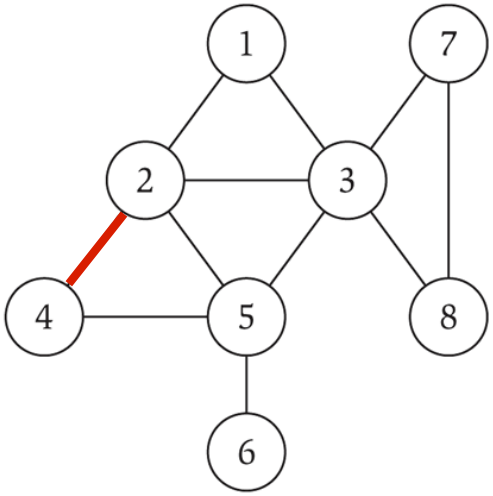
	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	0	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

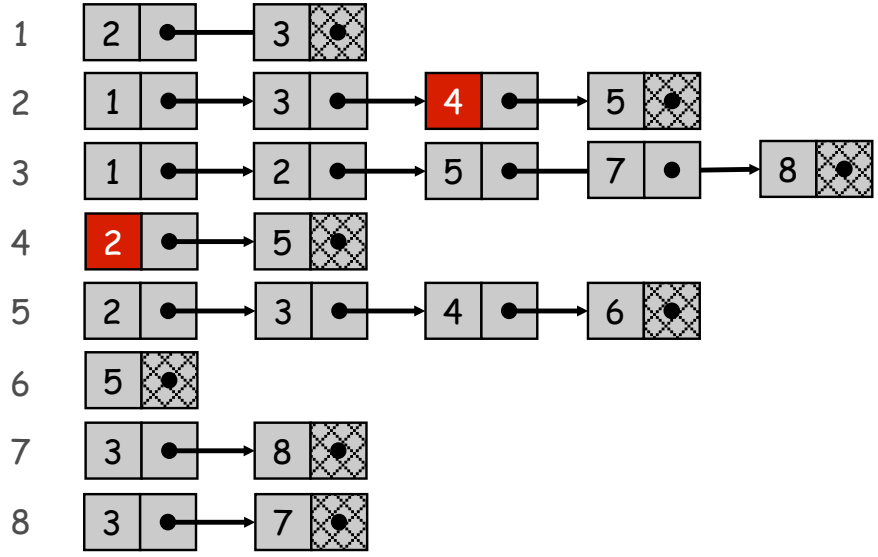
Rappresentazione Grafo: Liste adiacenze

Liste adiacenze.

- Due rappresentazioni di ogni arco.
- Spazio proporzionale a m + n.
- Verificare se esiste l'arco (u,v) richiede tempo $O(\deg(u))$.
- Listare tutti gli archi richiede tempo $\Theta(m + n)$.
- Listare tutti gli archi incidenti su un nodo u richiede tempo $\Theta(\deg(u))$.

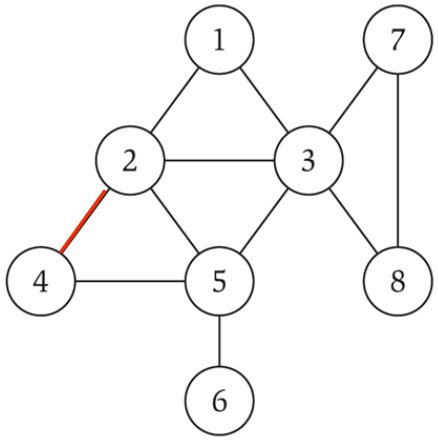
degree = numero di vicini di u





Breadth First Search: implementazione

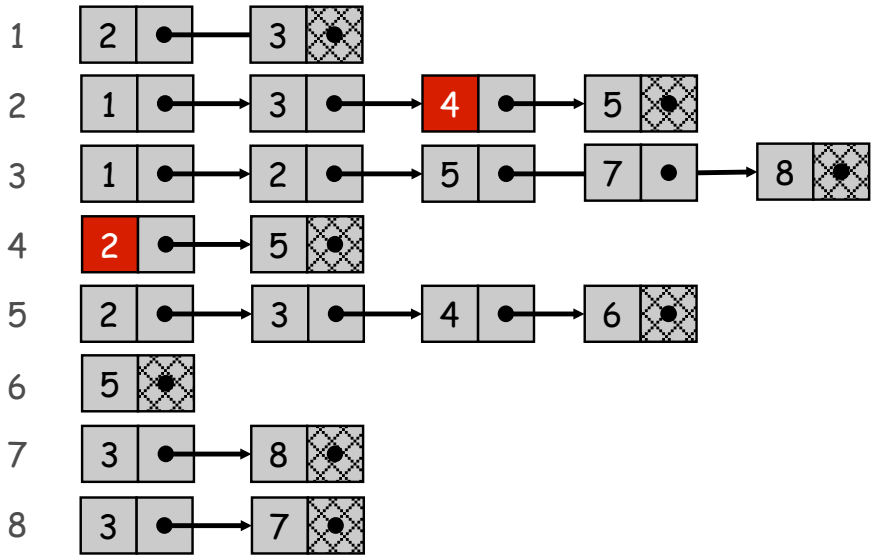
Due rappresentazioni per un grafo



Matrice adiacenze

	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	0	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

Liste adiacenze



Breadth First Search: implementazione

```
BFS(s):
  Set Discovered[s] = true and Discovered[v] = false for all other v
  Initialize L[0] to consist of the single element s
  Set the layer counter i = 0
  Set the current BFS tree T = ∅
  While L[i] is not empty
    Initialize an empty list L[i + 1]
    For each node u ∈ L[i]
      Consider each edge (u, v) incident to u
      If Discovered[v] = false then
        Set Discovered[v] = true
        Add edge (u, v) to the tree T
        Add v to the list L[i + 1]
      Endif
    Endfor
    Increment the layer counter i by one
  Endwhile
```

Breadth First Search: analisi

```
BFS(s):
  Set Discovered[s] = true and Discovered[v] = false for all other v
  Initialize L[0] to consist of the single element s
  Set the layer counter i = 0
  Set the current BFS tree T = ∅
  While L[i] is not empty
    Initialize an empty list L[i+1]
    For each node u ∈ L[i]
      Consider each edge (u, v) incident to u
      If Discovered[v] = false then
        Set Discovered[v] = true
        Add edge (u, v) to the tree T
        Add v to the list L[i+1]
      Endif
    Endfor
    Increment the layer counter i by one
  Endwhile
```

Teorema. L' implementazione della BFS ha un running time O(m + n) se il grafo è rappresentato con una lista delle adiacenze.

Breadth First Search: analisi

```
BFS(s):
  Set Discovered[s] = true and Discovered[v] = false for all other v
  Initialize L[0] to consist of the single element s
  Set the layer counter i = 0
  Set the current BFS tree T = ∅
  While L[i] is not empty
    Initialize an empty list L[i+1]
    For each node u ∈ L[i]
      Consider each edge (u, v) incident to u
      If Discovered[v] = false then
        Set Discovered[v] = true
        Add edge (u, v) to the tree T
        Add v to the list L[i+1]
      Endif
    Endfor
    Increment the layer counter i by one
  Endwhile
```

Al massimo n liste L[i]

E' facile provare un running time $O(n^2)$:

- Ogni nodo è presente in al massimo una lista. Ciclo for al massimo ≤ n volte.
- Per un fissato nodo u, vi sono ≤ n archi incidenti (u, v), e spendiamo $O(1)$ per ogni arco

Breadth First Search: analisi

```
BFS(s):
  Set Discovered[s] = true and Discovered[v] = false for all other v
  Initialize L[0] to consist of the single element s
  Set the layer counter i = 0
  Set the current BFS tree T = ∅
  While L[i] is not empty
    Initialize an empty list L[i+1]
    For each node u ∈ L[i]
      Consider each edge (u, v) incident to u
      If Discovered[v] = false then
        Set Discovered[v] = true
        Add edge (u, v) to the tree T
        Add v to the list L[i+1]
      Endif
    Endfor
    Increment the layer counter i by one
  Endwhile
```

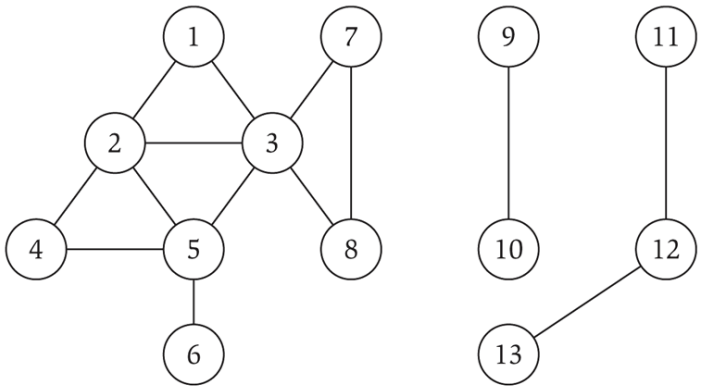
Una migliore limitazione runnning time: O(m + n)

- Per un fissato nodo u vi sono deg(u) archi incidenti (u, v)
- Tempo totale per processare gli archi è $\Sigma_{u \in V} \deg(u) = 2m$

Ogni arco (u, v) è contato esattamente due volte nella somma: una volta in deg(u) e una volta in deg(v)

Componente Connessa

Componente connessa. Trovare tutti i nodi raggiungibili da s.

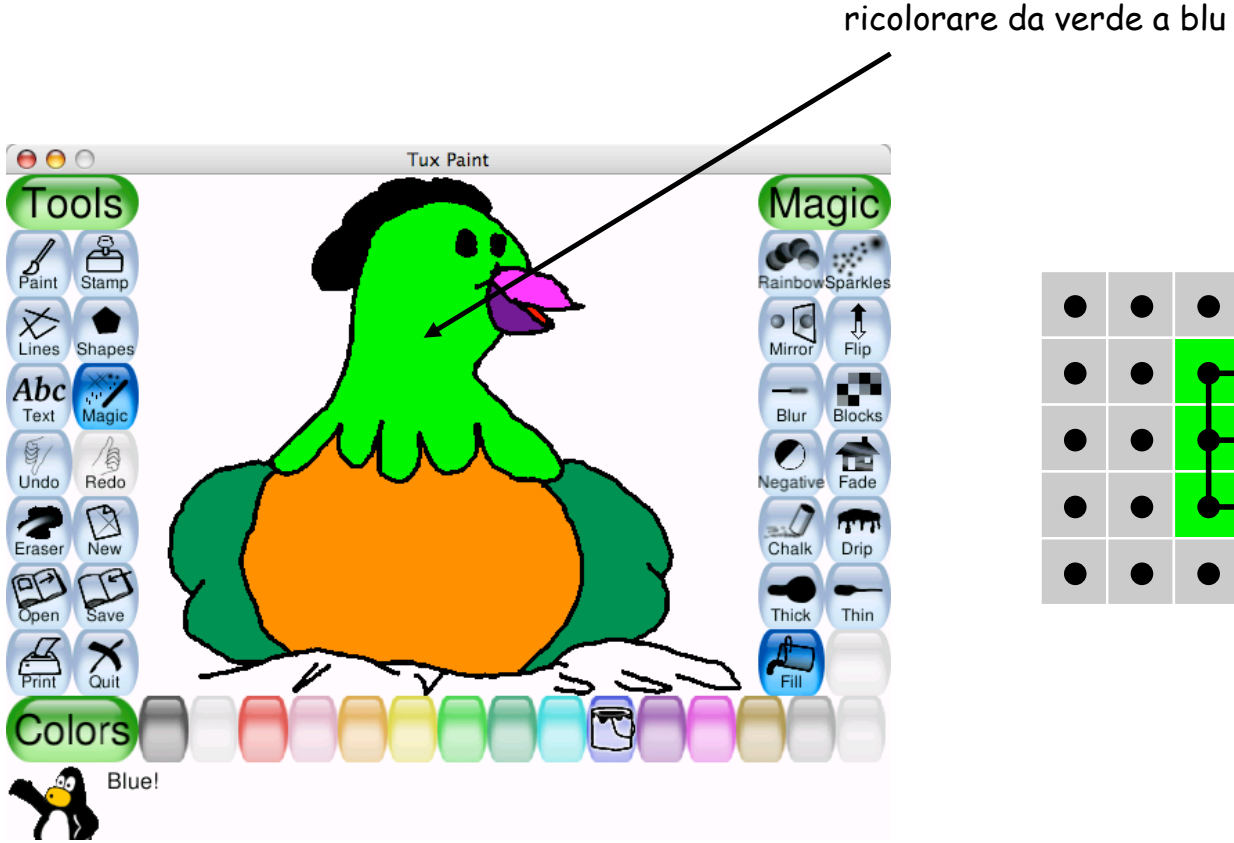


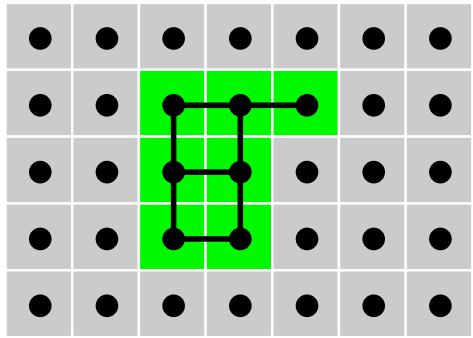
La componente connessa che contiene il nodo 1 è { 1, 2, 3, 4, 5, 6, 7, 8 }.
La componente connessa che contiene il nodo 9 è { 9, 10 }.
La componente connessa che contiene il nodo 11 è { 11, 12, 13 }.

Riempimento aree

Riempimento aree. Per un fissato pixel verde in un'immagine, cambia colore a tutti i pixel adiacenti verdi.

- Nodo: pixel.
- Arco: due pixel adiacenti.
- L'area da cambiare è una componente connessa.

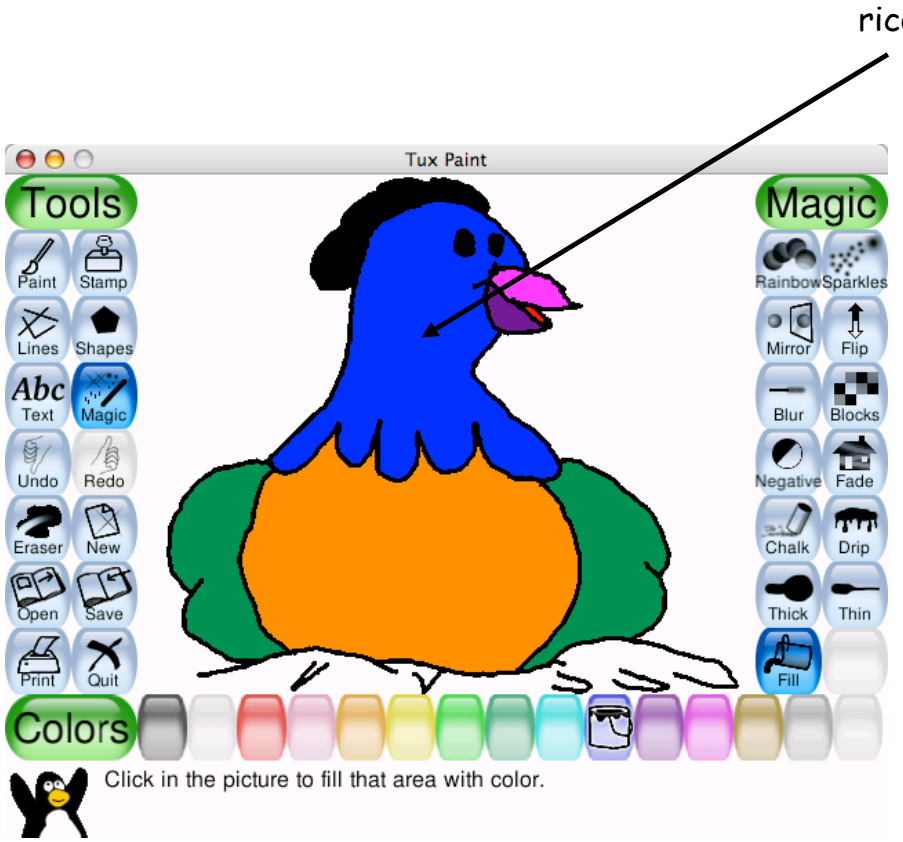


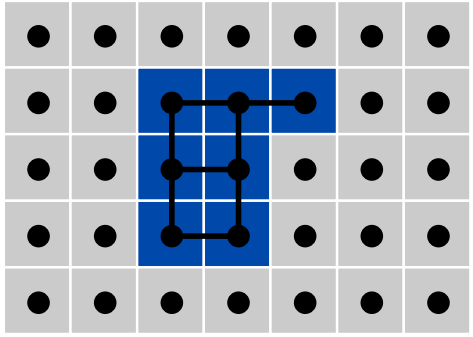


Riempimento aree

Riempimento aree. Per un fissato pixel verde in un'immagine, cambia colore a tutti i pixel adiacenti verdi.

- Nodo: pixel.
- Arco: due pixel adiacenti.
- L'area da cambiare è una componente connessa.





Componente Connessa

Componente connessa di un grafo che contiene un nodo s:
Trovare tutti i nodi raggiungibili da s.

Possiamo usare una Breadth First Search partendo da s.

Ma l'ordine in cui visitiamo i nodi non è importante.

Possiamo computare la componente connessa, visitando i nodi in un arbitrario ordine! Non necessariamente quello della BFS.

Componente Connessa

Componente connessa di un grafo che contiene un nodo s:

Trovare tutti i nodi raggiungibili da s.

R will consist of nodes to which s has a path

Initially $R = \{s\}$

While there is an edge (u, v) where $u \in R$ and $v \notin R$

  Add v to R

Endwhile

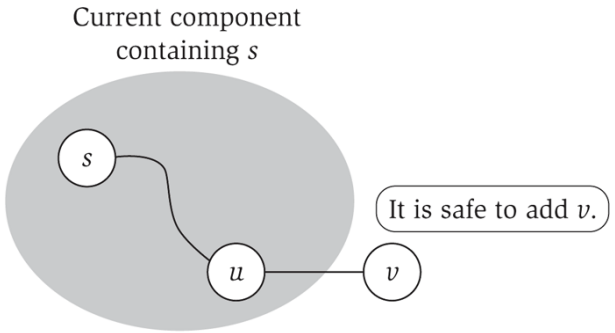


Figure 3.4 When growing the connected component containing s, we look for nodes like v that have not yet been visited.

Componente Connessa

Componente connessa di un grafo che contiene un nodo s:
Trovare tutti i nodi raggiungibili da s.

R will consist of nodes to which s has a path
Initially $R = \{s\}$
While there is an edge (u, v) where $u \in R$ and $v \notin R$
  Add v to R
Endwhile

Proprietà. Terminato l'algoritmo, l'insieme R è la componente connessa che contiene s.

Prova. Per assurdo.

Supponiamo che esista w non in R ed un cammino s-w nel grafo.

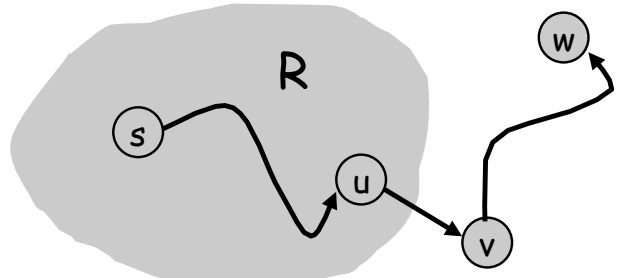
Allora sia v il primo nodo non in R in tale cammino.

v non può essere s, perchè s è in R

Sia u il nodo che precede v nel cammino.

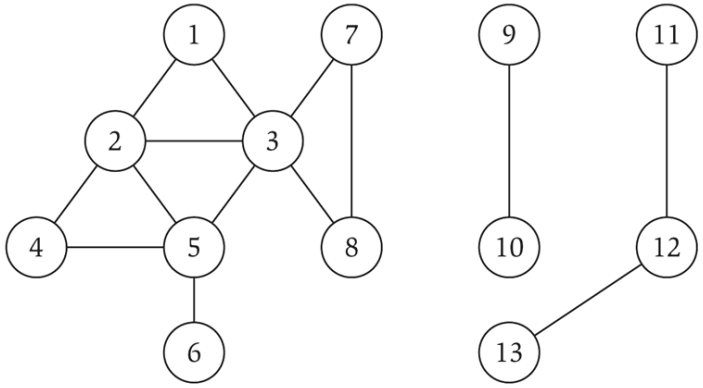
Allora esiste (u,v).

Contraddizione con la regola di termine del while.



Insieme di tutte le componenti connesse

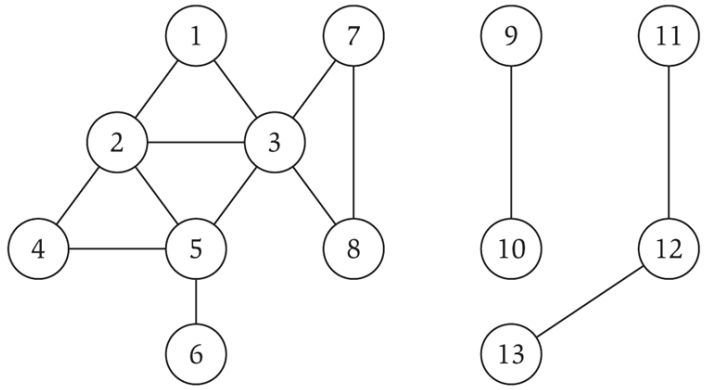
Che relazione c'è tra due componenti connesse?
Possono avere una intersezione?



La componente connessa che contiene il nodo 1 è { 1, 2, 3, 4, 5, 6, 7, 8 }.
La componente connessa che contiene il nodo 9 è { 9, 10 }.
La componente connessa che contiene il nodo 11 è { 11, 12, 13 }.

Insieme di tutte le componenti connesse

Proprietà. Siano x e y due nodi in un grafo G. Allora le loro componenti connesse sono identiche oppure disgiunte.



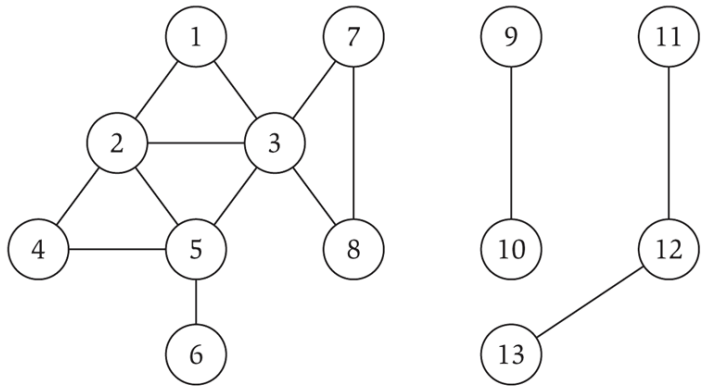
La componente connessa che contiene il nodo 1 è { 1, 2, 3, 4, 5, 6, 7, 8 }.
La componente connessa che contiene il nodo 9 è { 9, 10 }.
La componente connessa che contiene il nodo 11 è { 11, 12, 13 }.

Insieme di tutte le componenti connesse

Proprietà. Siano x e y due nodi in un grafo G. Allora le loro componenti connesse sono identiche oppure disgiunte.

Prova. Distinguiamo due casi:

- Esiste un cammino tra x e y.
- Non esiste un cammino tra x e y.



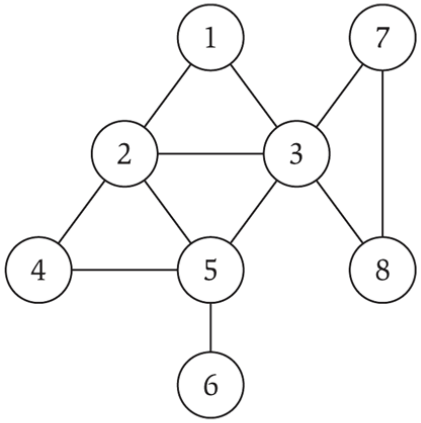
La componente connessa che contiene il nodo 1 è { 1, 2, 3, 4, 5, 6, 7, 8 }.
La componente connessa che contiene il nodo 9 è { 9, 10 }.
La componente connessa che contiene il nodo 11 è { 11, 12, 13 }.

Depth-First Search

DFS intuizione. Esplorare quanto più in profondità possibile e torna indietro (*backtrack*) solo quando è necessario.

```
DFS(u):
  Mark u as "Explored" and add u to R
  For each edge (u, v) incident to u
    If v is not marked "Explored" then
      Recursively invoke DFS(v)
    Endif
  Endfor
```

Depth-First Search



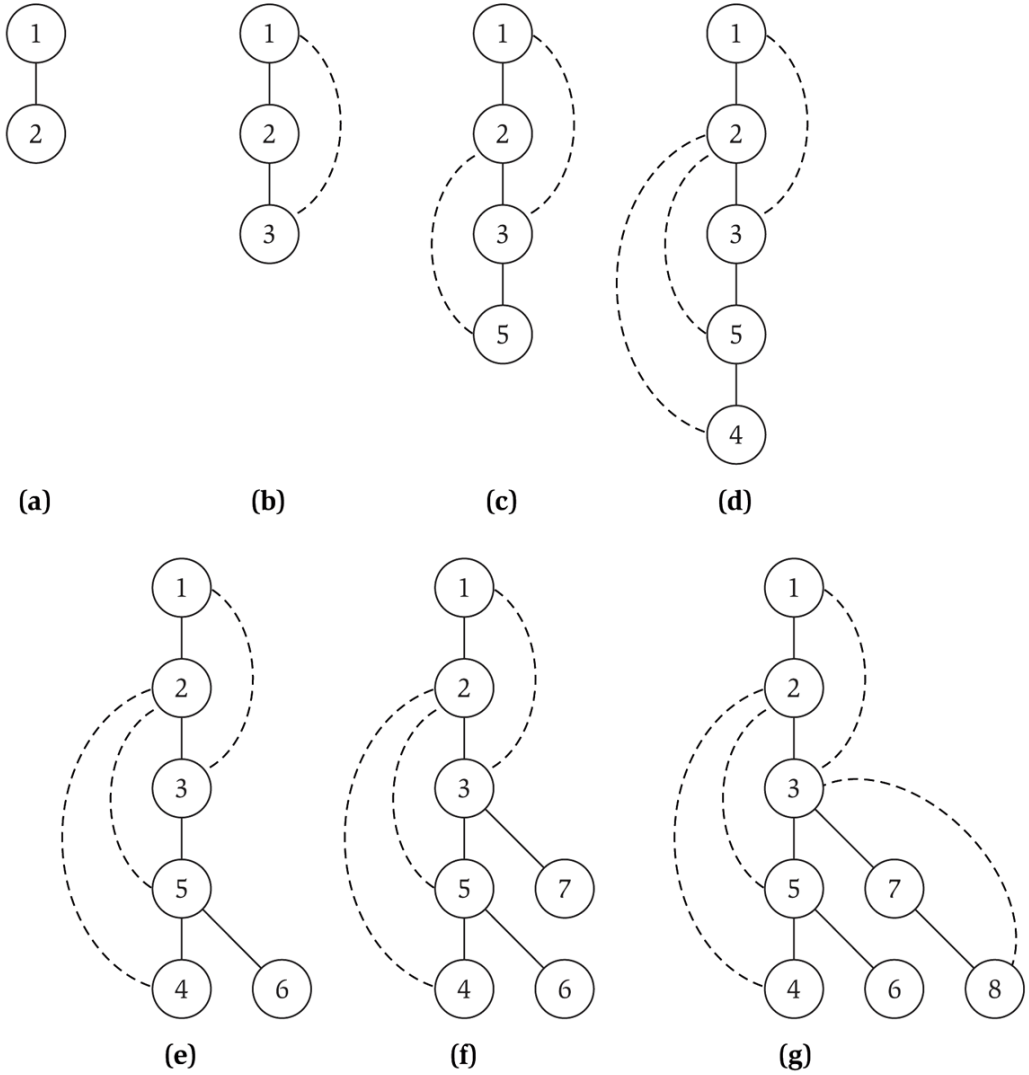


Figure 3.5 The construction of a depth-first search tree T for the graph in Figure 3.2, with (a) through (g) depicting the nodes as they are discovered in sequence. The solid edges are the edges of T; the dotted edges are edges of G that do not belong to T.

Depth-First Search

Proprietà. Per una fissata chiamata ricorsiva DFS(u), tutti i nodi che sono marcati "Explored" tra l'invocazione e la fine della chiamata ricorsiva sono discendenti di u in T.

```
DFS(u):
  Mark u as "Explored" and add u to R
  For each edge (u, v) incident to u
    If v is not marked "Explored" then
      Recursively invoke DFS(v)
    Endif
  Endfor
```

Depth-First Search

Proprietà. Sia T un albero di ricerca depth-first, siano x e y nodi in T, e sia (x,y) un arco di G che non è arco di T. Allora x è antenato di y oppure y è antenato di x.

Prova. Per assurdo.

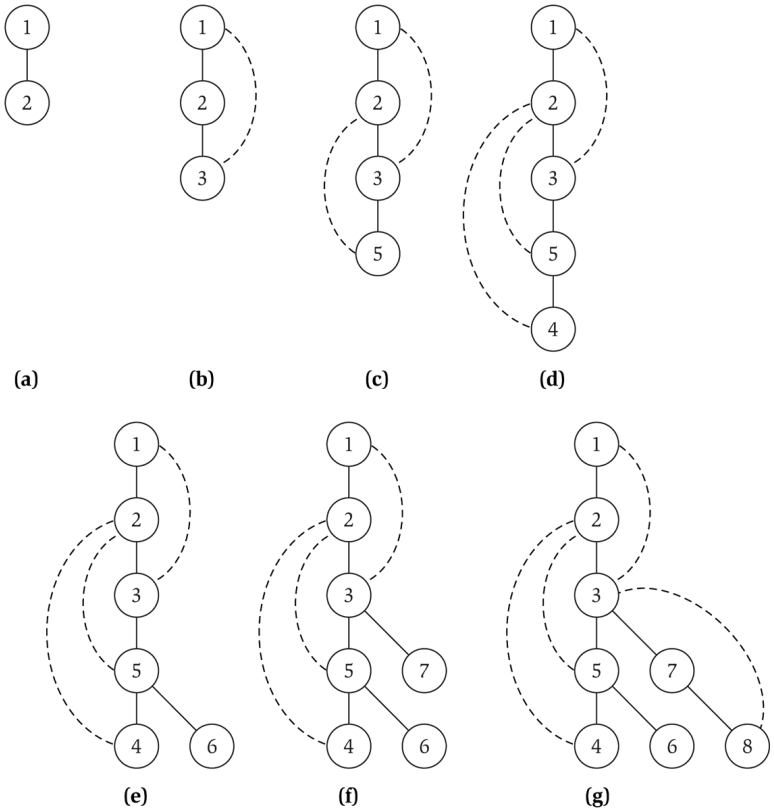
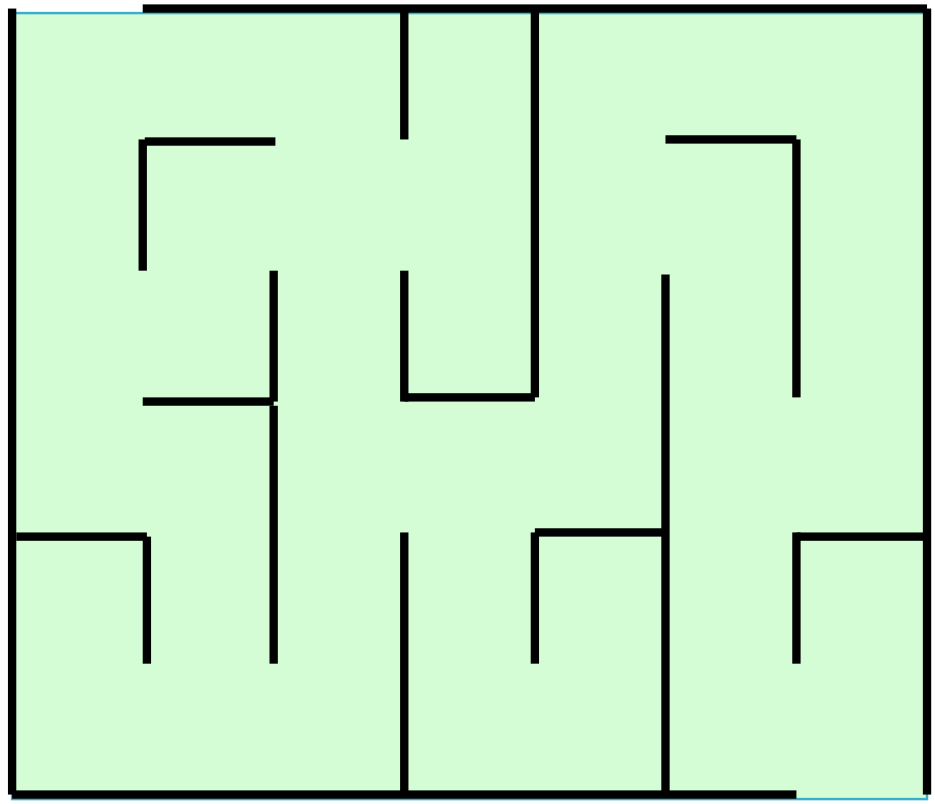


Figure 3.5 The construction of a depth-first search tree T for the graph in Figure 3.2, with (a) through (g) depicting the nodes as they are discovered in sequence. The solid edges are the edges of T; the dotted edges are edges of G that do not belong to T.

DFS e labirinti

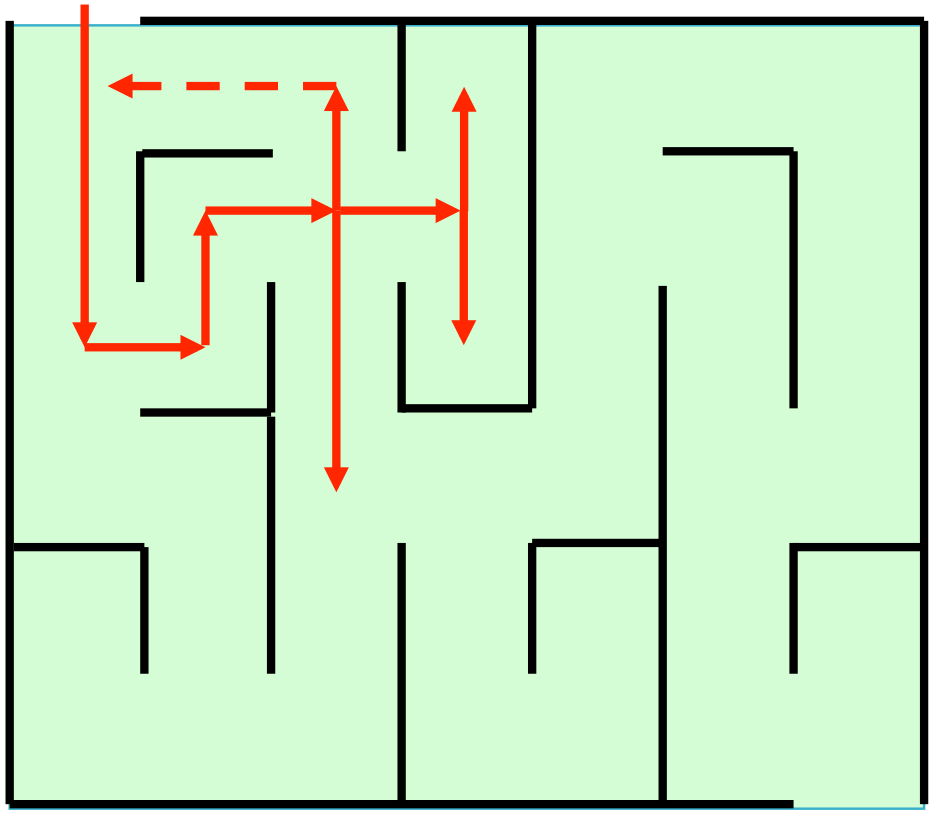
DFS è simile alla classica strategia per esplorare un labirinto



DFS e labirinti

DFS è simile alla classica strategia per esplorare un labirinto

- Marchiamo ogni intersezione, angolo e vicolo cieco (nodo) visitato
- Marchiamo ogni corridoio (arco) visitato
- Teniamo traccia del cammino fatto dall'entrata (nodo partenza) usando una corda (stack della ricorsione)



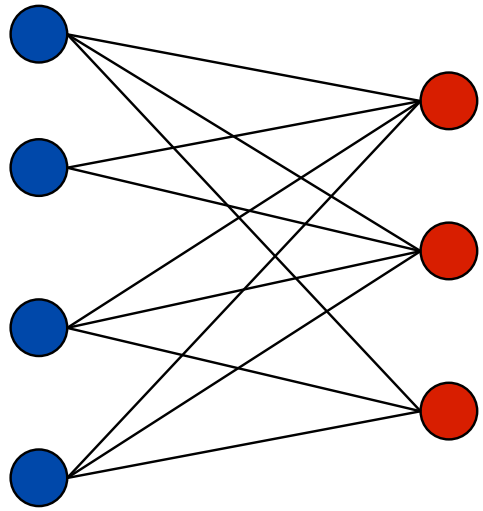
3.4 Testing Bipartiteness

Grafi bipartiti

Definizione. Un grafo nondiretto $G = (V, E)$ è *bipartito* se i nodi possono essere colorati rosso o blu in modo che ogni arco è incidente su un nodo rosso ed un nodo blu.

Applicazioni.

- Matrimonio stabile: uomini = blu, donne = rosso.
- Scheduling: macchine = blu, job = rosso.

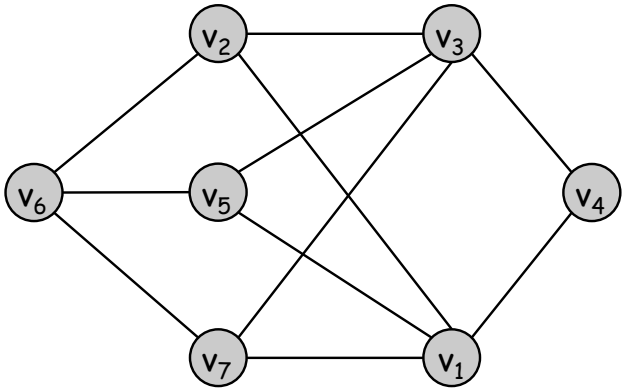


un grafo bipartito

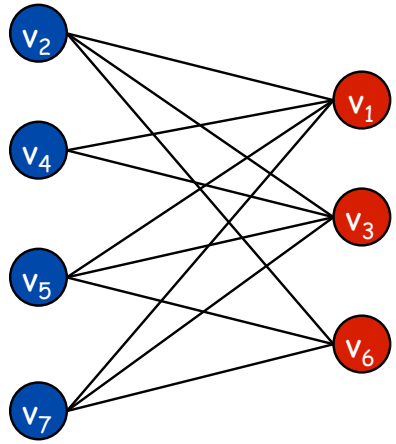
Verificare se un grafo è bipartito

Dato un grafo G, è bipartito?

- Molti problemi su grafi diventano:
  - più semplici se il grafo è bipartito (per esempio, matching)
  - trattabili se il grafo è bipartito (per esempio, independent set)
- Capiamo meglio la struttura dei grafi bipartiti.



un grafo bipartito G

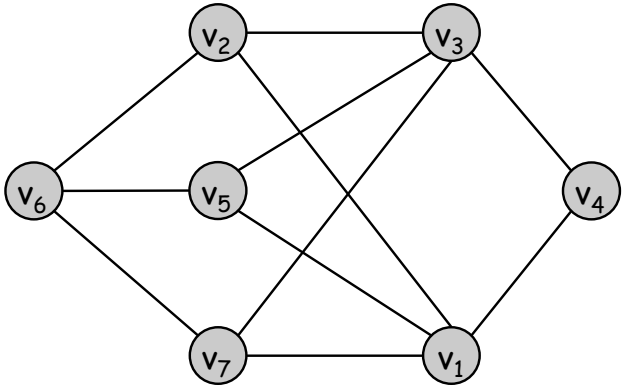


lo stesso grafo G

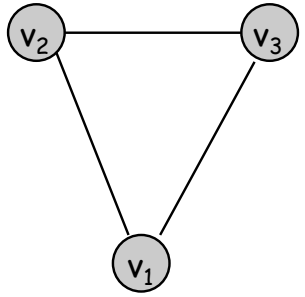
Verificare se un grafo è bipartito

Dato un grafo G, è bipartito?

- Molti problemi su grafi diventano:
  - più semplici se il grafo è bipartito (per esempio, matching)
  - trattabili se il grafo è bipartito (per esempio, independent set)
- Capiamo meglio la struttura dei grafi bipartiti.



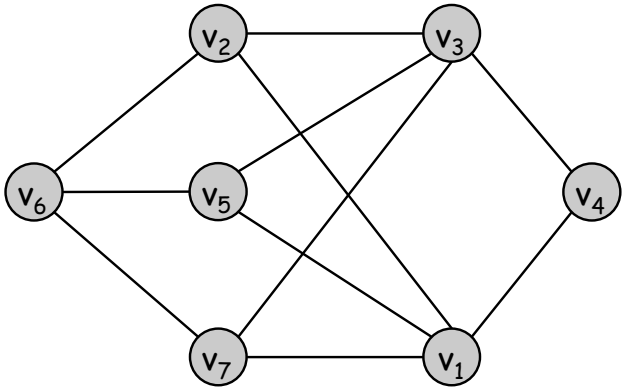
un grafo bipartito G



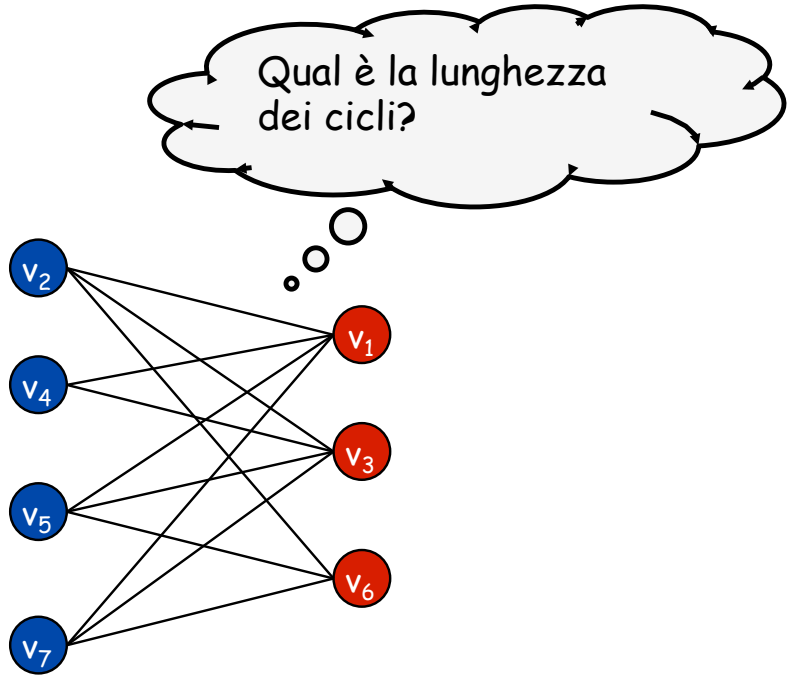
un grafo non bipartito

Cicli in grafi bipartiti

In un grafo bipartito possono esserci cicli



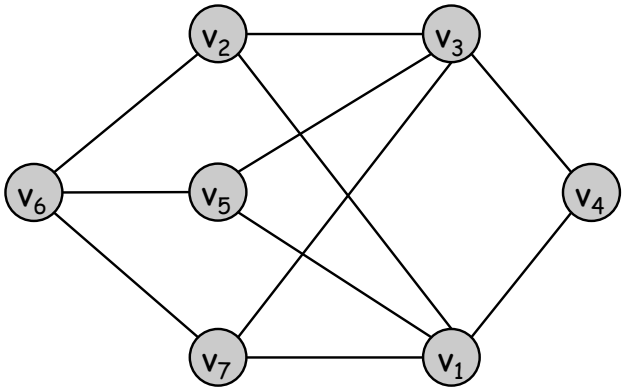
un grafo bipartito *G*



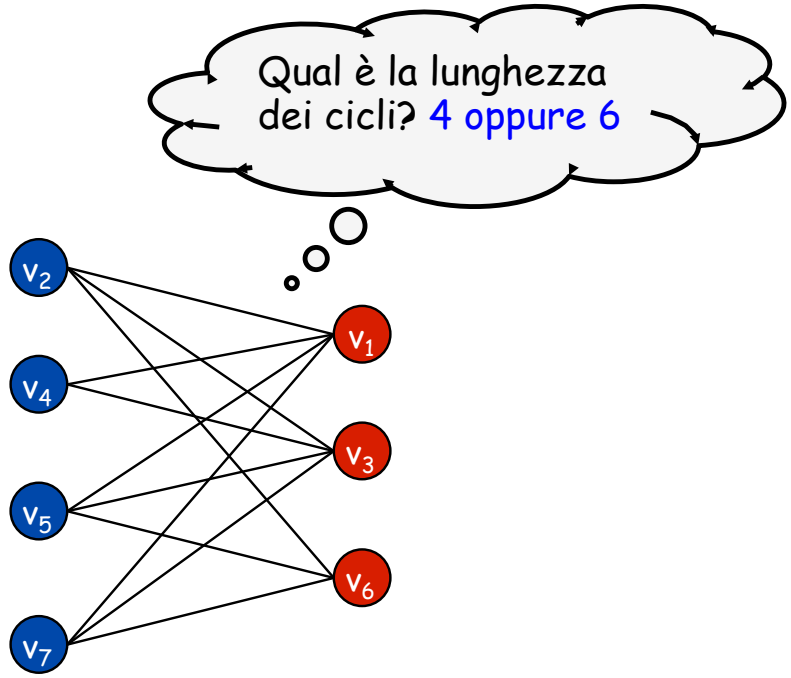
lo stesso grafo *G*

Cicli in grafi bipartiti

In un grafo bipartito possono esserci cicli



un grafo bipartito G

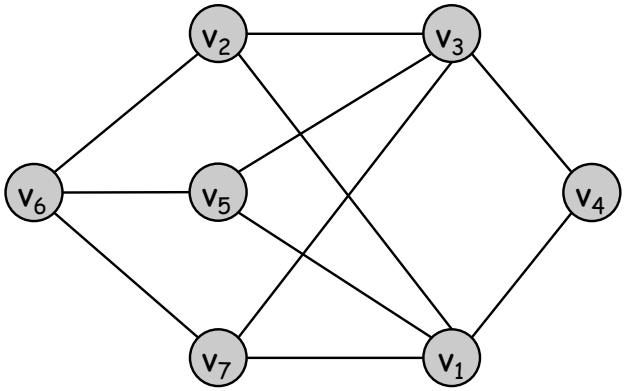


lo stesso grafo G

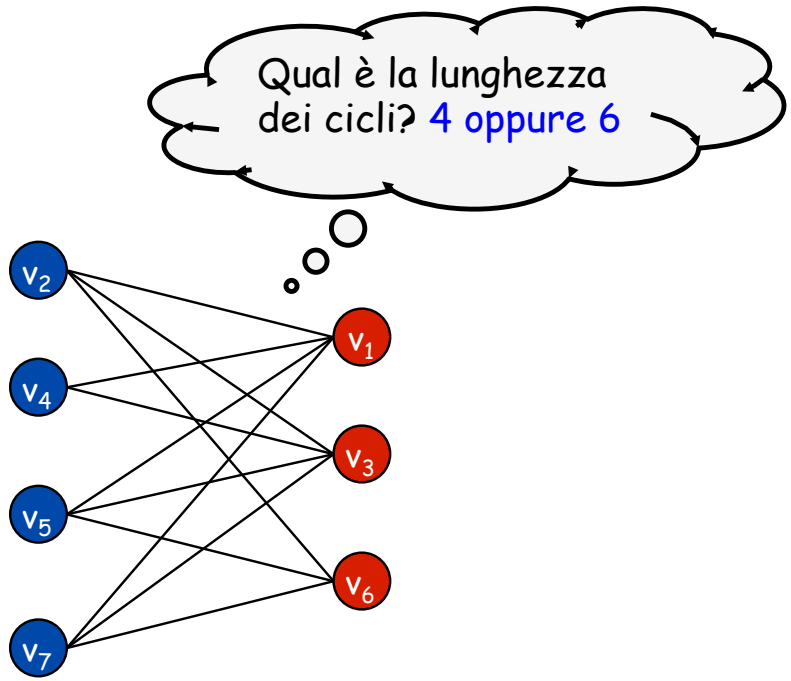
Cicli in grafi bipartiti

In un grafo bipartito possono esserci cicli

- Lunghezza pari ?
- Lunghezza dispari ?



un grafo bipartito *G*

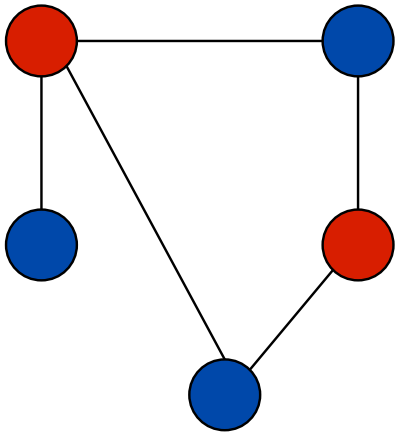


lo stesso grafo *G*

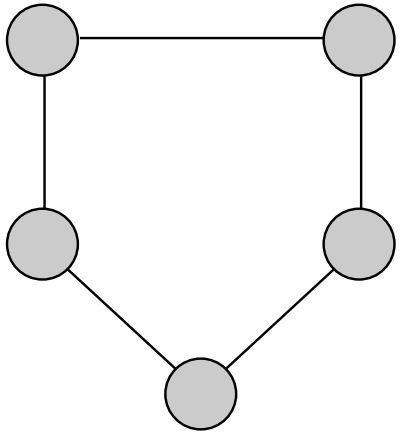
Cicli in grafi bipartiti

Lemma. Se un grafo G è bipartito, non può contenere un ciclo di lunghezza dispari.

Prova. Non è possibile colorare con 2 colori un ciclo dispari.



bipartito
(2-colorabile)



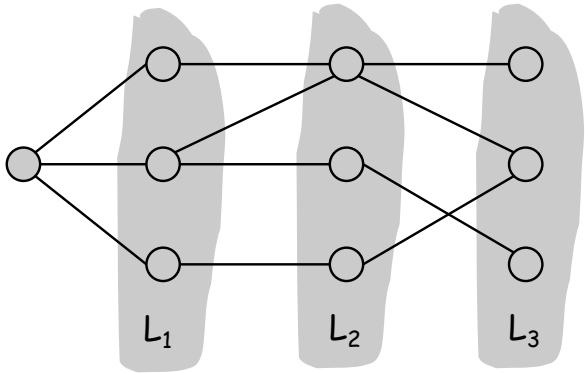
non bipartito
(non 2-colorabile)

Grafi bipartiti

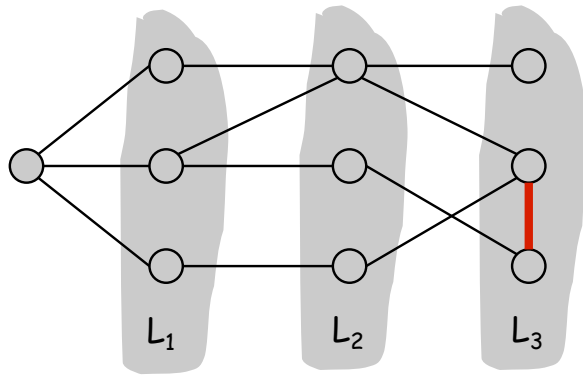
Lemma. Sia G un grafo connesso, e siano L_0, …, L_k layer prodotti da una BFS partendo dal nodo s. Esattamente una delle seguenti è vera:

(i) Nessun arco di G collega due nodi dello stesso layer, e quindi G è bipartito

(ii) Un arco di G collega due nodi dello stesso layer, e quindi G contiene un ciclo di lunghezza dispari (e quindi non è bipartito).



Caso (i)



Caso (ii)

Grafi bipartiti

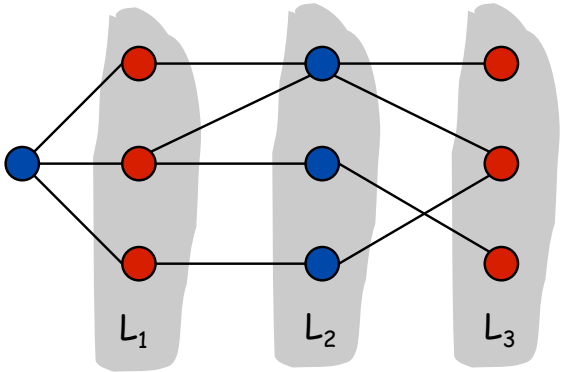
Lemma. Sia G un grafo connesso, e siano L_0, …, L_k layer prodotti da una BFS partendo dal nodo s. Esattamente una delle seguenti è vera:

(i) Nessun arco di G collega due nodi dello stesso layer, e quindi G è bipartito

(ii) Un arco di G collega due nodi dello stesso layer, e quindi G contiene un ciclo di lunghezza dispari (e quindi non è bipartito).

Prova. (i)

- Supponiamo che nessun arco collega due nodi dello stesso layer
- Allora ogni arco collega nodi tra layer successivi
- Colorazione grafo: rosso = nodi livelli dispari, blu = nodi livelli pari



Caso (i)

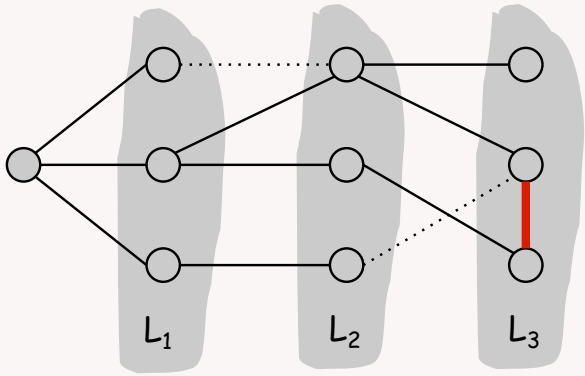
Grafi bipartiti

Lemma. Sia G un grafo connesso, e siano $L_0, \ldots, L_k$ layer prodotti da una BFS partendo dal nodo s. Esattamente una delle seguenti è vera:

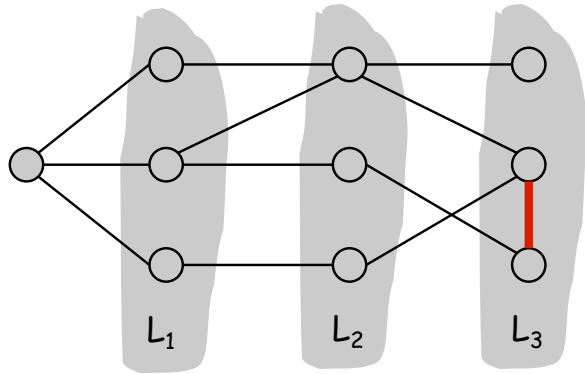
(i) Nessun arco di G collega due nodi dello stesso layer, e quindi G è bipartito

(ii) Un arco di G collega due nodi dello stesso layer, e quindi G contiene un ciclo di lunghezza dispari (e quindi non è bipartito).

Facciamo vedere che vi è un ciclo di lunghezza dispari usando l'albero BFS



Caso (ii)



Caso (ii)

Grafi bipartiti

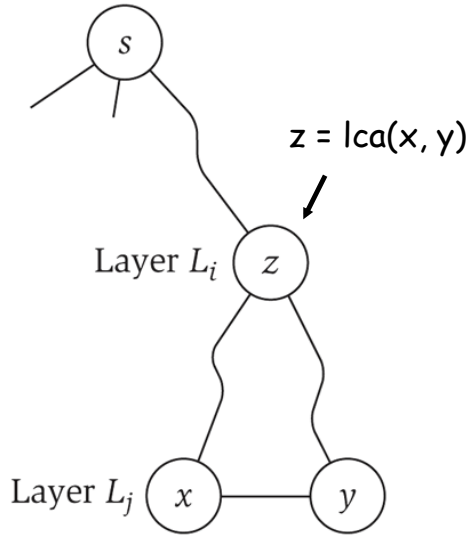
Lemma. Sia G un grafo connesso, e siano L_0, …, L_k layer prodotti da una BFS partendo dal nodo s. Esattamente una delle seguenti è vera:

(i) Nessun arco di G collega due nodi dello stesso layer, e quindi G è bipartito

(ii) Un arco di G collega due nodi dello stesso layer, e quindi G contiene un ciclo di lunghezza dispari (e quindi non è bipartito).

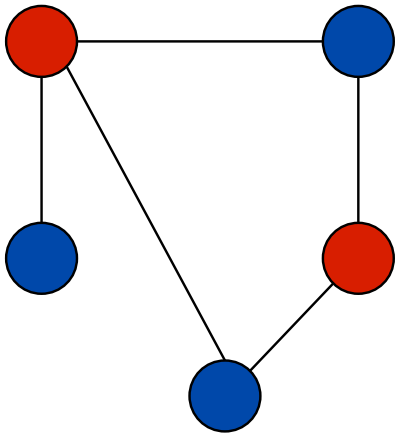
Prova. (ii)

- Supponiamo {x, y} è arco con x, y nello stesso L_j.
- Sia z = lca(x, y) = lowest common ancestor.
- Sia L_i il layer che contiene z.
- Considera il ciclo con archi nel cammino x-y, poi cammino y-z, poi cammino z-x.
- Lunghezza $\underbrace{1}_{\{x,\,y\}} + \underbrace{(j-i)}_{\text{cammino da y to z}} + \underbrace{(j-i)}_{\text{cammino da z to x}}$, che è dispari.

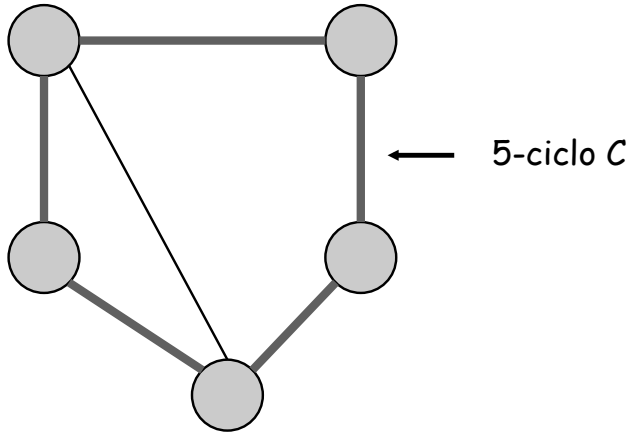


Grafi bipartiti

Corollario. Un grafo G è bipartito *se e solo se* non contiene cicli di lunghezza dispari.



bipartito
(2-colorabile)



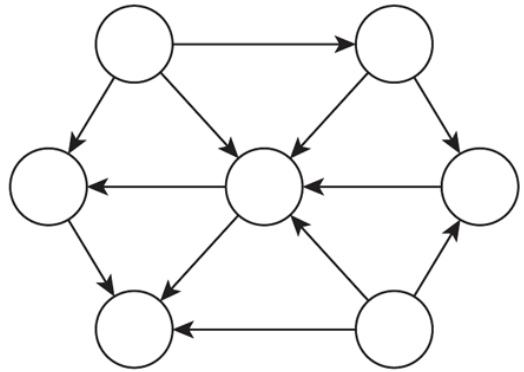
non bipartito
(non 2-colorabile)

3.5 Connectivity in Directed Graphs

Grafi diretti

Grafi diretti. $G = (V, E)$

- Arco (u, v) da nodo u a nodo v.



Esempio. Grafo del Web - hyperlink da una pagina ad un'altra.

- Motori di ricerca utilizzano la struttura degli hyperlink per fare un ranking delle pagine web per importanza.

Ricerca in un grafo

Raggiungibilità diretta. Dato un nodo s, trovare tuti i nodi raggiungibili da s.

Web crawler. Partire dalla pagina web s. Trovare tutte le pagine web linkate da s, direttamente oppure indirettamente.

Problema del cammino diretto s-t minimo. Dati due nodi s e t, qual'è la lnghezza del cammino minimo da s a t?

Ricerca su un grafo. DFS e BFS si estendono naturalmente a grafi diretti.

Connettività forte

Definizione. Nodo u e nodo v sono mutualmente raggiungibili se e solo se esistono un cammino da u a v ed anche un cammino da v ad u.

Definizione. Un grafo è fortemente connesso se ogni coppia di nodi è mutualmente raggiungibile.

Lemma. Sia s un nodo. G è fortemente connesso se e solo se ogni nodo è raggiungibile da s, ed s è raggiungibile da ogni nodo.

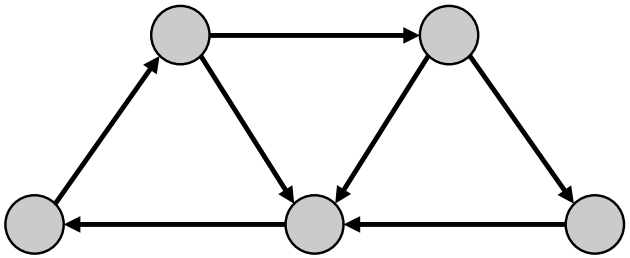
Prova $\Rightarrow$

Prova $\Leftarrow$

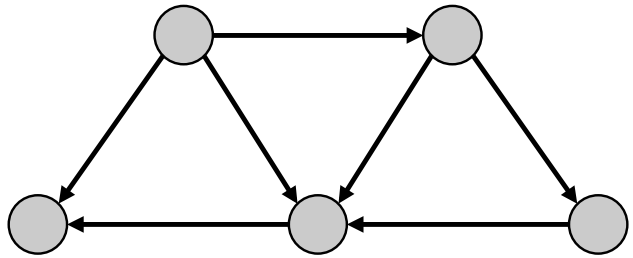
Connettività forte

Definizione. Nodo u e nodo v sono mutualmente raggiungibili se e solo se esistono un cammino da u a v ed anche un cammino da v ad u.

Definizione. Un grafo è fortemente connesso se ogni coppia di nodi è mutualmente raggiungibile.



fortemente connesso



non fortemente connesso

Connettività forte

Definizione. Nodo u e nodo v sono mutualmente raggiungibili se e solo se esistono un cammino da u a v ed anche un cammino da v ad u.

Definizione. Un grafo è fortemente connesso se ogni coppia di nodi è mutualmente raggiungibile.

Lemma. Sia s un nodo. G è fortemente connesso se e solo se ogni nodo è raggiungibile da s, ed s è raggiungibile da ogni nodo.

Prova $\Rightarrow$ Immediato dalla definizione.

Prova $\Leftarrow$

Connettività forte

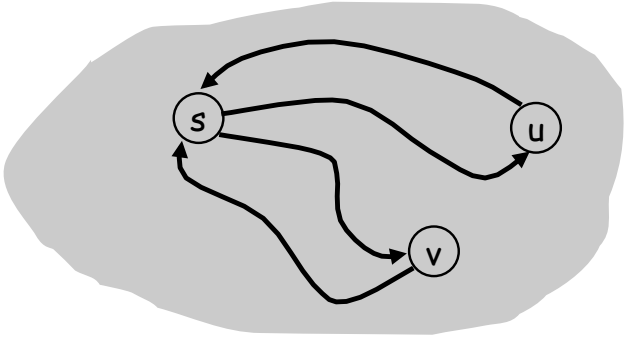
Definizione. Nodo u e nodo v sono mutualmente raggiungibili se e solo se esistono un cammino da u a v ed anche un cammino da v ad u.

Definizione. Un grafo è fortemente connesso se ogni coppia di nodi è mutualmente raggiungibile.

Lemma. Sia s un nodo. G è fortemente connesso se e solo se ogni nodo è raggiungibile da s, ed s è raggiungibile da ogni nodo.

Prova $\Rightarrow$ Immediato dalla definizione.

Prova $\Leftarrow$ Cammino da u a v: concatenare cammino u-s con cammino s-v
Cammino da v ad u: concatenare cammino v-s con cammino s-u



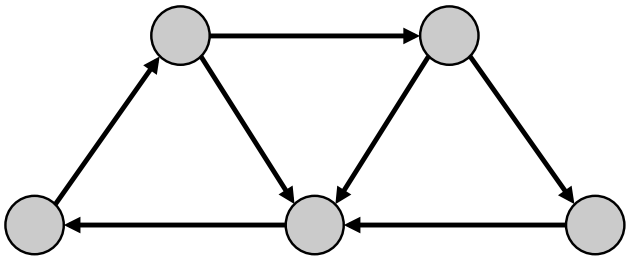
Connettività forte: Algoritmo

Teorema. Possiamo determinare se G è fortemente connesso in tempo O(m + n).

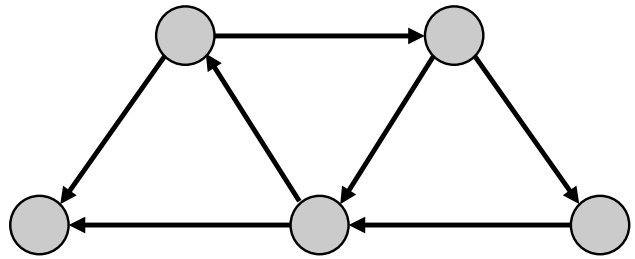
Prova.

Costruiamo un algoritmo prendendo spunto dal lemma. Come?

Lemma. Sia s un nodo. G è fortemente connesso se e solo se ogni nodo è raggiungibile da s, ed s è raggiungibile da ogni nodo.



fortemente connesso



non fortemente connesso

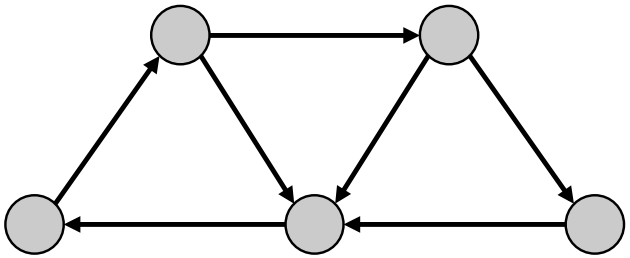
Connettività forte: Algoritmo

Teorema. Possiamo determinare se G è fortemente connesso in tempo O(m + n).

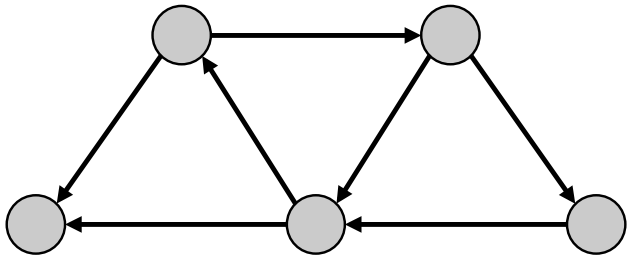
Prova.

- Prendiamo un nodo arbitrario s.
- BFS da s in G.
- Verifichiamo che s è raggiungibile da ogni nodo. Come?

Lemma. Sia s un nodo. G è fortemente connesso se e solo se ogni nodo è raggiungibile da s, ed s è raggiungibile da ogni nodo.



fortemente connesso



non fortemente connesso

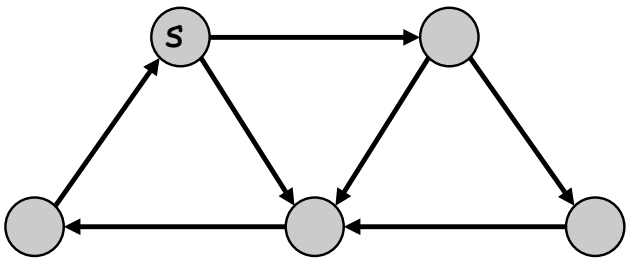
Connettività forte: Algoritmo

Teorema. Possiamo determinare se G è fortemente connesso in tempo O(m + n).

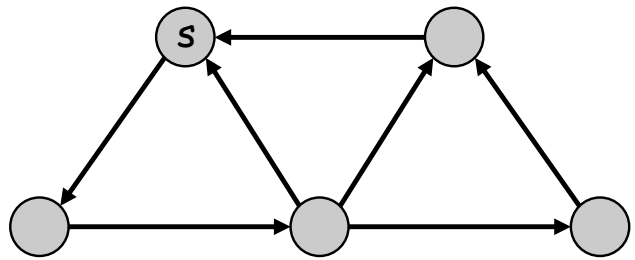
Prova.

- Prendiamo un nodo arbitrario s.
- BFS da s in G.
- BFS da s in G^{rev}. (orientazione al contrario di ogni arco in G)

Esempio



G
(è fortemente connesso)



G^{rev}

Connettività forte: Algoritmo

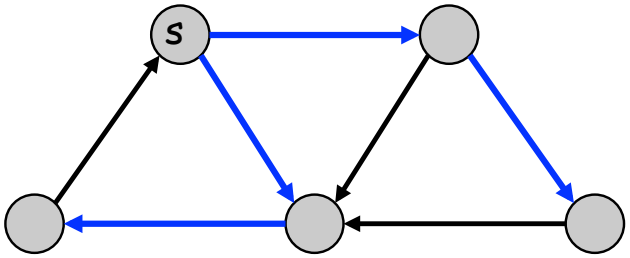
Teorema. Possiamo determinare se G è fortemente connesso in tempo $O(m + n)$.

Prova.

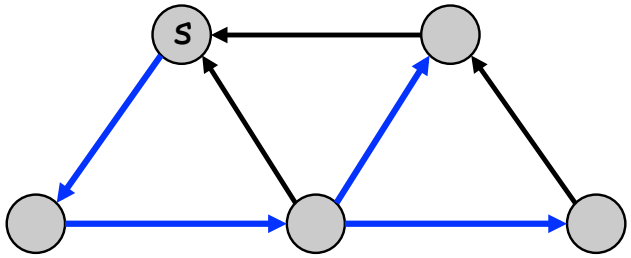
- Prendiamo un nodo arbitrario s.
- BFS da s in G.
- BFS da s in G^{rev}.

orientazione al contrario di ogni arco in G

Esempio



BFS da s in G



BFS da s in G^{rev}

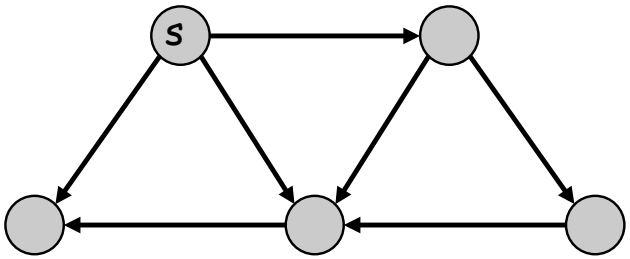
Connettività forte: Algoritmo

Teorema. Possiamo determinare se G è fortemente connesso in tempo O(m + n).

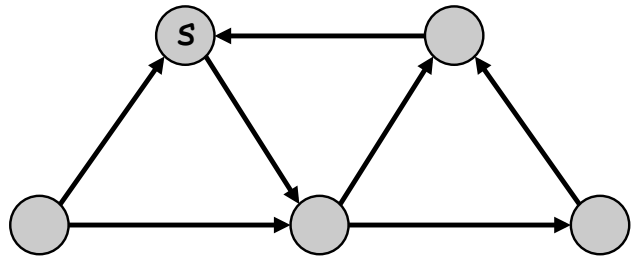
Prova.

- Prendiamo un nodo arbitrario s.
- BFS da s in G.
- BFS da s in G^{rev}. (orientazione al contrario di ogni arco in G)

Altro esempio



G
(non è fortemente connesso)



G^{rev}

Connettività forte: Algoritmo

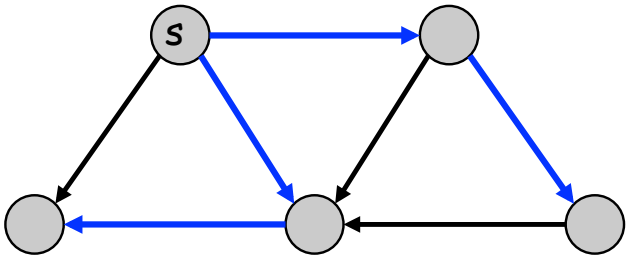
Teorema. Possiamo determinare se G è fortemente connesso in tempo $O(m + n)$.

Prova.

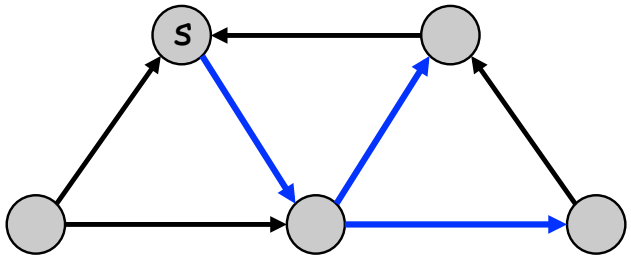
- Prendiamo un nodo arbitrario s.
- BFS da s in G.
- BFS da s in G^{rev}.

orientazione al contrario di ogni arco in G

Altro esempio



BFS da s in G



BFS da s in G^{rev}

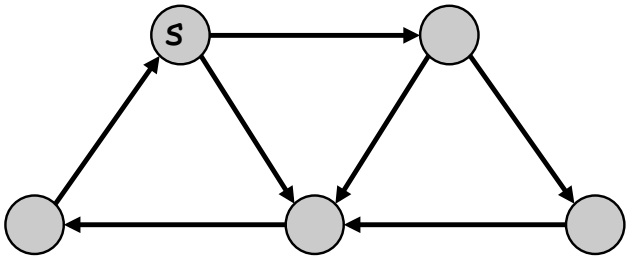
(non è fortemente connesso)

Connettività forte: Algoritmo

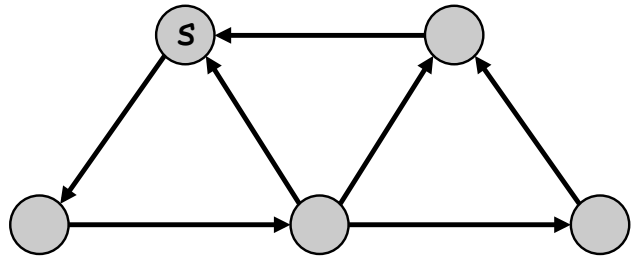
Teorema. Possiamo determinare se G è fortemente connesso in tempo O(m + n).

Prova.

- Prendiamo un nodo arbitrario s.
- BFS da s in G.
- BFS da s in G^{rev}. (orientazione al contrario di ogni arco in G)
- Ritorna vero se e solo se tutti i nodi sono raggiunti in entrambe le esecuzioni BFS.
- La correttezza segue immediatamente dal lemma precedente.



G (fortemente connesso)



G^{rev}

3.6 DAGs and Topological Ordering

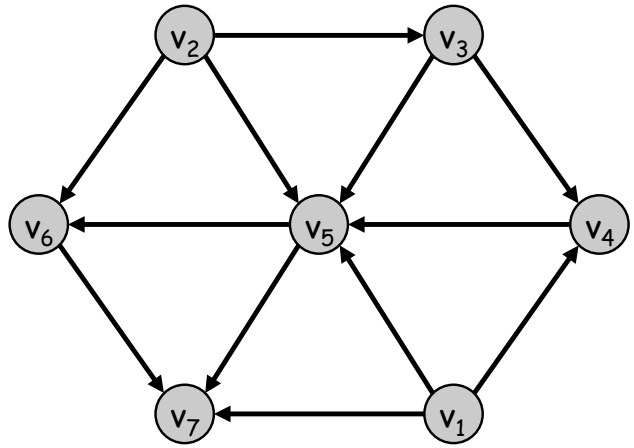
Directed Acyclic Graphs (DAG)

Definizione. Un DAG è un grafo diretto che non contiene cicli diretti.

Vincoli precedenza. Arco (v_i, v_j) significa che task v_i deve venire prima di v_j.

Applicazioni.

- Grafo propedeuticità corsi: corso v_i deve essere sostenuto prima di v_j.
- Compilazione: modulo v_i deve essere compilato prima di v_j.
- Sequenza di computing job: output del job v_i necessario per determinare l' input del job v_j.

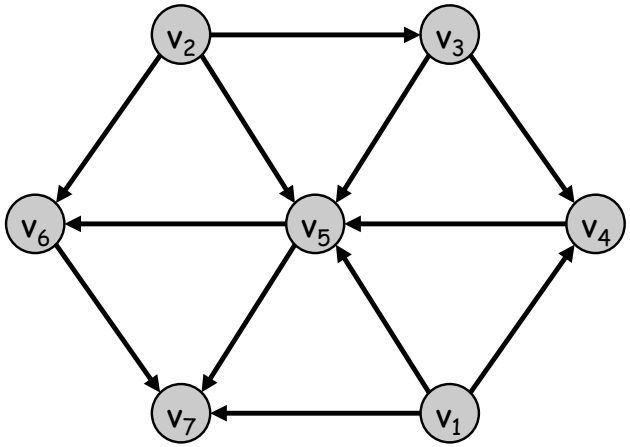


un DAG

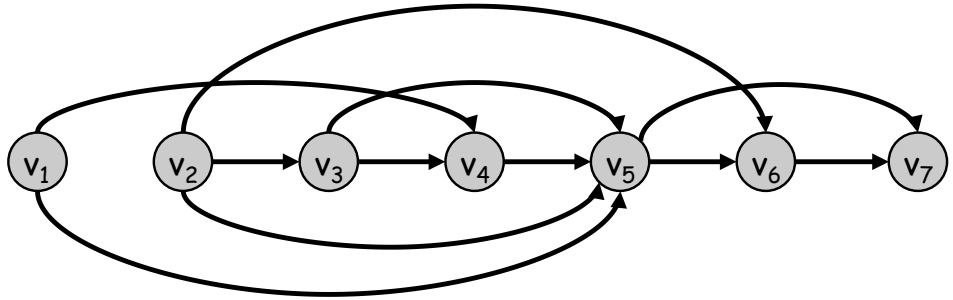
Directed Acyclic Graphs (DAG)

Definizione. Un DAG è un grafo diretto che non contiene cicli diretti.

Definizione. Un ordine topologico di un grafo diretto $G = (V, E)$ è un ordinamento dei suoi nodi $v_1, v_2, \ldots, v_n$ così che per ogni arco (v_i, v_j) si ha $i<j$.



un DAG



un ordinamento topologico

Directed Acyclic Graphs (DAG)

Definizione. Un DAG è un grafo diretto che non contiene cicli diretti.

Definizione. Un ordine topologico di un grafo diretto $G = (V, E)$ è un ordinamento dei suoi nodi $v_1, v_2, \ldots, v_n$ così che per ogni arco (v_i, v_j) si ha $i<j$.

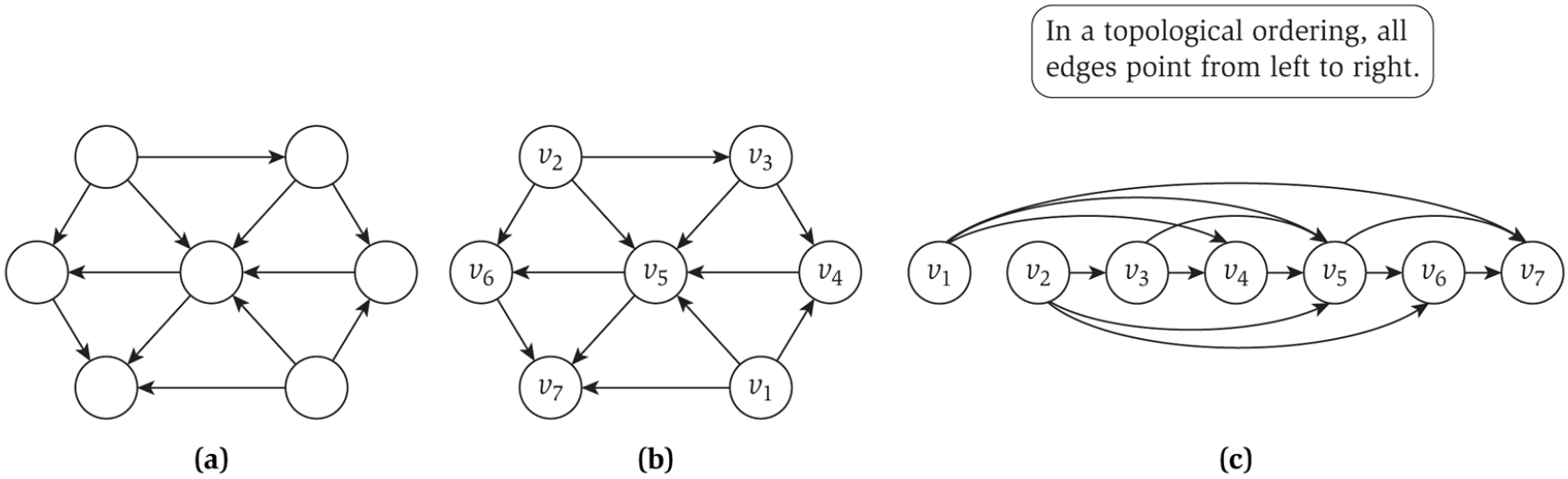


Figure 3.7 (a) A directed acyclic graph. (b) The same DAG with a topological ordering, specified by the labels on each node. (c) A different drawing of the same DAG, arranged so as to emphasize the topological ordering.

Directed Acyclic Graphs (DAG)

Definizione. Un DAG è un grafo diretto che non contiene cicli diretti.

Definizione. Un ordine topologico di un grafo diretto $G = (V, E)$ è un ordinamento dei suoi nodi $v_1, v_2, \ldots, v_n$ così che per ogni arco (v_i, v_j) si ha $i<j$.

Domanda. Ogni DAG ha un ordinamento topologico?

Domanda. Se c'è, come lo troviamo in modo efficiente?

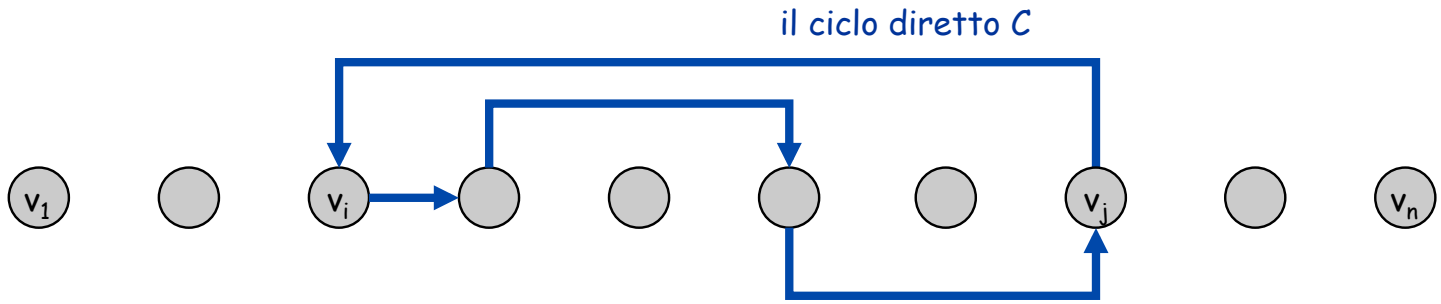
Faremo vedere che dato un grafo diretto G:
G è un DAG se e solo se G ha un ordinamento topologico.

Directed Acyclic Graphs

Lemma. Se G ha un ordinamento topologico allora G è un DAG.

Prova. (per assurdo)

- Supponiamo che G ha un ordinamento topologico $v_1, \ldots, v_n$ e che G ha anche un ciclo diretto.
- Sia v_i il nodo in C con l'indice più piccolo, e sia v_j il nodo che precede v_i in C; allora (v_j, v_i) è un arco.
- Dato che i è l' indice più piccolo del ciclo, allora i < j.
- Dato che (v_j, v_i) è un arco e $v_1, \ldots, v_n$ è un ordinamento topologico, allora j < i. Contraddizione.



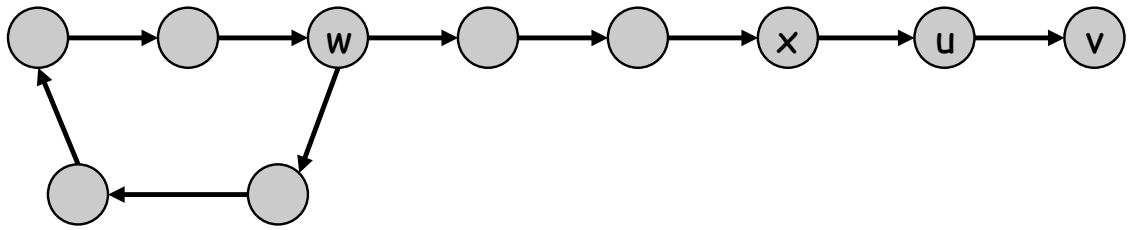
ordine topologico ipotizzato: $v_1, \ldots, v_n$

Directed Acyclic Graphs

Lemma. Se G è un DAG, allora in G vi è almeno un nodo senza archi entranti.

Prova. (per assurdo)

- Supponiamo che G è un DAG e che ogni nodo ha almeno un arco entrante.
- Fissiamo un arbitrario nodo v, e seguiamo archi a ritroso da v. Dato che v ha almeno un arco entrante (u, v), possiamo andare indietro ad u.
- Poi, dato che u ha almeno un arco entrante (x, u), possiamo andare indietro ad x.
- E cosi via fino a visitare un nodo (sia esso w) due volte.
- Sia C la sequenza di nodi incontrati tra due visite successive a w. C è un ciclo.



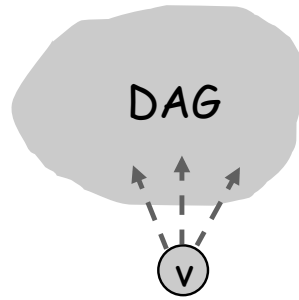
Directed Acyclic Graphs

Lemma. Se G è un DAG, allora G ha un ordinamento topologico.

Prova. (per induzione su n)

- Caso base: vero per n = 1.
- Dato un DAG con n > 1 nodi, prendiamo un nodo v senza archi entranti
- G - { v } è un DAG, dato che la cancellazione di v non può creare cicli
- Dall'ipotesi induttiva, G - { v } ha un ordinamento topologico
- Poniamo v all'inizio dell' ordinamento topologico; poi appendiamo i nodi di G - { v } in ordine topologico
- La sequenza risultante è un ordinamento topologico di G.

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: running time

```
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
```

- Come computiamo in modo efficiente un nodo senza archi entranti?
- Mantenere la seguente informazione:
  - `count[w]` = numero restante di archi entranti
  - S = insieme dei nodi rimanenti senza archi entranti
- Inizializzazione: $O(m + n)$ con una singola visita al grafo
- Aggiornamento: cancellazione di v
  - Rimuovere v da S
  - Decrementare `count[w]` per tutti gli archi da v a w, e aggiungere w ad S se `count[w]` = 0
  - Costo: $O(1)$ per arco

Algoritmo per l'ordinamento topologico: running time

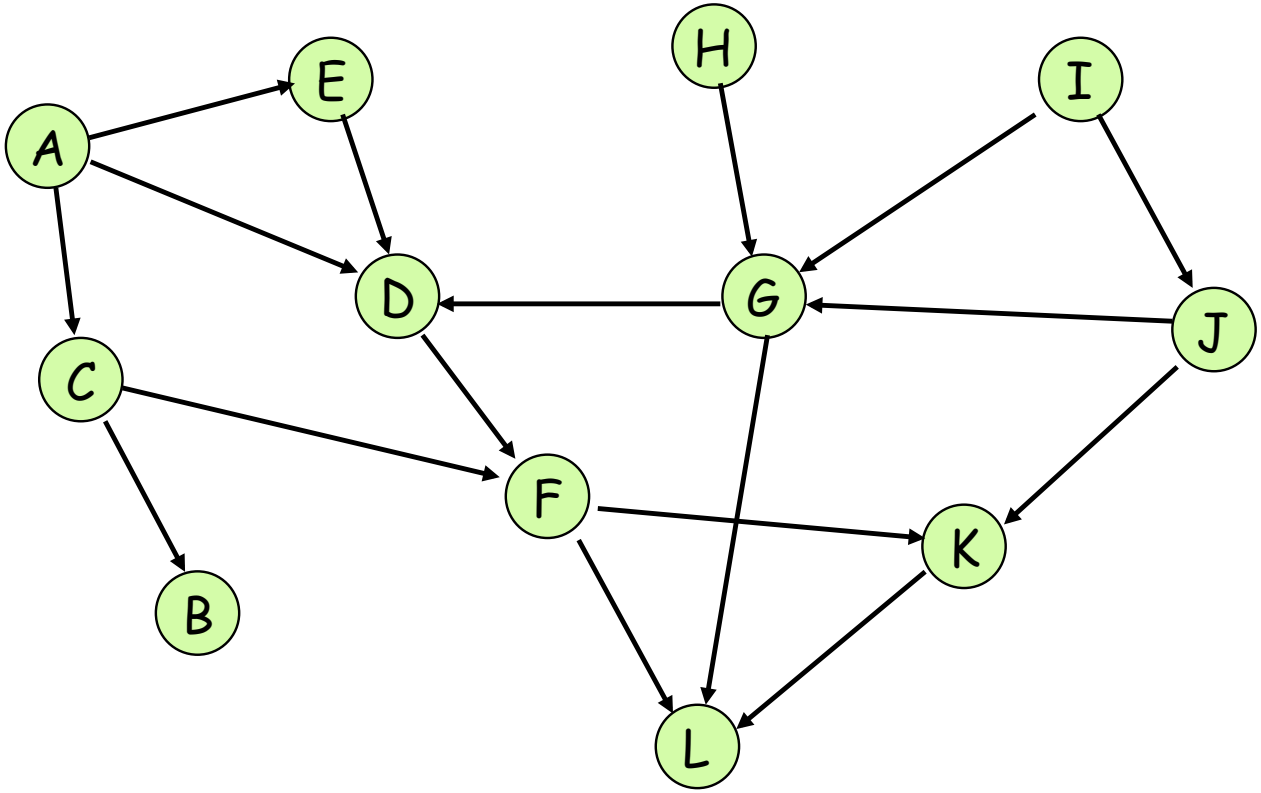
To compute a topological ordering of G:
Find a node v with no incoming edges and order it first
Delete v from G
Recursively compute a topological ordering of $G-\{v\}$
  and append this order after v

- Come computiamo in modo efficiente un nodo senza archi entranti?
- Mantenere la seguente informazione:
  - `count[w]` = numero restante di archi entranti
  - S = insieme dei nodi rimanenti senza archi entranti
- Inizializzazione: $O(m + n)$ con una singola visita al grafo
- Aggiornamento: cancellazione di v
  - Rimuovere v da S
  - Decrementare `count[w]` per tutti gli archi da v a w, e aggiungere w ad S se `count[w]` = 0
  - Costo: $O(1)$ per arco

Teorema. L'algoritmo trova un ordinamento topologico in tempo $O(m + n)$.

Ordinamento topologico: esercizio

Trovare un ordinamento topologico del seguente grafo



Solved Exercise 1, page 104

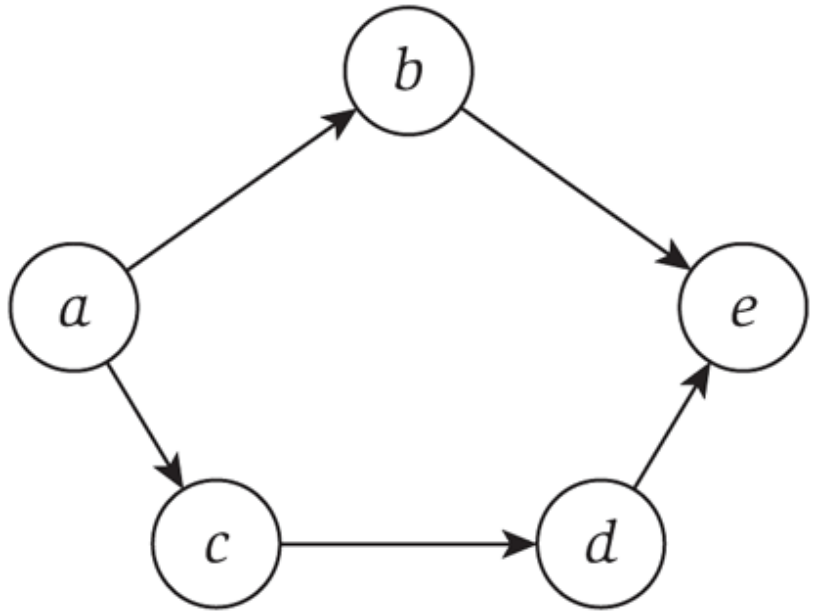


Figure 3.9 How many topological orderings does this graph have?

Solved Exercise 1, page 104

Sono $\binom{3}{1}$

- a bcd e
- a cbd e
- a cdb e

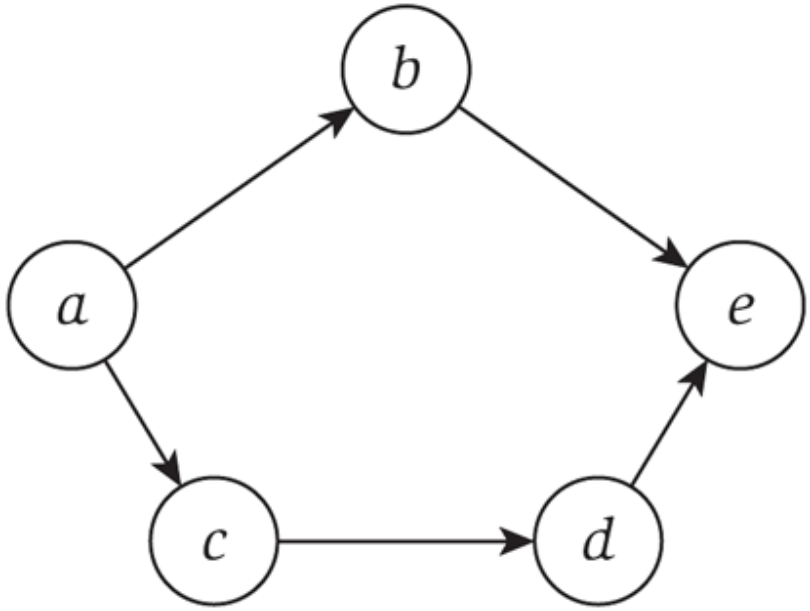


Figure 3.9 How many topological orderings does this graph have?

Exercise 1, page 107

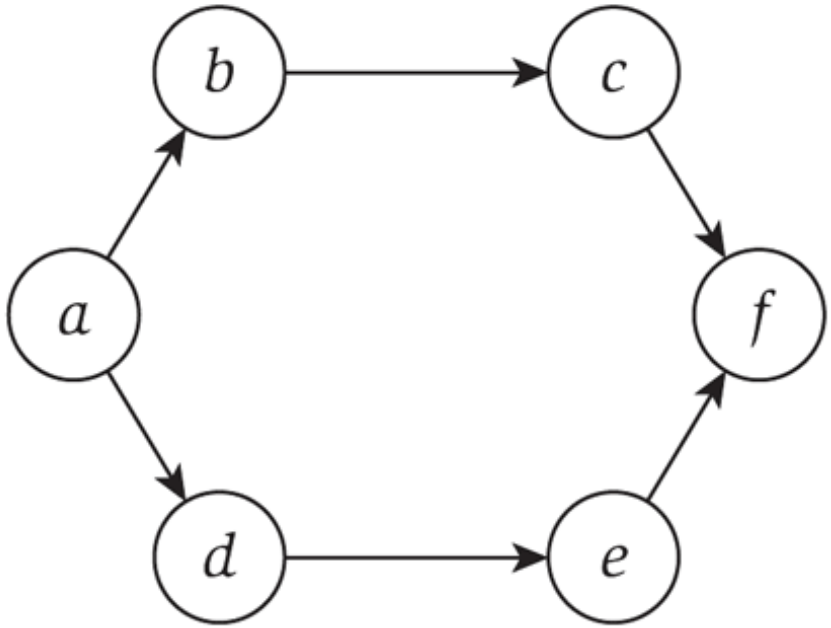


Figure 3.10 How many topological orderings does this graph have?

Exercise 1, page 107

Sono $\binom{4}{2} = 6$

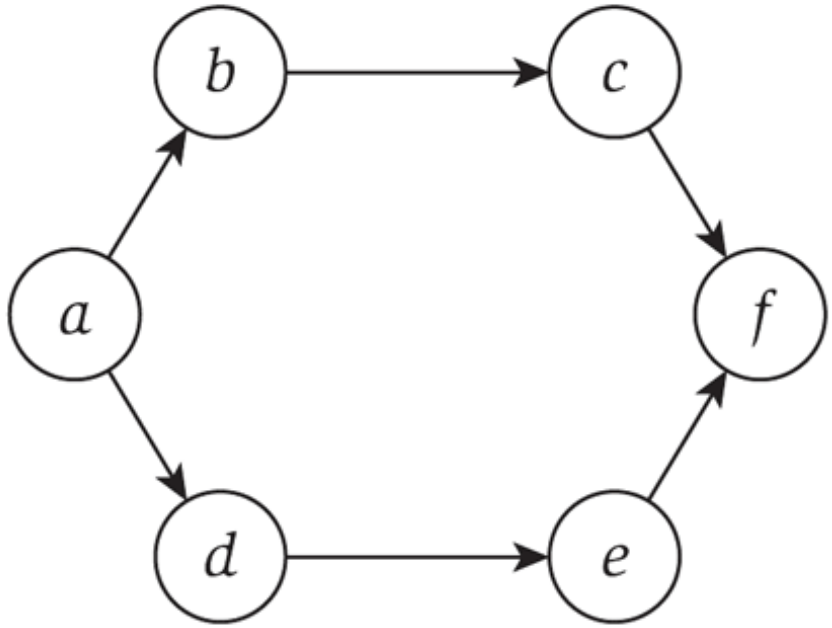


Figure 3.10 How many topological orderings does this graph have?

- a bcde f
- a bdce f
- a bdec f
- a dbce f
- a dbec f
- a debc f

Riepilogo Capitolo 3, Graphs

3.1 Basic Definitions and Applications
3.2 Graph Traversal
3.3 Implementing Graph Traversal Using Queues and Stacks (in parte)
3.4 Testing Bipartiteness
3.5 Connectivity in Directed Graphs
3.6 DAGs and Topological Ordering