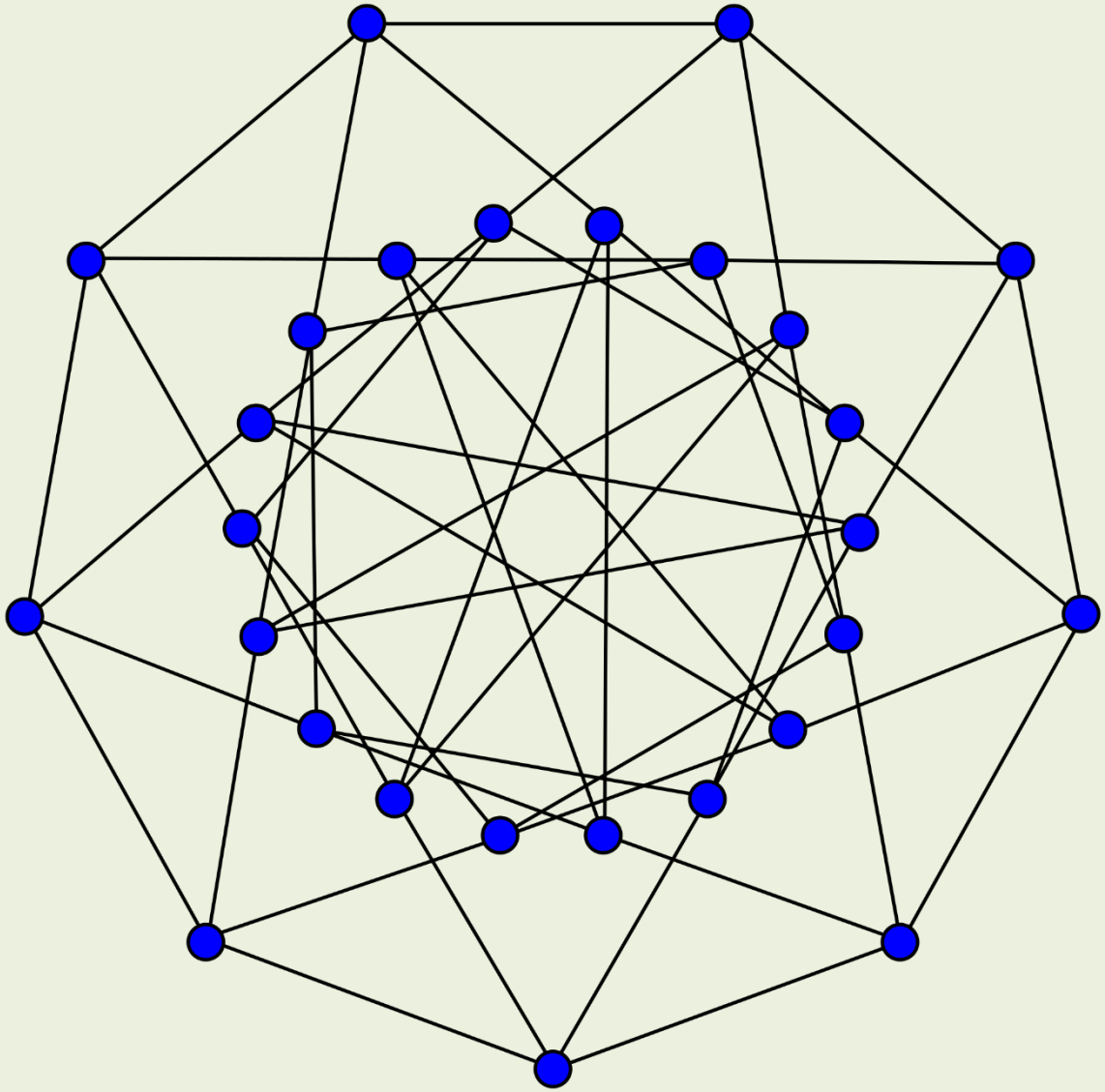
Grafi:
visite e applicazioni

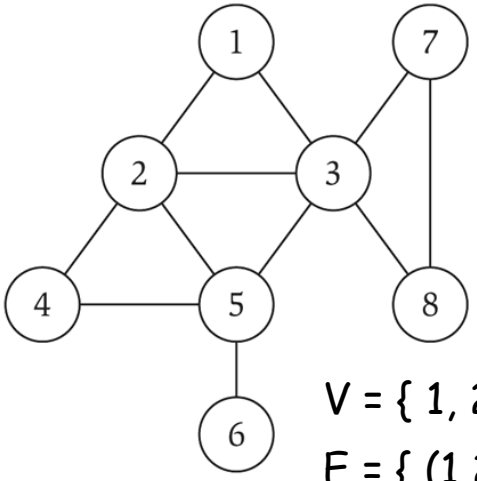
28 Aprile 2023



Grafi (non orientati)

Grafo (non orientato): G = (V, E)

- V = nodi (o vertici)
- E = archi fra coppie di nodi distinti.
- Modella relazioni fra coppie di oggetti.
- Parametri della taglia di un grafo: n = |V|, m = |E|.



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

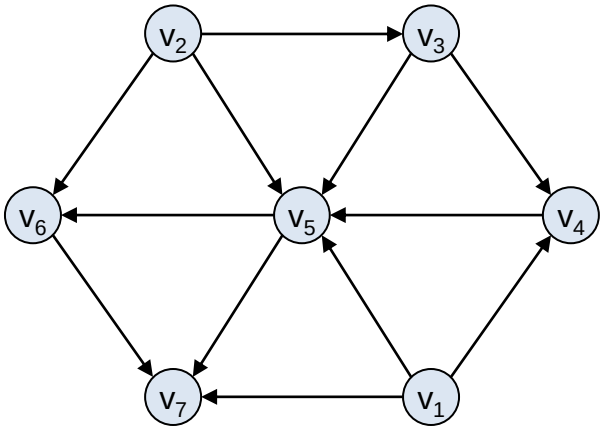
E = { (1,2), (1,3), (2,3), (2,4), (2,5), (3,5), (3,7), (3,8), (4,5), (5,6), (7,8) }

n = 8

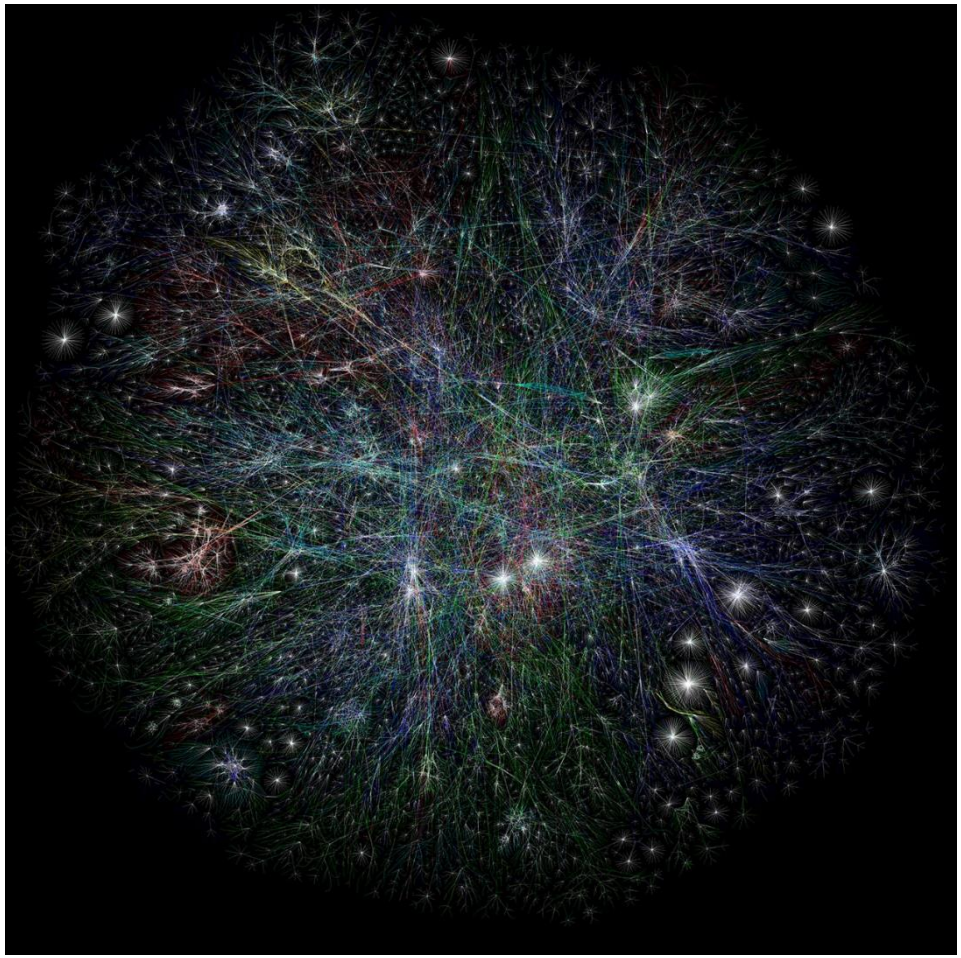
m = 11

Grafi diretti (o orientati)

- Grafo diretto: G = (V, E)
- V = nodi (o vertici)
- E = archi diretti da un nodo (coda) in un altro (testa)
- Modella relazioni non simmetriche fra coppie di oggetti.
- Parametri della taglia di un grafo: n = |V|, m = |E|.



Mappa di Internet

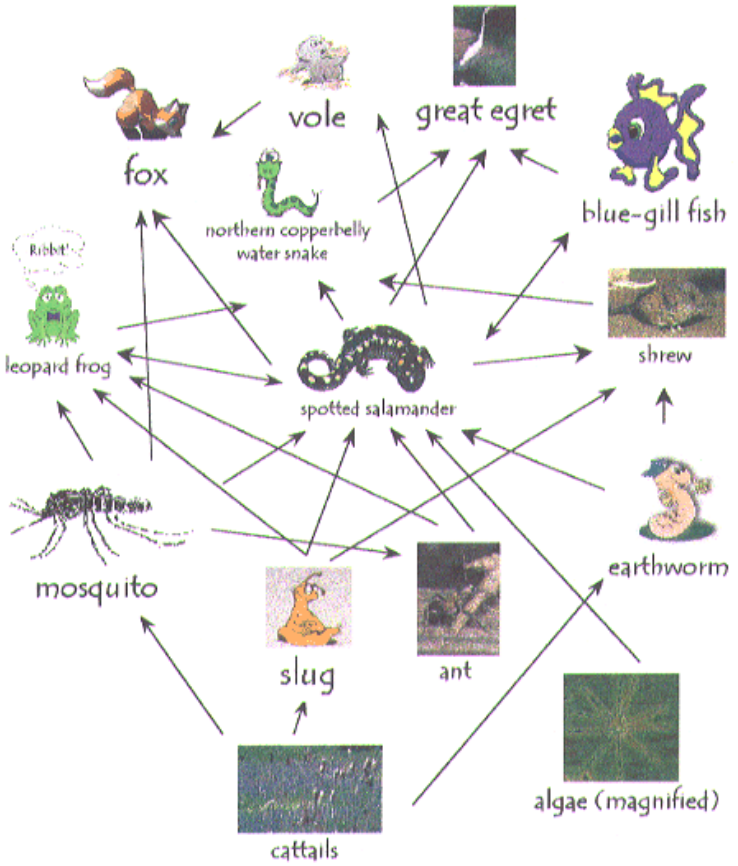


https://blog.kaspersky.it/amazing-internet-maps/6861/

Ecological Food Web

Food web (grafo orientato).

- nodo = specie.
- arco = dalla preda al predatore



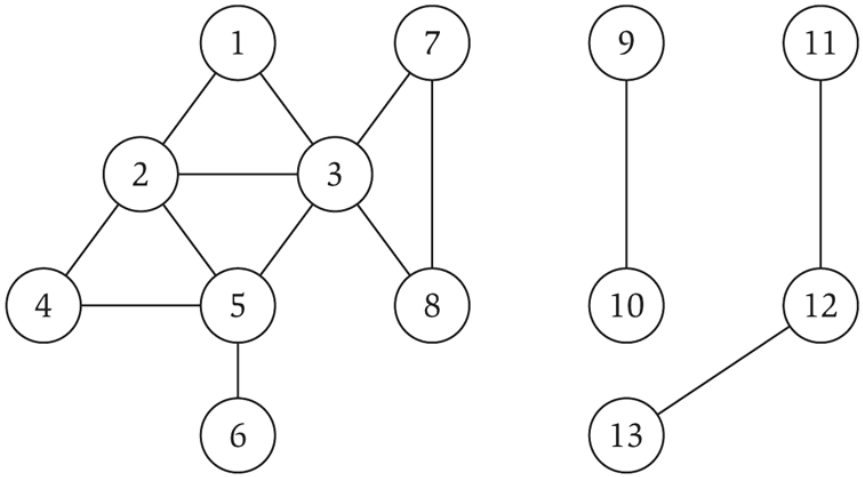
Reference: http://www.twingroves.district96.k12.il.us/Wetlands/Salamander/Salgrafoics/salfoodweb.giff

Un pò di terminologia ……

Def. In un grafo G = (V, E) se l'arco (u,v) $\in$E allora diremo che:

- l'arco è incidente u e v
- u e v sono gli estremi dell'arco
- u e v sono adiacenti
- (u,v) è un arco uscente da u.

Def. Il grado ("degree") di un nodo u, indicato deg(u), è il numero di archi uscenti da u.



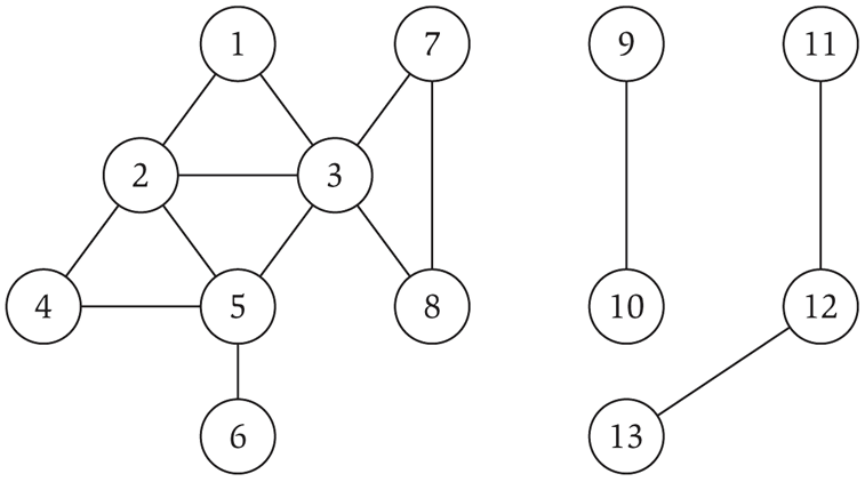
Cammini

Def. Un cammino in un grafo $G = (V, E)$ è una sequenza 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. La sua lunghezza è k-1.

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

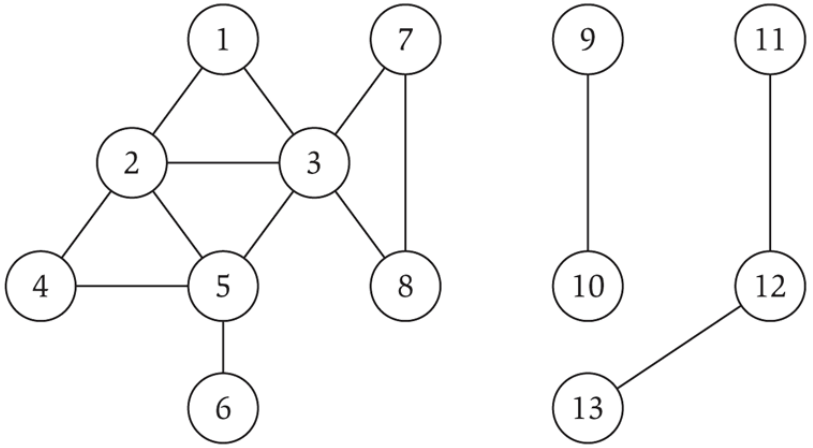
Def. Un nodo v è raggiungibile dal nodo u, se esiste un cammino da u a v.

Def. Un grafo è connesso se per ogni coppia di nodi u e v, c'è un cammino fra u e v.



Componenti connesse

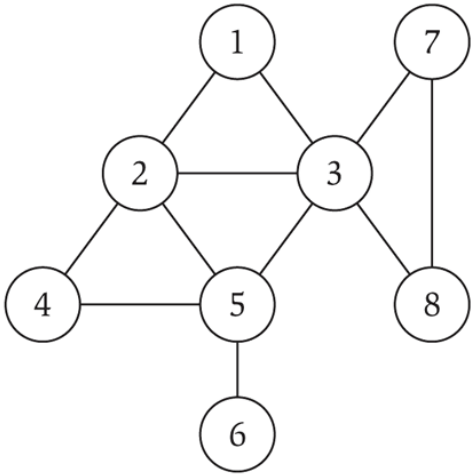
- Def. La componente connessa di un grafo che contiene un nodo s è l'insieme dei nodi raggiungibili da s.



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

Cicli

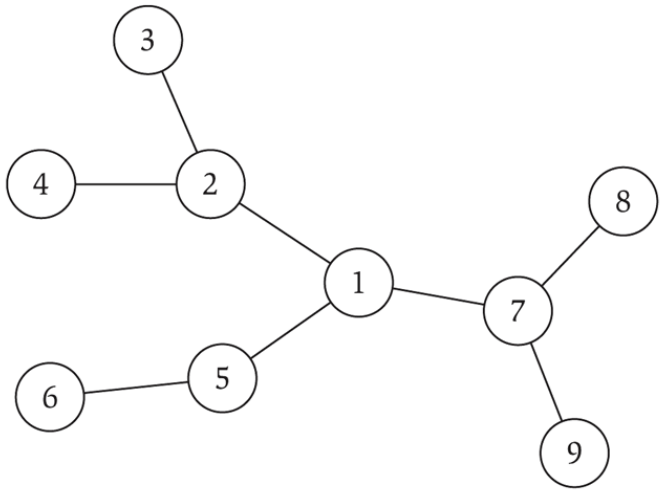
Def. Un ciclo è un cammino $v_1, v_2, \ldots, v_{k-1}, v_k$ nel quale $v_1 = v_k$, $k > 3$, e i primi k-1 nodi sono tutti distinti.



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

Alberi

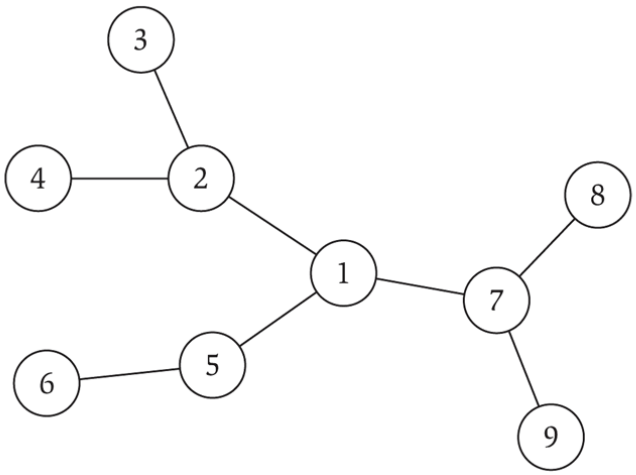
Def. Un grafo è un albero (libero) se è connesso e non contiene un ciclo.



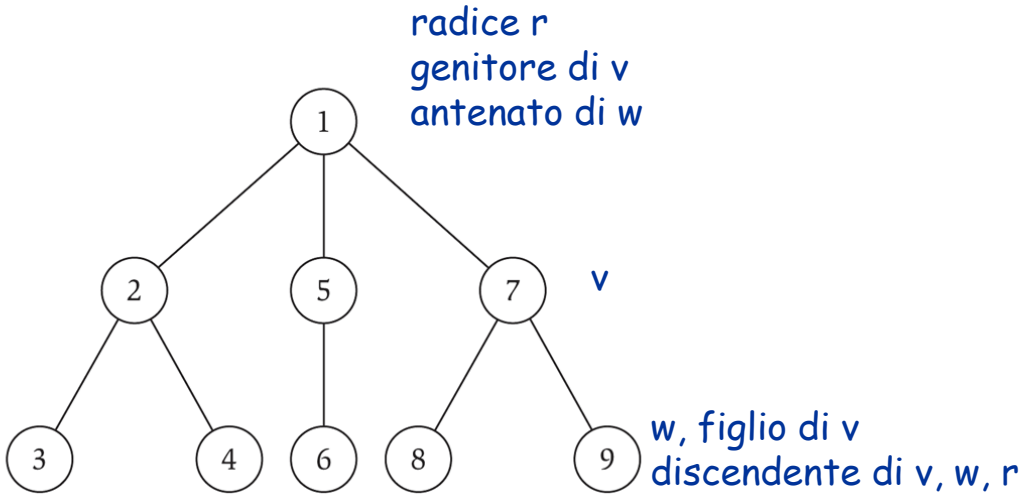
Alberi radicati

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

Importanza: Modella una struttura gerarchica



un albero



lo stesso albero, radicato in 1

Programma su grafi

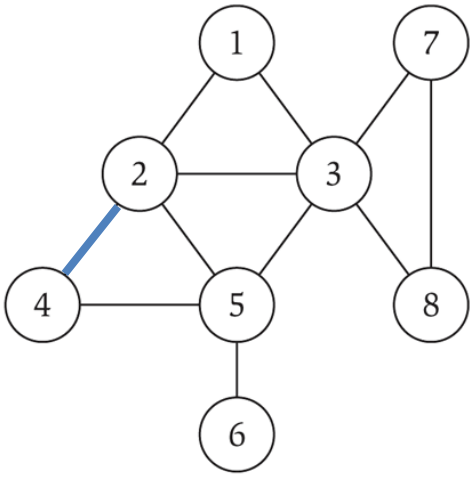
- Definizioni. Problemi di connettività e sulle componenti connesse. Visite BFS e DFS. Problemi di connessione in grafi orientati. Test per grafi bipartiti.
- Ordine topologico in DAG.

- Calcolo di cammini minimi: algoritmo di Dijkstra (*greedy*) (par. 4.4). Calcolo di cammini minimi con costi anche negativi: algoritmo di Bellman-Ford (*programmazione dinamica*) (par. 6.8)
- Albero di ricoprimento minimo: algoritmi di Prim e di Kruskal (*greedy*) (par. 4.5). Applicazione di MST: clustering (par. 4.7)

Rappresentazione di un grafo: Matrice di adiacenza

Matrice di adiacenza: matrice $n \times n$ con $A_{uv} = 1$ se (u, v) è un arco.

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

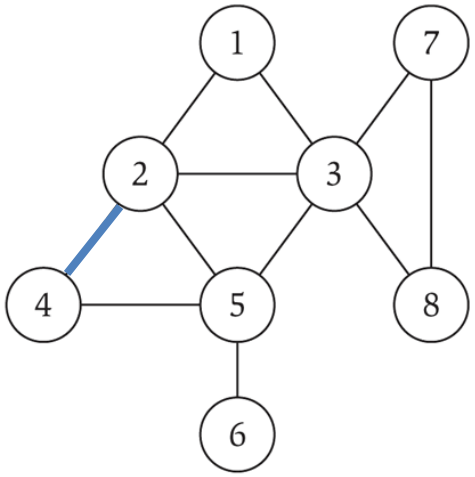


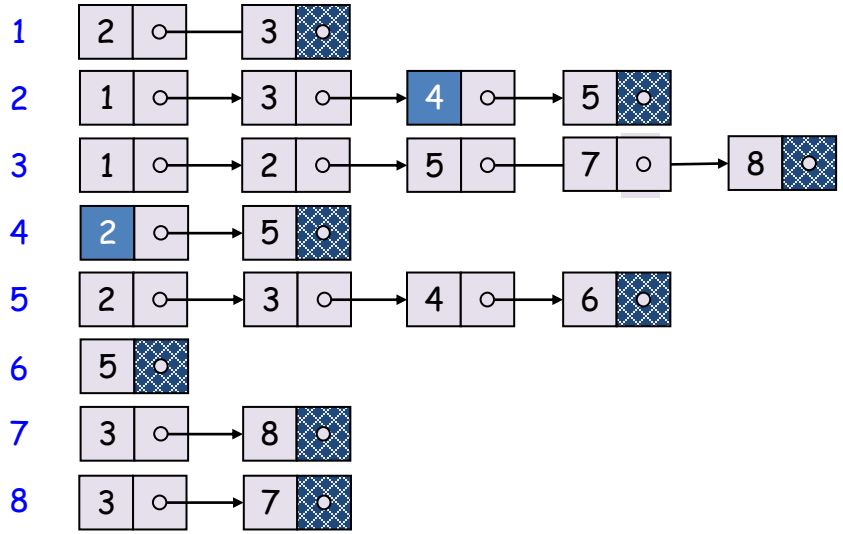
	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	0	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

Rappresentazione di un grafo: Lista di adiacenza

Lista di adiacenza: array di n liste indicizzate dai nodi.

- Due rappresentazioni di ogni arco.
- Spazio proporzionale ad m + n. (ogni arco compare 2 volte)
- Verificare se (u, v) è un arco richiede tempo O(deg(u)).
- Elencare tutti gli archi richiede tempo $\Theta(m + n)$.
- Elencare tutti gli archi incidenti su un fissato nodo u richiede tempo $\Theta(\deg(u))$.





m=11

Somma lunghezze liste = 2m =22

Confronto fra le rappresentazioni

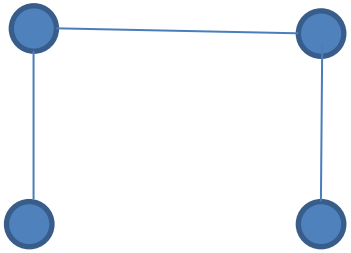
$|V|=n$, $|E|=m$

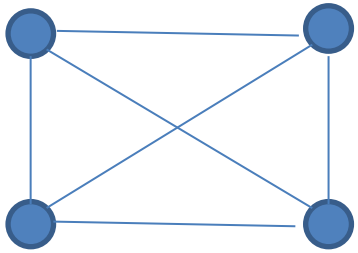
E' maggiore n^2 o $m+n$?

In un grafo connesso con n vertici:
quanti archi posso avere, al minimo e al massimo?

$$n-1 \leq m \leq n(n-1)/2$$

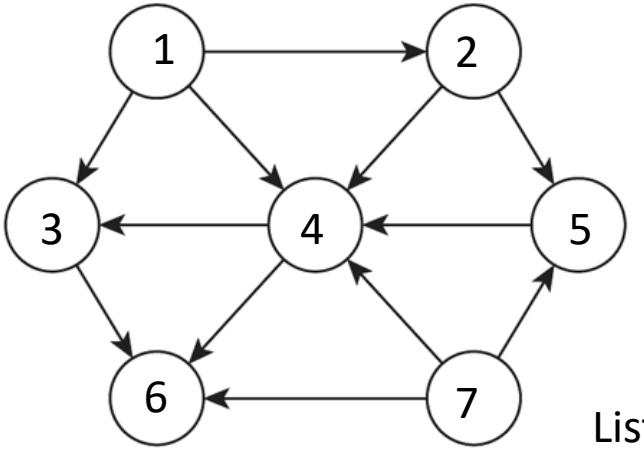
$|E| = \Omega(n)$ $\qquad$ $|E| = O(n^2)$



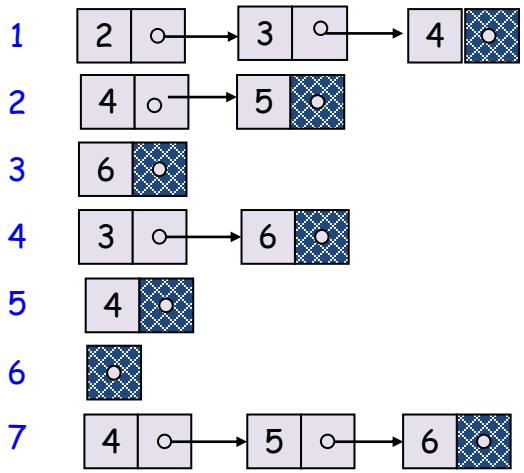


Rappresentazione di un grafo diretto: 2 liste di adiacenza

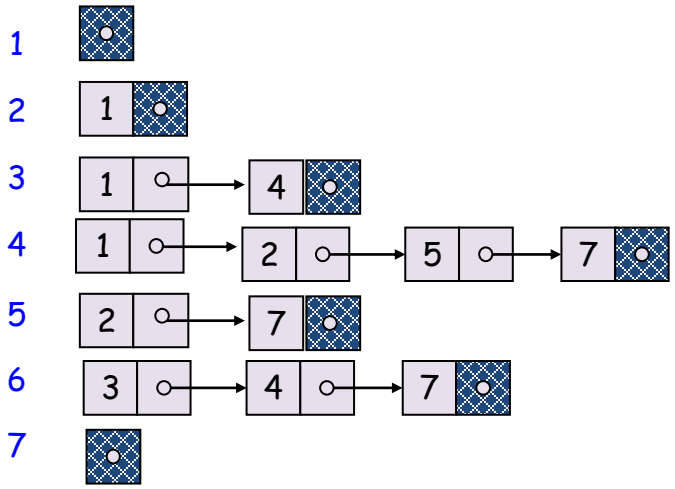
Lista di adiacenza: array di n liste indicizzate dai nodi.



Lista out



Lista in



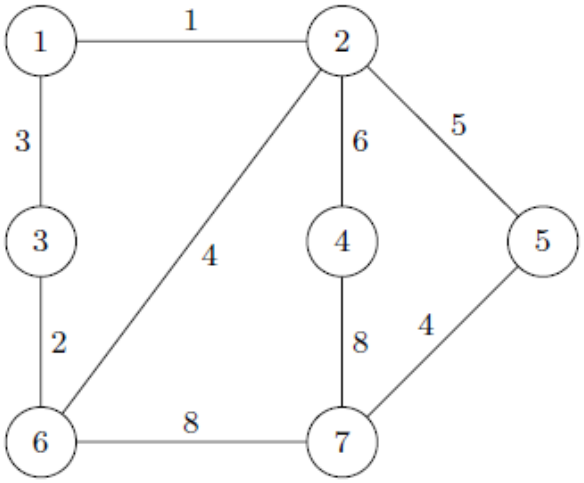
Rappresentazione di un grafo pesato

Sia G = (V, E) un grafo e c : E $\rightarrow$ N una funzione che associa ad ogni arco un peso/costo, intero positivo.

G può essere rappresentato con una matrice M di dimensione n $\times$ n in cui:

$M_{uv} = 0$ se (u, v) non è un arco in E

$M_{uv} = c((u, v))$ altrimenti



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

Grafi:
Visite BFS, DFS

Problemi di connettività in grafi non orientati

Problema della connettività s-t: Dati due nodi s e t, esiste un cammino fra s e t?

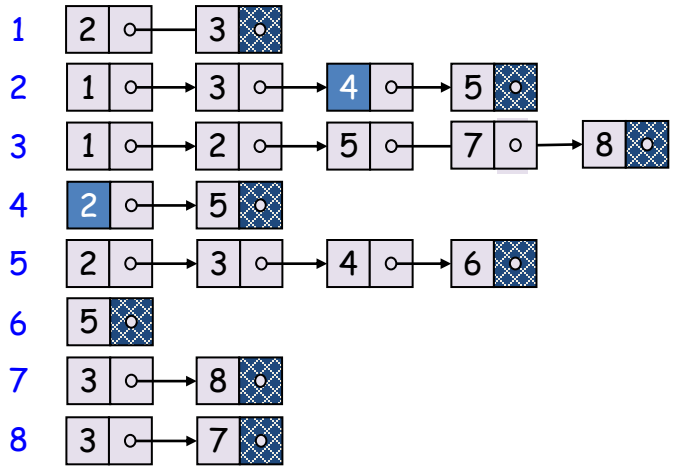
Problema del cammino minimo s-t:

Dati due nodi s e t, qual è la lunghezza del cammino minimo fra s e t?

Diremo: t raggiungibile da s se esiste un cammino fra s e t;

distanza di s da t = lunghezza del cammino minimo fra s e t

	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



Abbiamo due tipi di visite: **Breadth First Search (BFS) e Depth First Search (DFS)**

Problemi risolvibili con BFS o DFS

- Problema della connettività s-t:
  Dati due nodi s e t, esiste un cammino fra s e t?

- Problema del cammino minimo s-t:
  Dati due nodi s e t, qual è la lunghezza del cammino minimo fra s e t?

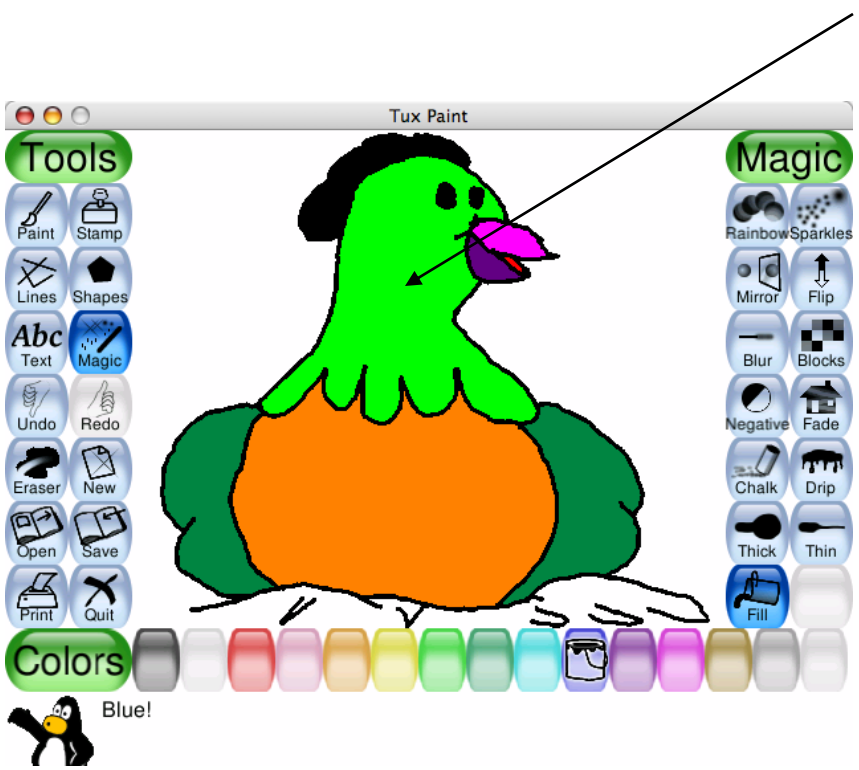
- Problema della componente connessa di s: trovare tutti i nodi raggiungibili da s

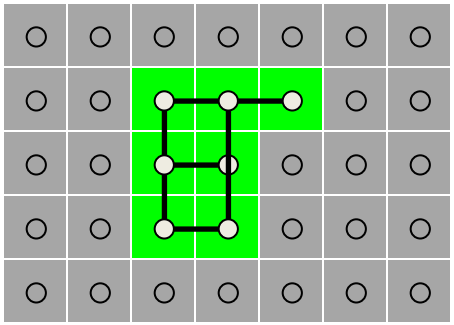
- Problema di tutte le componenti connesse di un grafo G: trovare tutte le componenti connesse di G

Applicazione componenti connesse: Riempimento aree

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

- nodo: pixel.
- arco: fra due pixel adiacenti (nella stessa area)
- L'area da cambiare è una componente connessa

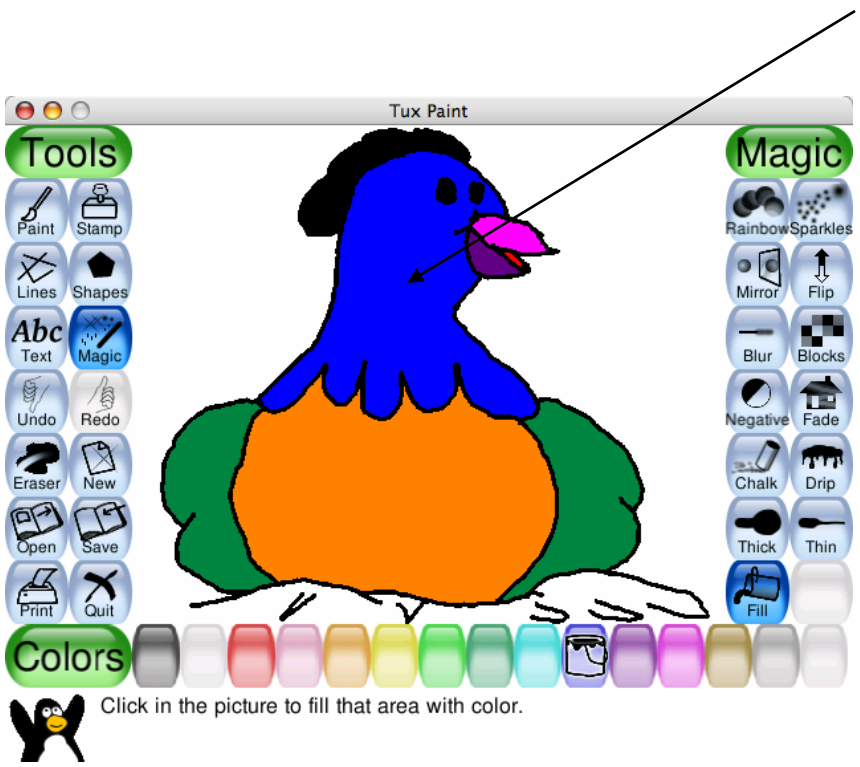


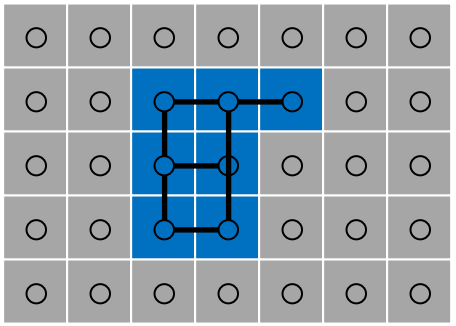


Applicazione componenti connesse: Riempimento aree

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

- nodo: pixel.
- arco: fra due pixel adiacenti (nella stessa area)
- L'area da cambiare è una componente connessa

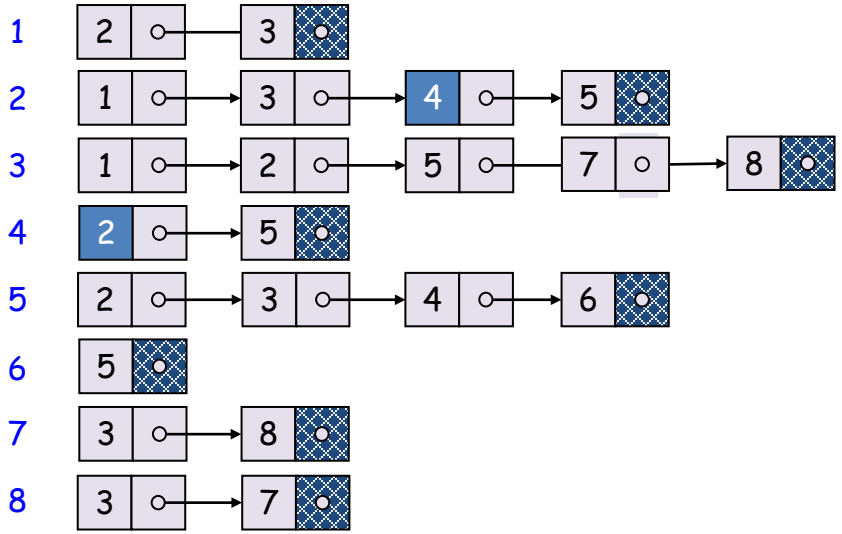




Esplorare un grafo da una sua rappresentazione

What parts of the graph are reachable from a given vertex?

To understand this task, try putting yourself in the position of a computer that has just been given a new graph, say in the form of an adjacency list. This representation offers just one basic operation: finding the neighbors of a vertex. With only this primitive, the reachability problem is rather like exploring a labyrinth (Figure 3.2). You start walking from a fixed place

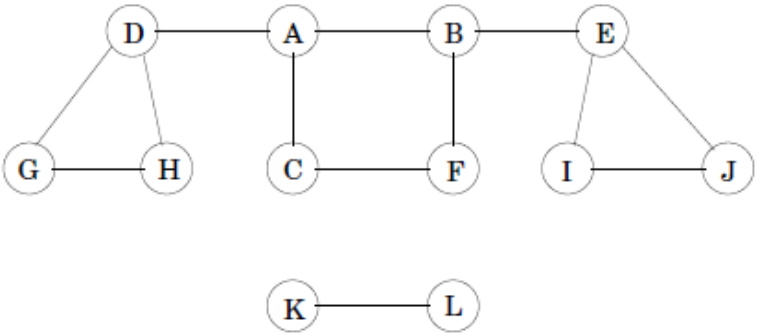


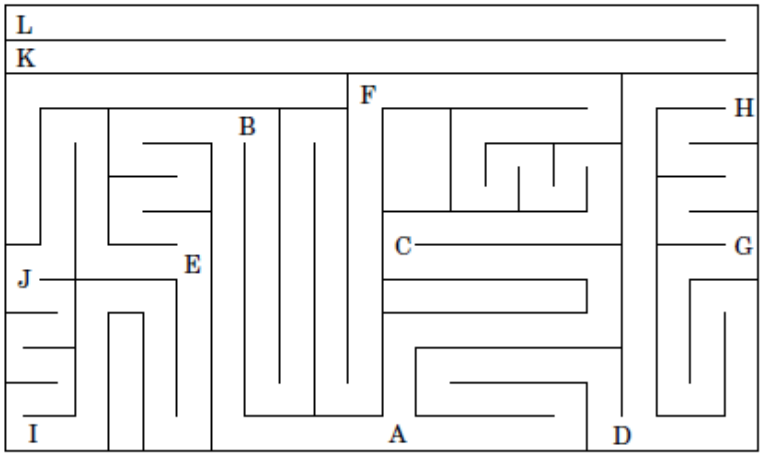
	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	0	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

da [DPV]

(Non) perdersi in un labirinto

Figure 3.2 Exploring a graph is rather like navigating a maze.



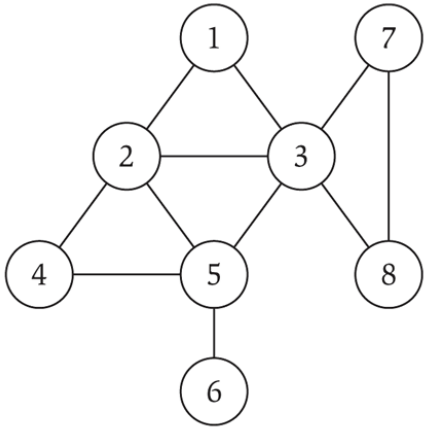


This classic challenge has amused people for centuries. Everybody knows that all you need to explore a labyrinth is a ball of string and a piece of chalk. The chalk prevents looping, by marking the junctions you have already visited. The string always takes you back to the starting place, enabling you to return to passages that you previously saw but did not yet investigate.

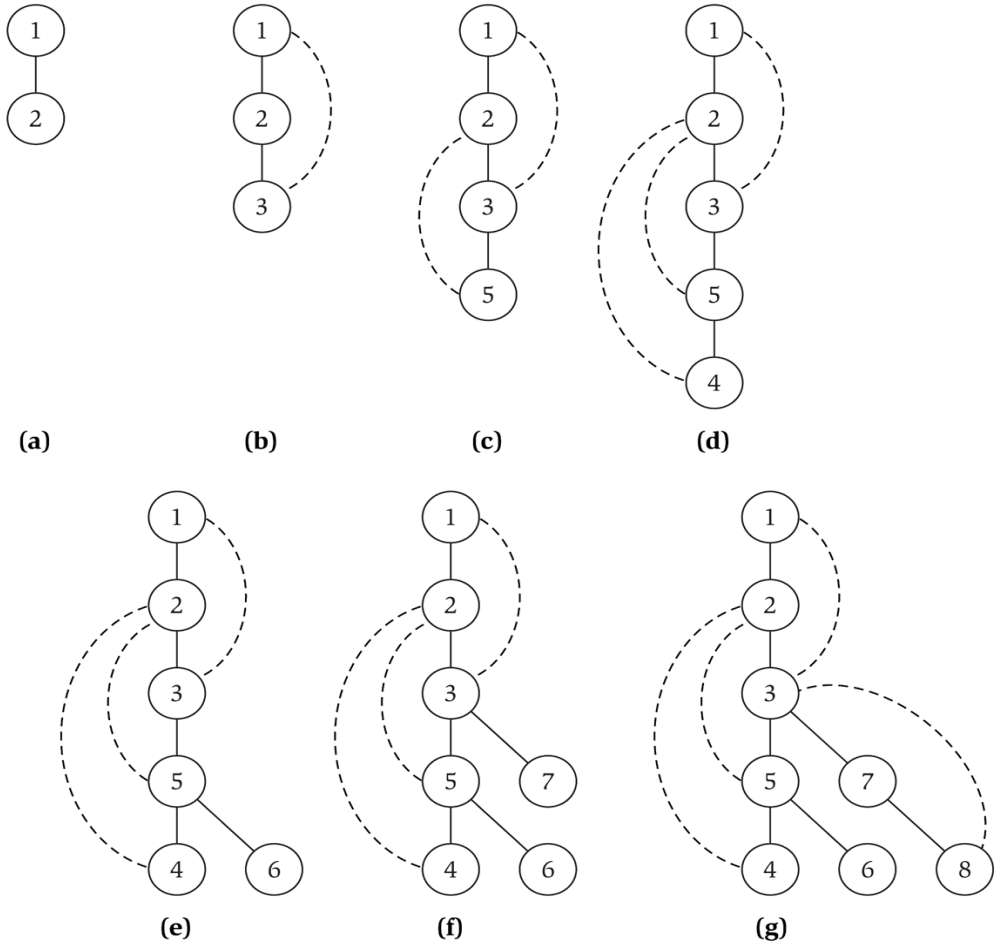
Questa idea è implementata nella DFS

DFS: visita in profondità

Idea di DFS: Esplorare quanto più in profondità possibile e tornare indietro ("backtrack") solo quando è necessario (*come in un labirinto…)*



G



Gli archi non tratteggiati in (g) formano l'albero DFS di G a partire da 1

Algoritmo DFS

Idea di DFS: Esplorare quanto più in profondità possibile e tornare indietro ("backtrack") solo quando è necessario (*come in un labirinto…)*

Algoritmo ricorsivo

R, insieme dei nodi esplorati.

```
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
```

Albero DFS

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

Proprietà 1: Per una fissata chiamata ricorsiva DFS(u), tutti i nodi che sono marcati `Explored` tra l'inizio e la fine della chiamata ricorsiva sono discendenti di u nell'albero DFS (dimostrazione per induzione).

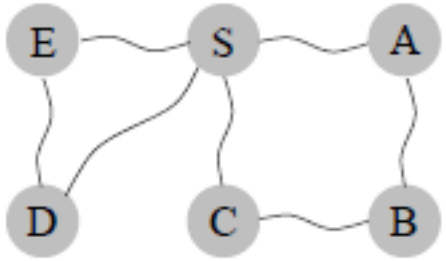
Proprietà 2: Sia T un albero DFS, siano x e y nodi in T si supponga che (x,y) è un arco del grafo, ma non è in T. Allora x è antenato di y, o viceversa.

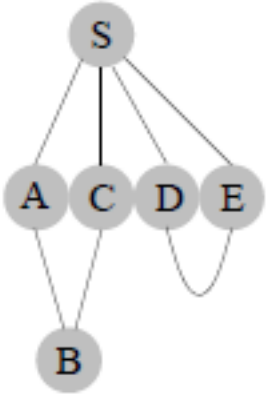
Prova:

Sia x il nodo esplorato per primo, cioè al momento della chiamata DFS(x) y non è `Explored`. Dato che (x,y) è incidente x, l'arco (x,y) verrà considerato. Poiché non è aggiunto a T, significa che y è già `Explored`, quando sarà considerato. Quindi, è stato marcato durante la chiamata DFS(x); e per la Proprietà 1, il nodo y è discendente di x.

BFS: visita in ampiezza

Figure 4.2 A physical model of a graph.



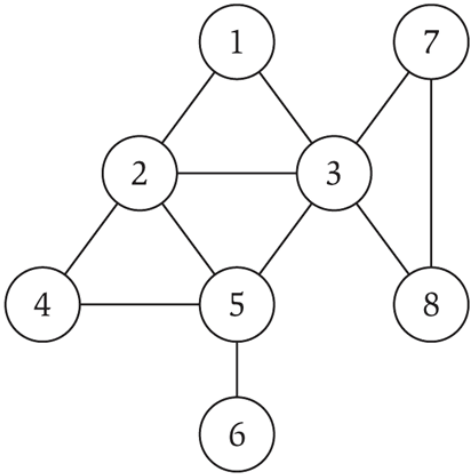


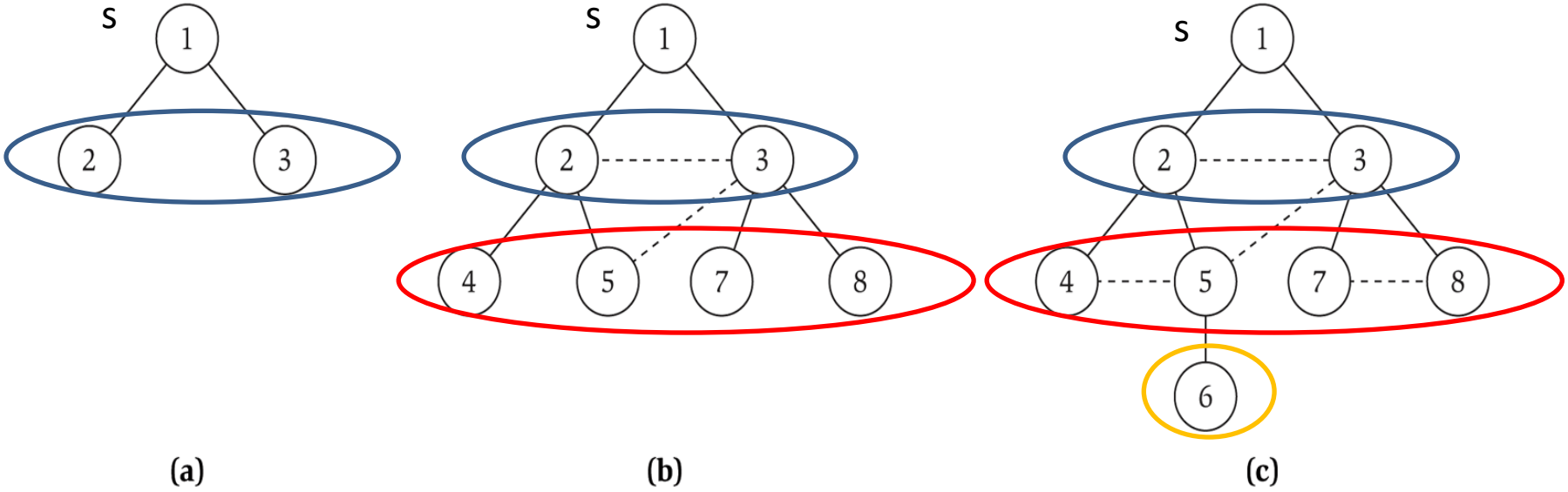
BFS esplora i vertici in ordine di distanza da s.

Da [DPV]

BFS: visita in ampiezza

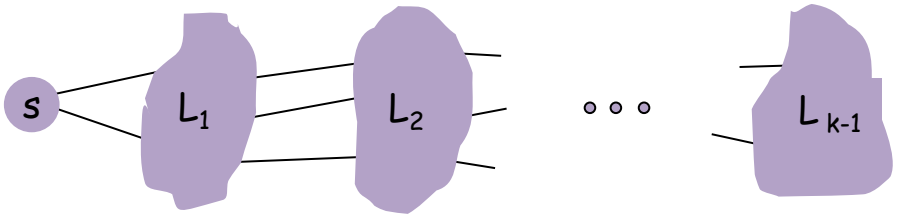
Idea della BFS: Esplorare a partire da s in tutte le possibili direzioni, aggiungendo nodi, uno strato ("layer“) alla volta.





Breadth First Search

L_i sono i layers:



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, e che hanno un arco con un nodo in L_i.

Teorema.

Per ogni i, L_i consiste di tutti i nodi a distanza i da s.

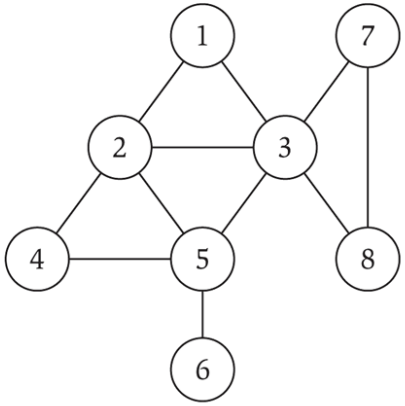
Esiste un cammino da s a t **se e solo se** t appare in qualche layer.

Prova: per induzione su i

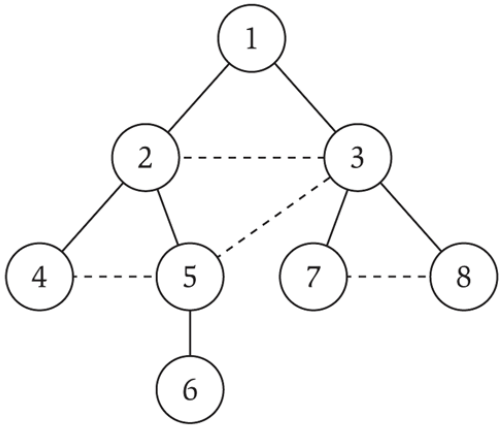
Breadth First Search: proprietà

- Determina i nodi raggiungibili da s (insieme dei layer)
- Determina la loro distanza da s (indice del layer)
- Produce un albero radicato in s: l'albero BFS

  (aggiungo (u,v) quando u $\in L_i$ e v$\notin L_0 \cup \ldots \cup L_i$)



Grafo G



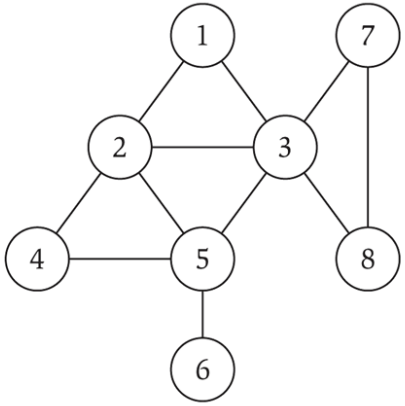
Gli archi non tratteggiati formano l'albero BFS di G

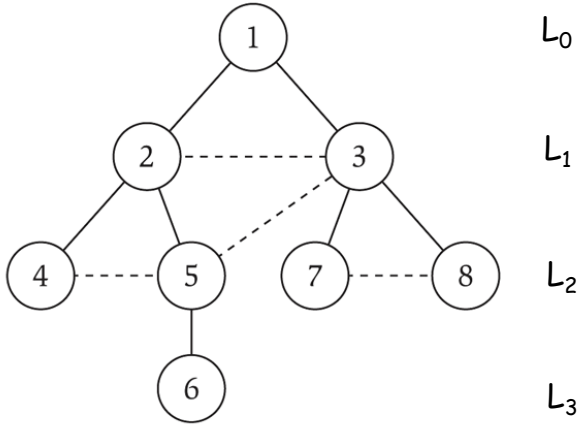
Proprietà dell'albero BFS

Proprietà: Sia T un albero BFS di un grafo G, sia (x, y) un arco di G con x appartenente ad L_i, y ad L_j.

Allora i e j differiscono di al più 1.

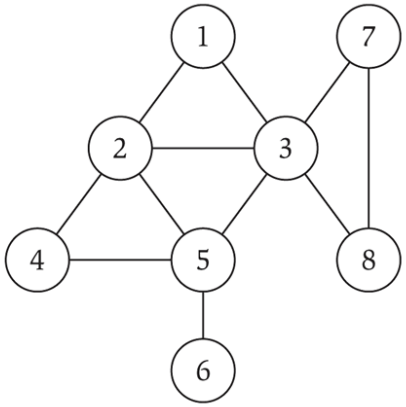
Prova: Supponi $i \leq j$. Quando BFS esamina gli archi incidenti x, o y viene scoperto ora (y in L_{i+1}), o è stato scoperto prima (contro $i \leq j$).



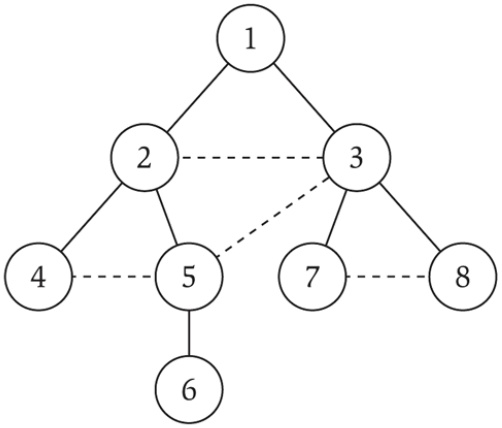


Alberi BFS e DFS

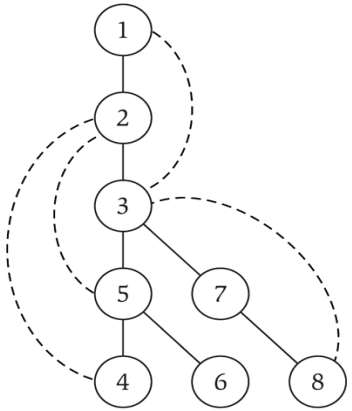
Grafo G



Albero BFS di G



Albero DFS di G



(g)

Implementazioni e complessità

Vedremo (in una prossima lezione) delle implementazioni di BFS e DFS che avranno tempo di esecuzione O(m+n), cioè lineare nella taglia del grafo, se il grafo è rappresentato con una lista delle adiacenze.

Le visite si estendono in maniera naturale ai grafi orientati.

Adesso consideriamo delle applicazioni.

Applicazioni di BFS e DFS

- Problema della connettività s-t:
  Dati due nodi s e t, esiste un cammino fra s e t?

- Problema del cammino minimo s-t:
  Dati due nodi s e t, qual è la lunghezza del cammino minimo fra s e t (ovvero la distanza di s da t)?

- Problema della componente connessa di s: trovare tutti i nodi raggiungibili da s

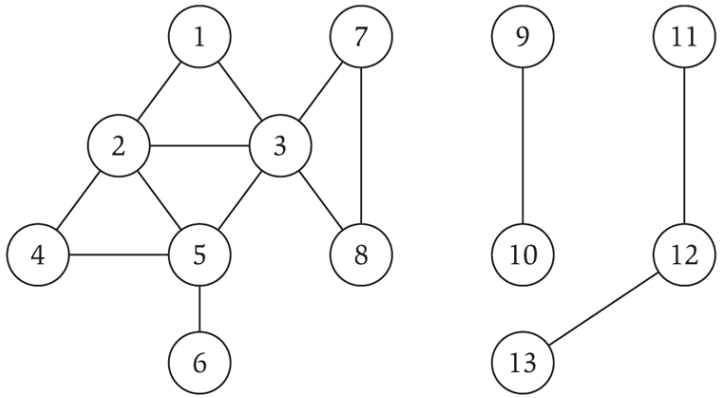
Resta:

- Problema di tutte le componenti connesse di un grafo G: trovare tutte le componenti connesse di G

Relazioni fra componenti connesse

Def. La componente connessa di un grafo che contiene un nodo s è l'insieme dei nodi raggiungibili da s.

Problema: Che relazione c'è fra due componenti connesse?



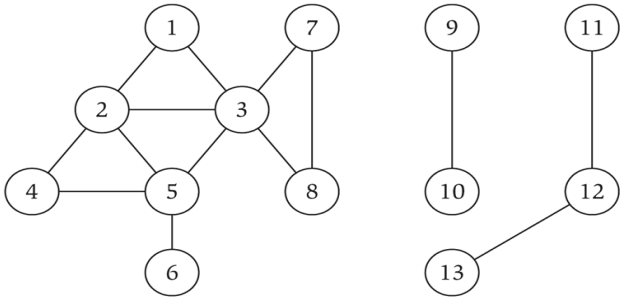
Proprietà: Siano x ed y due nodi in un grafo G. Allora le componenti connesse sono identiche oppure disgiunte

Tutte le componenti connesse

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

Prova . Due casi:

- Esiste un cammino tra x e y: (ogni nodo v raggiungibile da x è raggiungibile da y)
- Non esiste un cammino tra x e y (le componenti sono disgiunte)



Algoritmo per trovare tutte le componenti connesse:

Partendo da s qualsiasi, usa BFS (o DFS) per generare la componente connessa di s. Trova un nodo v non visitato. Se esiste, usa BFS da v per calcolare la componente connessa di v (che sarà una nuova componente, perchè è disgiunta). Ripeti.

Tempo: O(m+n)

In realtà BFS(s) richiede tempo lineare nel numero di nodi e archi della componente connessa di s.

Altre applicazioni di BFS e DFS

- Problema della verifica se un grafo è bipartito (par. 3.4)
- Problema della connessione nei grafi diretti (par. 3.5)

… e altre ancora.