Grafi

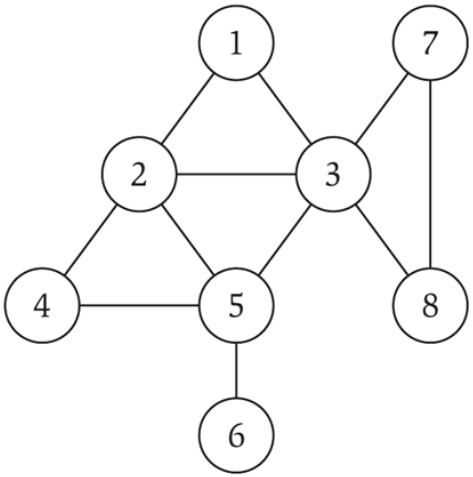
Progettazione di Algoritmi a.a. 2016-17

Matricole congrue a 1

Docente: Annalisa De Bonis

Grafi non direzionati

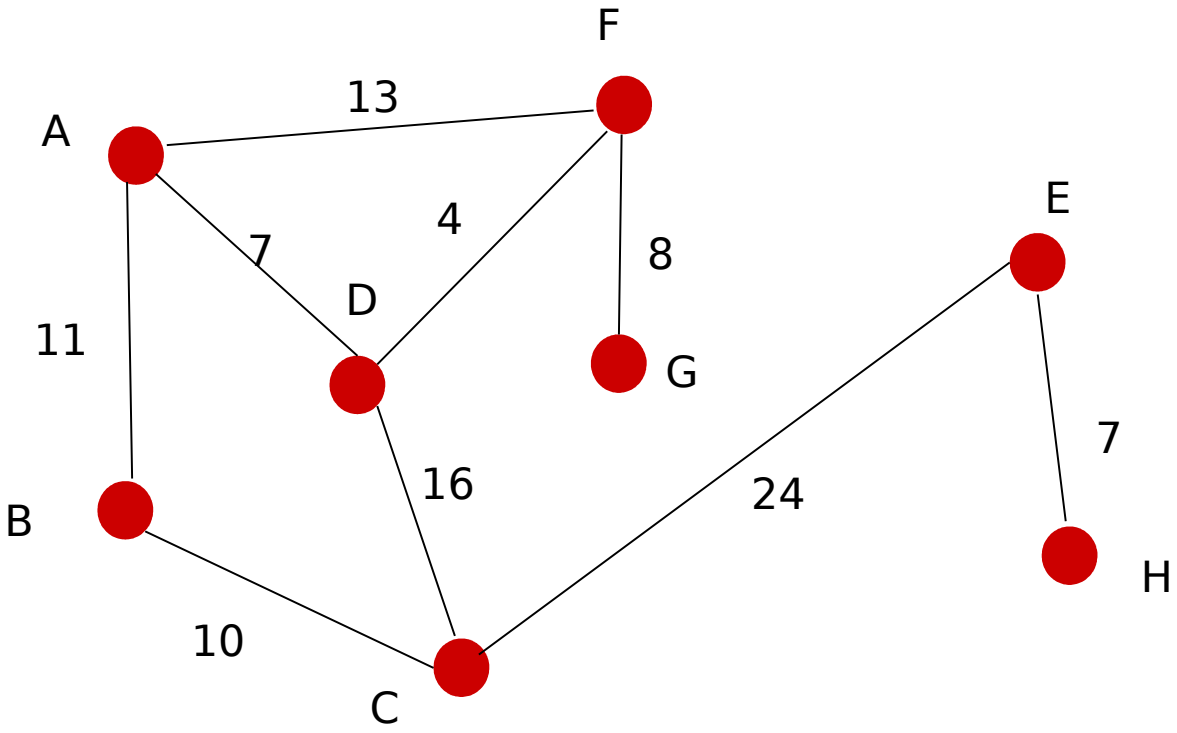
- Grafi non direzionati. G = (V, E)
  - V = insieme nodi.
  - E = insieme archi.
  - Esprime le relazioni tra coppie di oggetti.
  - Parametri del grafo: n = |V|, m = |E|.



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

Esempio di applicazione

- Archi: strade (a doppio senso di circolazione)
- Nodi: intersezioni tra strade
- Pesi archi: lunghezza in km



Alcune applicazioni dei grafi

Grafo	*Nodi*	*Archi*
trasporto	intersezioni di strade	strade
trasporto	aeroporti	voli diretti
comunicazione	computer	cavi di fibra ottica
World Wide Web	web page	hyperlink
rete sociale	persone	relazioni
catena del cibo	specie	predatore-preda
scheduling	task	vincoli di precedenza
circuiti	gate	wire

Alcune applicazione dei grafi

- Rete di amicizia su un social network: ogni utente è un nodo; ogni volta che due utenti diventano amici, si crea un arco del grafo

- Google maps: i nodi rappresentano città, intersezioni di strade, siti di interesse, ecc. e gli archi rappresentano le connessioni dirette tra i nodi. La rappresentazione mediante un grafo permette di trovare il percorso più corto per andare da un posto all'altro mediante un algoritmo.

- World Wide Web: le pagine web sono i nodi e il link tra due pagine è un arco. Google
- utilizza questa rappresentazione per esplorare il World Wide Web.

Terminologia

- Consideriamo due nodi di un grafo G connessi dall'arco e = (u,v)

- Si dice che
  - u e v sono adiacenti
  - l'arco (u,v) incide sui vertici u e v
  - u è un nodo vicino di v
  - v è un nodo vicino di u

- Dato un vertice u di un grafo G
  - Grado di u = numero archi incidenti su u
    - è indicato con deg(u)

Numero di archi di un grafo G

- m = numero di archi di G; n= numero di nodi di G

1. La somma di tutti i gradi dei nodi di G è 2m

$$\sum_{u \in V} deg(u) = 2m$$

Dim. Ciascun arco incide su due vertice e quindi viene contato due volte nella sommatoria in alto.

- L'arco (u,v) è contato sia in deg(u) che in deg(v)

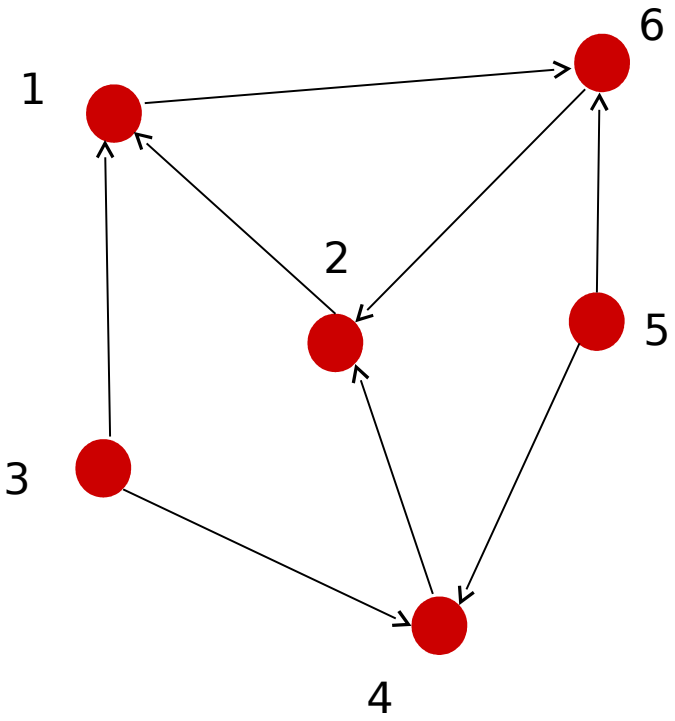
- Il numero m di archi di un grafo G **non direzionato** è **al più** n(n-1)/2

Dim. Il numero di coppie non ordinate distinte che si possono formare con n nodi è n(n-1)/2

- Posso scegliere il primo nodo dell'arco in n nodi e il secondo in modo che sia diverso dal primo nodo, cioè n-1 modi. Dimezzo in quanto l'arco (u,v) è uguale all'arco (v,u)

Grafi direzionati

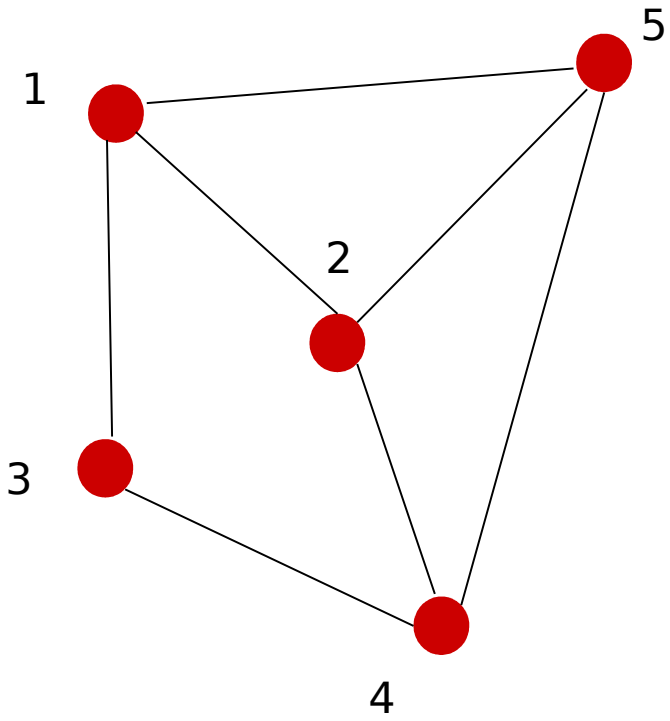
- Gli archi hanno una direzione
  - L'arco (u,v) è diverso dall'arco (v,u)
  - Si dice che l'arco e=(u,v) lascia u ed entra in v
  - e che u è l'origine dell'arco e v la destinazione dell'arco

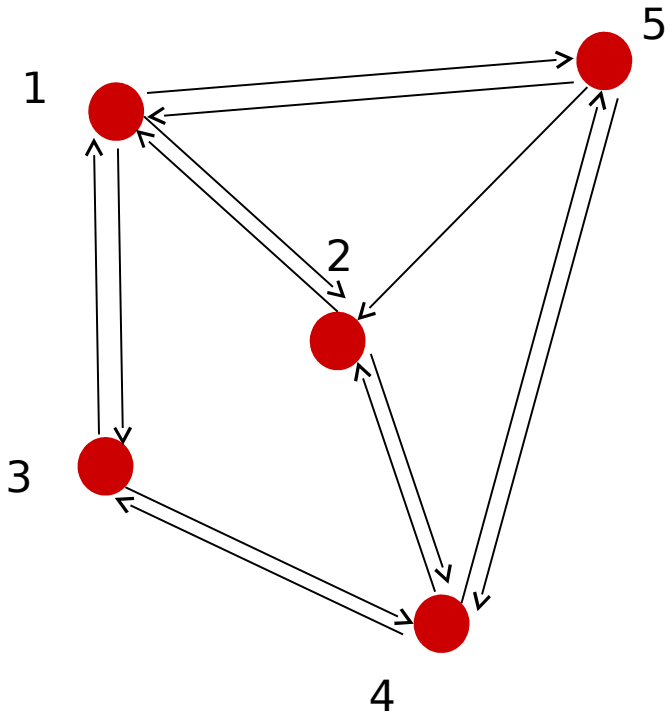


V = { 1, 2, 3, 4, 5, 6 }
E = {1-6, 2-1, 3-1, 3-4, 4-2, 5-4, 5-6, 6-2 }
n = 6
m = 8

Grafi direzionati

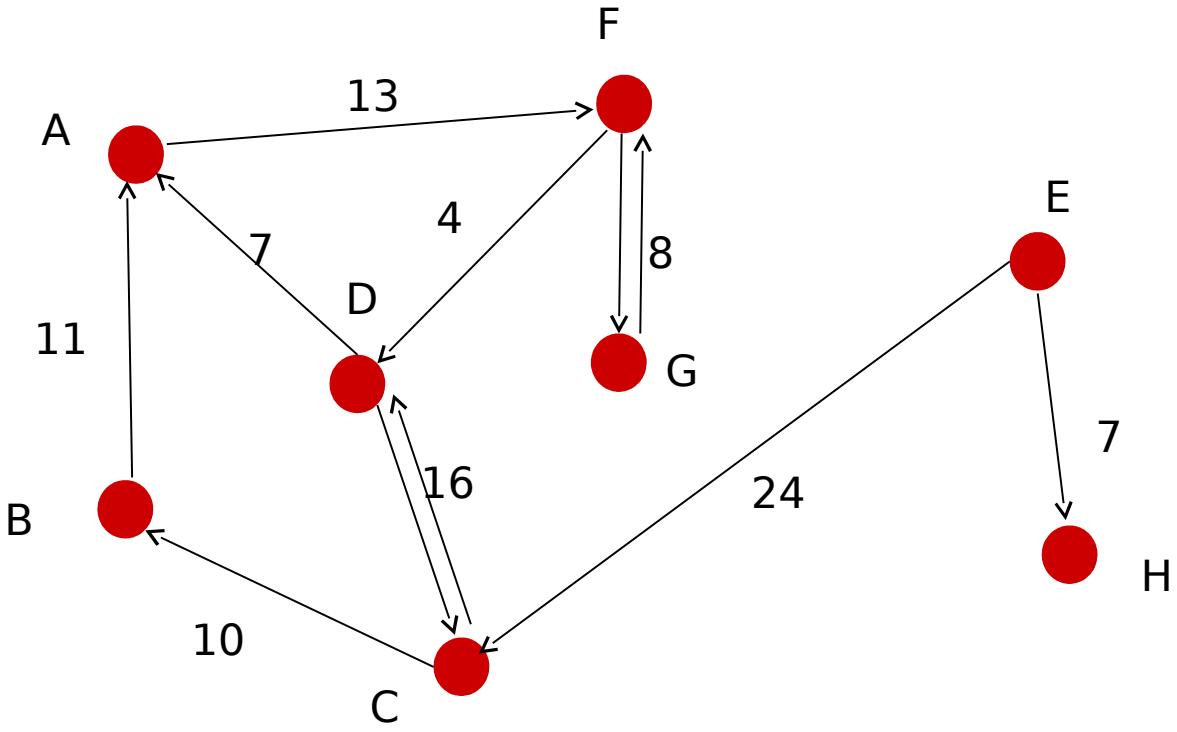
- Grafi non direzionati G = (V, E) possono essere visti come un caso particolare degli archi direzionati in cui per ogni arco (u,v) c'è l'arco di direzione opposta (v,u)





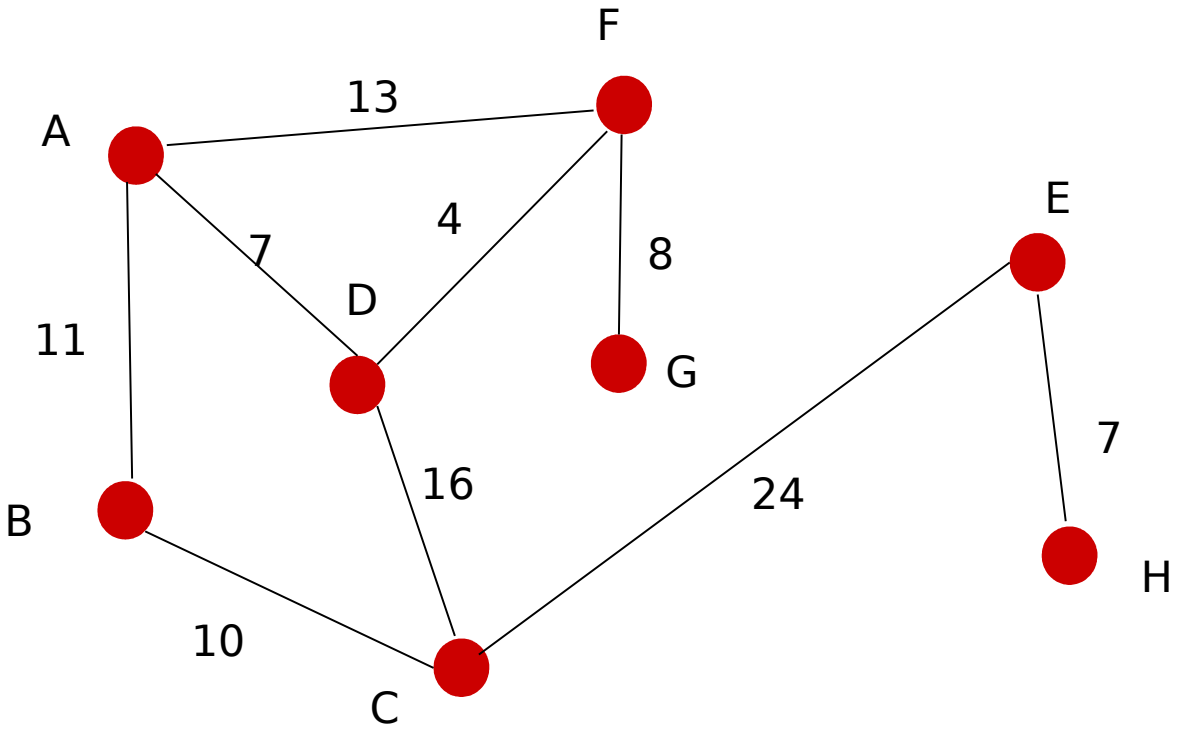
Esempio di applicazione

- Archi: strade (a senso unico di circolazione)
- Nodi: intersezioni tra strade
- Pesi archi: lunghezza in km



Esempio di applicazione

- Archi: strade (a doppio senso di circolazione)
- Nodi: intersezioni tra strade
- Pesi archi: lunghezza in km



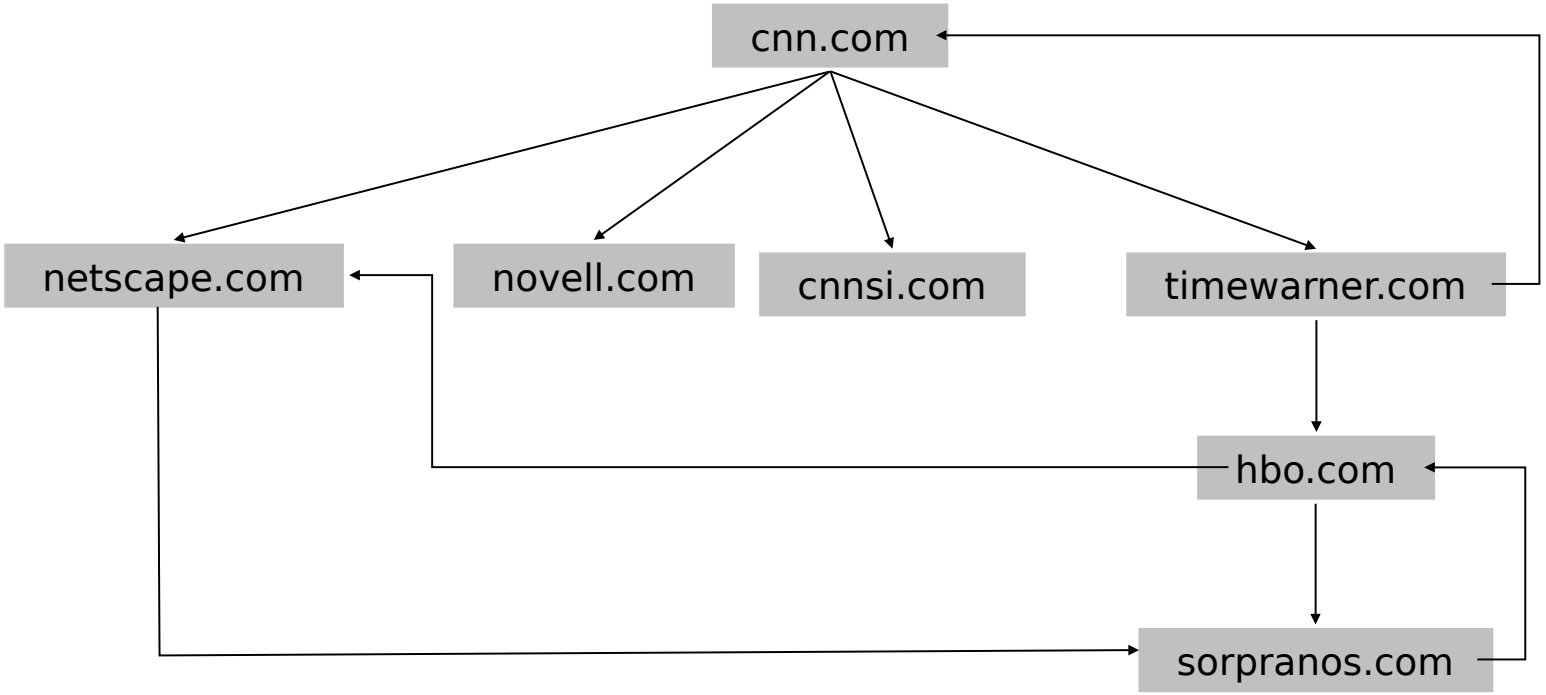
Numero di archi di un grafo G direzionato

- m = numero di archi di G; n= numero di nodi di G

- Il numero m di archi di G è **al più** n^2
- **Dim.** Il numero di coppie ordinate distinte che si possono formare con n nodi è n^2
    - Posso scegliere il primo nodo dell'arco in n nodi e il secondo in altri n modi (se ammettiamo archi con entrambe le estremità uguali).

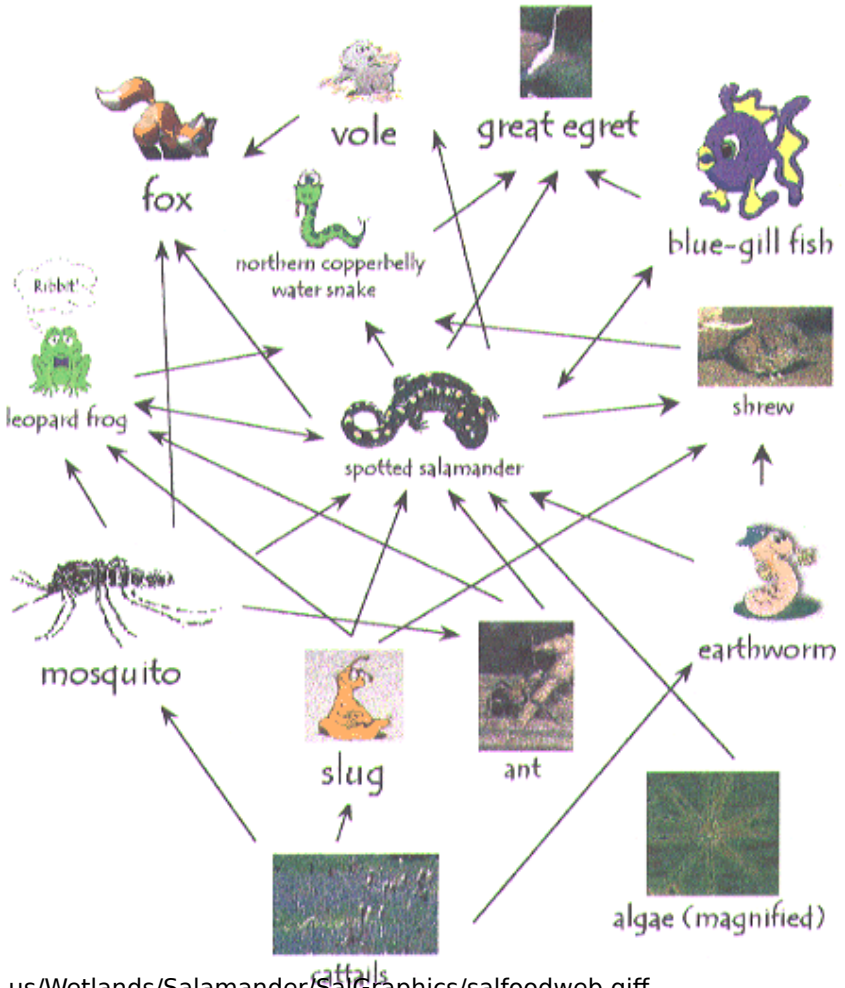
World Wide Web

- Web graph : grafo direzionato.
  - Nodo: pagina web.
  - Edge: hyperlink da una pagina all'altra.



Ecological Food Web

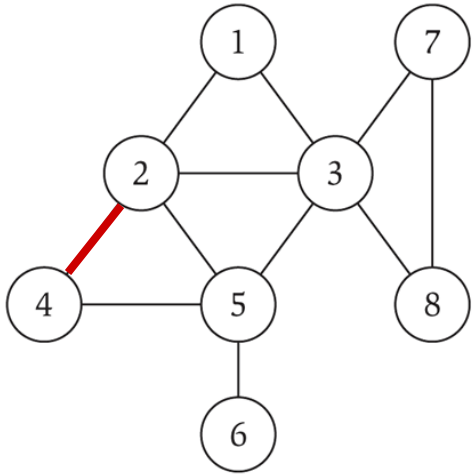
- Food web graph: grafo direzionato.
  - Nodo = specie.
  - Arco = dalla preda al predat



Reference: http://www.twingroves.dètrict96.k12.il.us/Wetlands/Salamander/SalGraphics/salfoodweb.giff

Rappresentazione di grafi: Matrice di Adiacenza

- **Matrice di adiacenza.** Matrice nxn con $A_{uv} = 1$ se (u, v) è an arco.
  - Due rappresentazioni di ciascun arco.
  - Spazio proporzionale ad n^2.
  - Controllare se (u, v) è un arco richiede tempo $\Theta(1)$.
  - Identificare gli archi incidenti su un nodo u richiede $\Theta(n)$
  - Identificare tutti gli archi richiede tempo $\Theta(n^2)$.

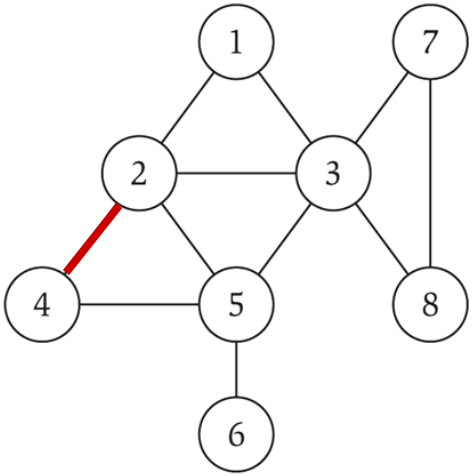


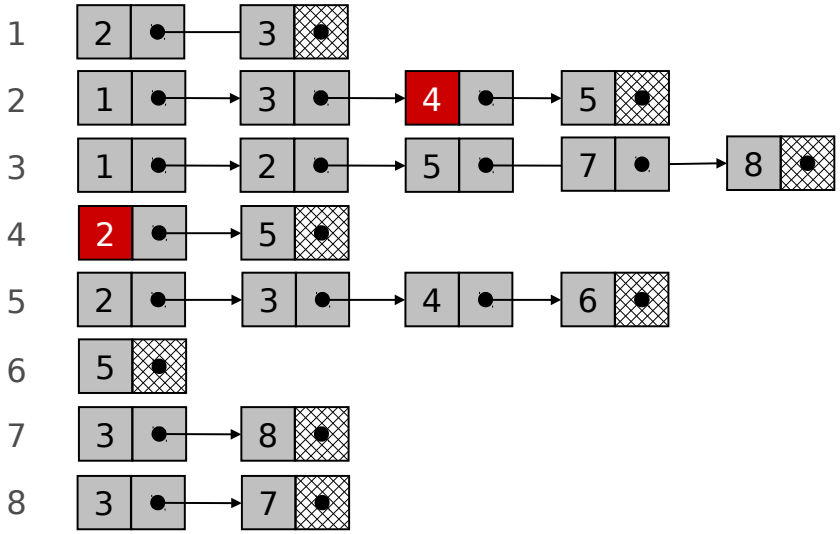
	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	0	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

Rappresentazione di un grafo: liste di adiacenza

- Liste di adiacenza. Array di liste in cui ogni lista è associata ad un nodo.
  - Ad ogni arco corrisponde un elemento della lista.
  - Spazio proporzionale a m + n.
  - Controllare se (u, v) è un arco richiede tempo O(deg(u)).
  - Individuare tutti gli archi richiede tempo $\Theta(m + n)$.

Degree (grado): numero di vicini di u



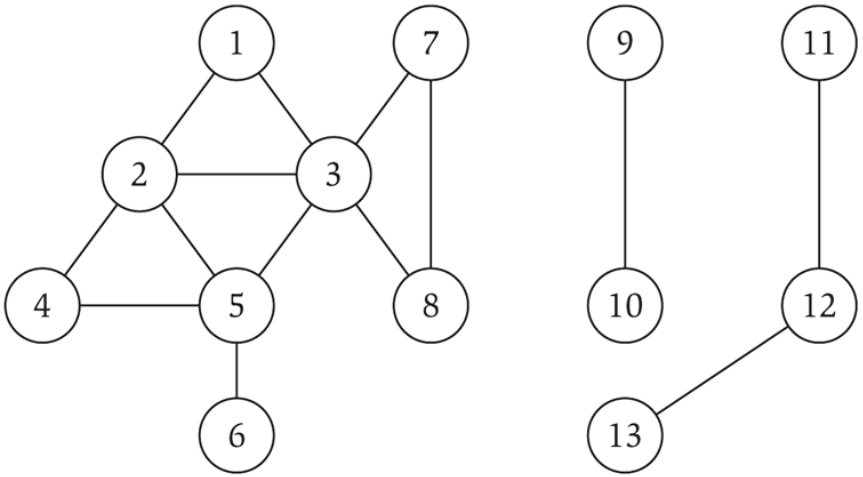


Percorsi e connettività

- Def. Un percorso in un grafo non direzionato G = (V, E) è una sequenza P di nodi v_1, v_2, ..., v_{k-1}, v_k con la proprietà che ciascuna coppia di vertici consecutivi v_i, v_{i+1} è unita da un arco in E.

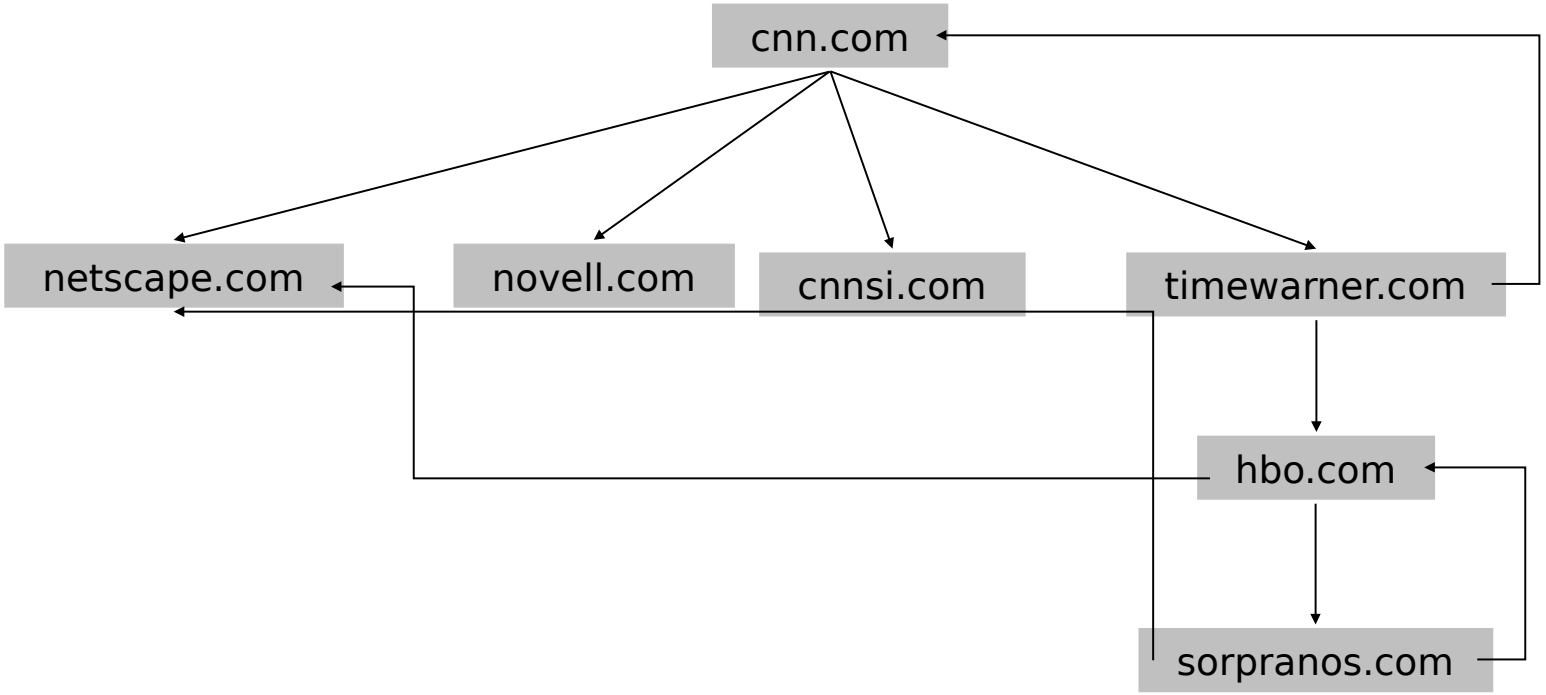
- Def. Un percorso è semplice se tutti i nodi sono distinti.

- Def. Un grafo non direzionato è connesso se per ogni coppia di nodi u e v, esiste un percorso tra u e v.



Applicazione del concetto di percorso

- Esempi:
  - Web graph. Voglio capire se è possibile, partendo da una pagina web è seguendo gli hyperlink nelle pagine via via attraversate, arrivare ad una determinata pagina



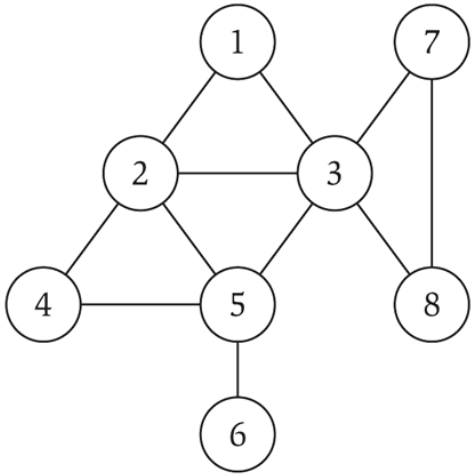
Applicazione del concetto di percorso

- In alcuni casi può essere interessante scoprire il percorso più corto tra due nodi

- Esempio:
  - Grafo : rete di trasporti dove i nodi sono gli aeroporti e gli archi i collegamenti diretti tra aeroporti.
  - Voglio arrivare da Napoli a New York facendo il minimo numero di scali.

Cicli

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



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

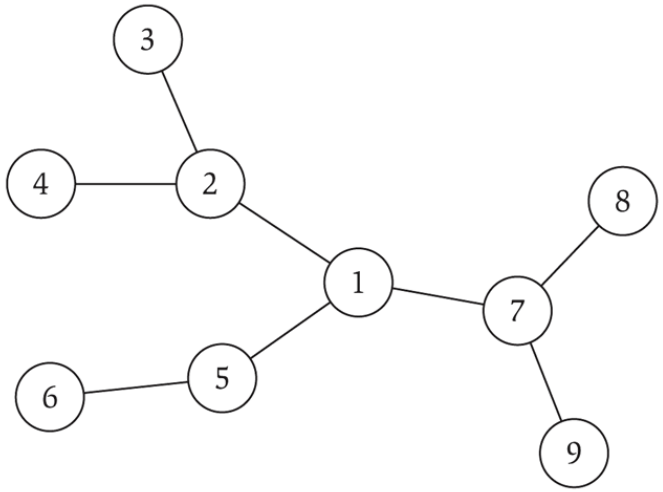
Alberi

- Def. Un grafo non direzionato è un albero (tree) se è connesso e non contiene cicli

- Teorema. Sia G un grafo non direzionato con n nodi. Ogni due delle seguenti affermazioni implicano la restante affermazione.
  - 1 e 2 ⟹ 3 ; 1 e 3 ⟹ 2 ; 2 e 3 ⟹ 1

1. G è connesso.
2. G non contiene clicli.
3. G ha n-1 archi.

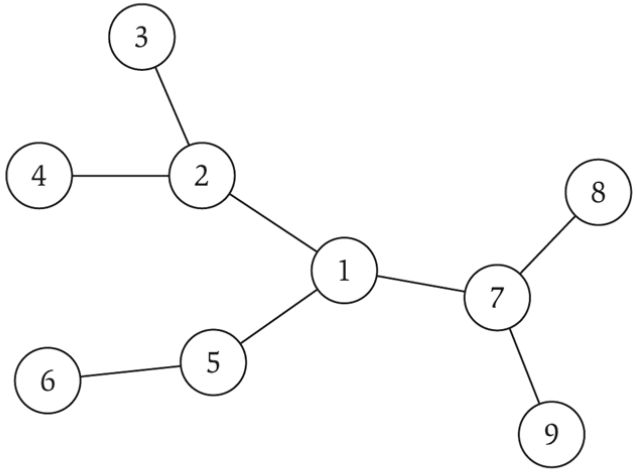
} albero



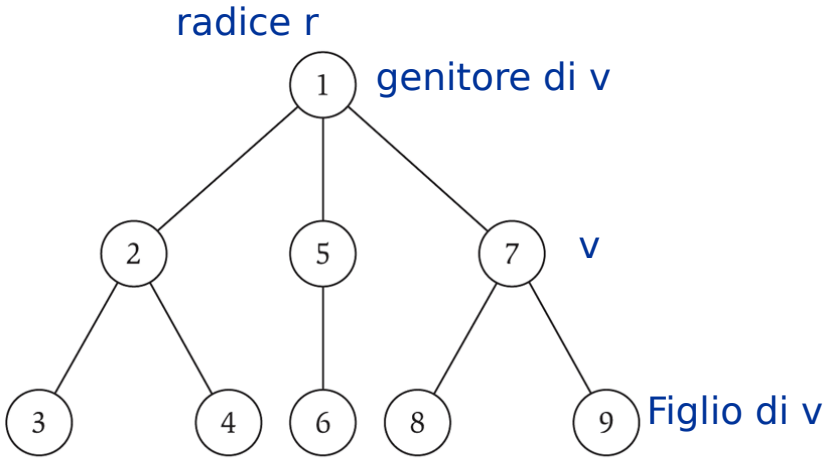
Alberi con radice

Albero con radice. Dato un albero T, si sceglie un nodo radice r e si considerano gli archi di T come orientati a partire da r

Dato un nodo v di T si dice
Genitore di v: il nodo che precede v lungo il percorso da r a v
Antenato: un qualsiasi nodo w lungo il percorso che va da r a v (v viene detto discendente di w)
Foglia: nodo senza discendenti



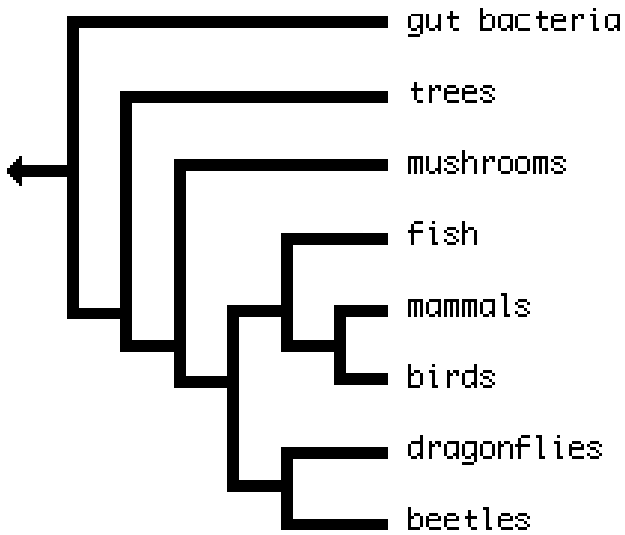
Un albero



Lo stesso albero con radice 1

Importanza degli alberi: rappresentano strutture gerarchiche

- Alberi filogenetici. Descrivono la storia evolutiva delle specie animali.

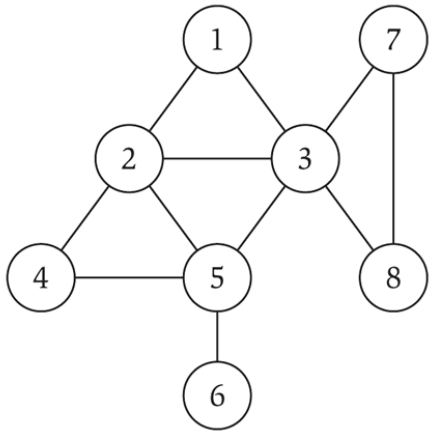


La filogenesi afferma l'esistenza di una specie ancestrale che diede origine a mammiferi e uccelli ma non alle altre specie rappresentate nell'albero (cioè, mammiferi e uccelli condividono un antenato che non è comune ad altre specie nell'albero). La filogenesi afferma inoltre che tutti gli animali discendono da un antenato non condiviso con i funghi, gli alberi e i batteri, e così via.

Problemi su grafi

- Problema della connettività tra s e t. Dati due nodi s e t, esiste un percorso tra s e t?

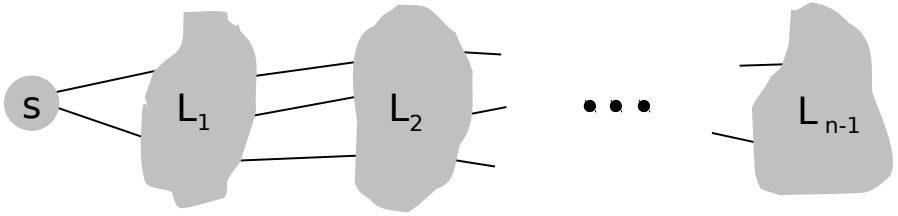
- Problema del percorso più corto tra s e t. Dati due nodi s e t, qual è la lunghezza del percorso più corto tra s e t?



- Applicazioni.
  - Attraversamento di un labirinto.
  - Erdős number.
  - Minimo numero di dispositivi che devono essere attraversati dai dati in una rete di comunicazione per andare dalla sorgente alla destinazione

Breadth First Search (visita in ampiezza)

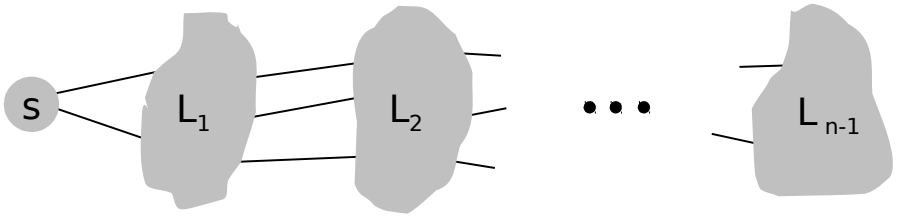
- BFS. Explora il grafo a partire da una sorgente s muovendosi in tutte le possibile direzioni e visitando i nodi livello per livello (N.B.: il libro li chiama layer e cioè strati).



- Algoritmo BFS.
  - L_0 = { s }.
  - L_1 = tutti i vicini di s.
  - L_2 = tutti i nodi che non appartengono a L_0 or L_1, e che sono uniti da un arco ad un nodo in L_1.
  - L_{i+1} = tutti i nodi che non appartengono agli strati precedenti e che sono uniti da un arco ad un nodo in L_i.

Breadth First Search

Teorema. Per ogni i, L_i consiste di tutti i nodi a distanza i da s. C'è un percorso da s a t se e solo t appare in qualche livello.



L_1: livello dei nodi a distanza 1 da s

L_2: livello dei nodi a distanza 2 da s

...

L_{n-1}: livello dei nodi a distanza n-1 da s

Breadth First Search

Pseudocodice

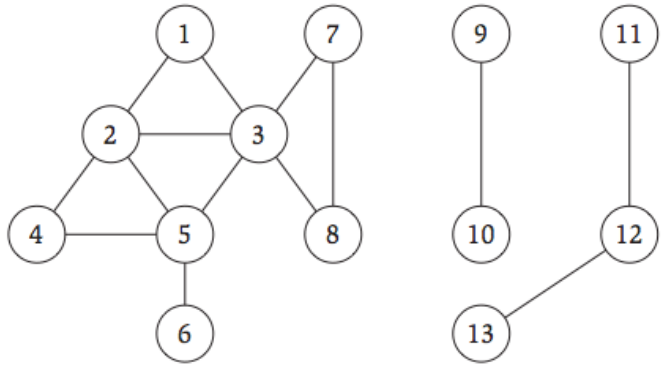
1. BFS(s)
2. L_0={ s }
3 **For**(i=0;i≤n-1;i++)
4. L_{i+1}= ∅ ;
5. **For each** nodo u in L_i
6. **For each** nodo v adiacente ad u
7. **if(** v non appartiene ad L_0,…,L_i)
8. $L_{i+1}=L_{i+1} \cup$ { v }
10. **Endfor**
11. **Endfor**

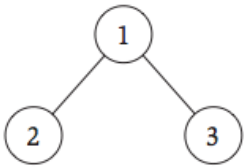
Il for alle linee 6-10 e` iterato $\sum_{u \in V} deg(u)$ volte

Occorre un modo per capire se un nodo è gia stato visitato in precedenza. Il tempo di esecuzione dipende dal modo scelto, da come è implementato il grafo e da come sono rappresentati gli insiemi L_i che rappresentano i livelli

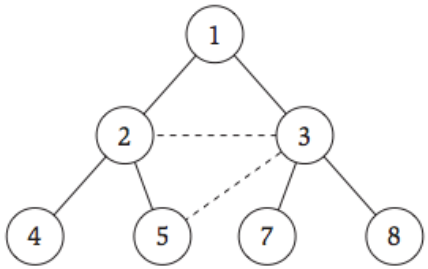
Esempio di esecuzione di BFS

G

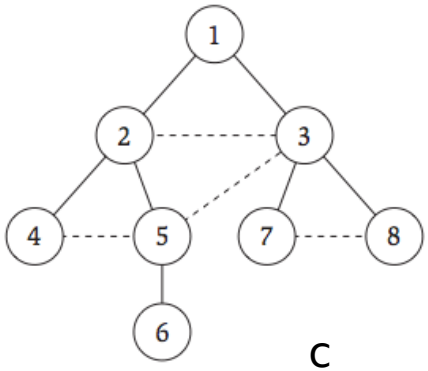




a



b



c

$L_0=\{1\}$

a. $L_1=\{2,3\}$

b. $L_2=\{4,5,7,8\}$

c. $L_3=\{6\}$

Breadth First Search Tree (Albero BFS)

- Proprietà. L'algoritmo BFS produce un albero che ha come radice la sorgente s e come nodi tutti i nodi del grafo raggiungibili da s.

- L'albero si ottiene in questo modo:
- Consideriamo il momento in cui un vertice v viene scoperto, e cioè il momento in cui visitato per la prima volta.
  - Ciò avviene durante l'esame dei vertici adiacenti ad un certo vertice u di un certo livello L_i (linea 6).
  - In questo momento, oltre ad aggiungere v al livello L_{i+1} (linea 8), aggiungiamo l'arco (u,v) e il nodo v all'albero

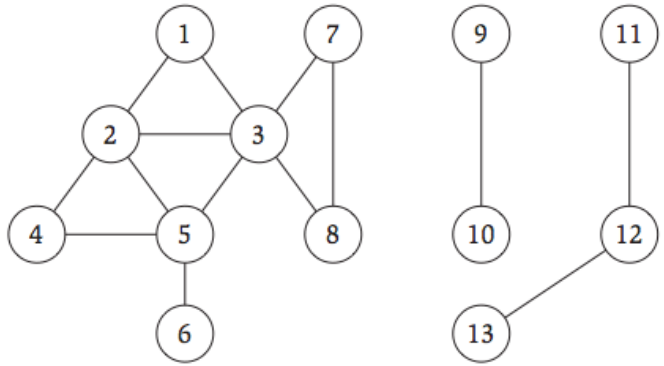
Implementazione di BFS

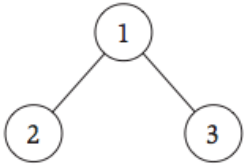
- Ciascun insieme L_i è rappresentato da una lista L[i]
- Usiamo un array di valori booleani Discovered per associare a ciascun nodo il valore vero o falso a seconda che sia già stato scoperto o meno
- Durante l'algoritmo costruiamo anche l'albero BFS

```
1.  BFS(s):
2.  Poni Discovered[s] = true e Discovered[v] = false per tutti gli altri v
3.  Inizializza L[0] in modo che contenga solo s
4.  Poni il contatore dei livelli i = 0
5.  Inizializza il  BFS tree T con un albero vuoto
6.  While L[i] non è vuota                          //L[i] è vuota se non ci sono
7.      Inizializza L[i+1] con una lista vuota   //nodi raggiungibili da L[i-1]
8.      For each u ∈ L[i]
9.      Considera ogni arco (u, v) incidente su u
10.       If Discovered[v] = false then
11.           Poni Discovered[v] = true
12.           Inserisci v in L[i+1]
13.           Aggiungi l'arco (u, v) all'albero T
14.     Endif
15.   Endfor
16.  i=i+1
17. Endwhile
```

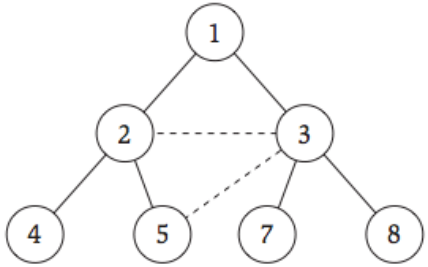
Esempio di esecuzione di BFS

G

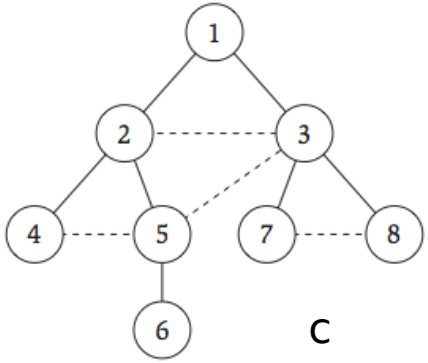




a



b



c

$L_0=\{1\}$

a. $L_1=\{2,3\}$ aggiunge gli archi (1,2) e (1,3) all'albero

b. $L_2=\{4,5,7,8\}$ aggiunge gli archi (2,4), (2,5), (3,7), (3,8) all'albero

c. $L_3=\{6\}$ aggiunge l'arco (5,6) all'albero

Analisi di BFS nel caso in cui il grafo è rappresentato con liste di adiacenza

1. BFS(s):
2. Poni Discovered[*s*] = true e Discovered[*v*] = false per tutti gli altri *v*
3. Inizializza *L*[0] in modo che contenga solo *s*
4. Poni il contatore dei livelli *i* = 0
5. Inizializza il BFS tree *T* con un albero vuoto
6. **While** *L*[*i*] non è vuota
7. Inizializza *L*[*i*+1] con una lista vuota
8. **For each** *u* ∈ *L*[*i*]
9. Considera ogni arco (*u*, *v*) incidente su *u*
10. **If** Discovered[*v*] = false **then**
11. Poni Discovered[*v*] = true
12. Inserisci v in L[i+1]
13. Aggiungi l'arco (*u*, *v*) all'albero *T*
14. *Endif*
15. *Endfor*
16. *i=i+1*
17. *Endwhile*

O(n)

O(1)

Al più n-1 volte in quanto al più n-1 livelli

For each in totale viene eseguito al più n volte in quanto ogni nodo appartiene ad una sola lista L_i

Linee 9-14 in totale vengono eseguite un numero di volte =

$$\sum_{i=0}^{n} \sum_{u \in L[i]} deg(u) \leq \sum_{u \in V} deg(u) \leq 2m$$

Algoritmo è O(n+m)

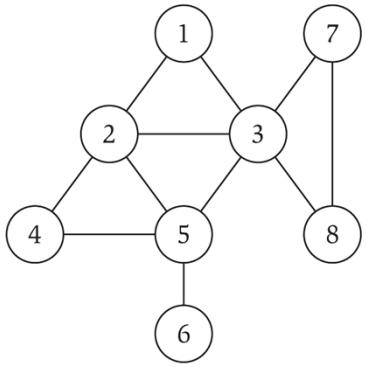
Breadth First Search Tree

- Proprietà. Sia T un albero BFS di G = (V, E), e sia (x, y) un arco di G. I livelli di x e y differiscono di al più di 1.

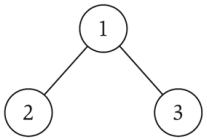
- Dim. Sia L_i il livello di x ed L_j quello di y. Supponiamo senza perdere di generalità che x venga scoperto prima di y cioè che **$i \leq j$**. Consideriamo il momento in cui l'algoritmo esamina gli archi incidenti su x.

- Caso 1. Il nodo y è stato già scoperto:
  Siccome per ipotesi y viene scoperto dopo x allora sicuramente y viene inserito o nel livello i dopo x (se adiacente a qualche nodo nel livello i-1) o nel livello i+1 (se adiacente a qualche nodo del livello i esaminato nel **For each** prima di x). Quindi in questo caso j= i o j=i+1.

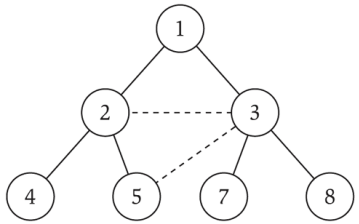
- Caso 2. Il nodo y non è stato ancora scoperto:
  Siccome tra gli archi incidenti su x c'è anche (x,y) allora y viene inserito in questo momento in L_{i+1} . Quindi in questo caso j=i+1.



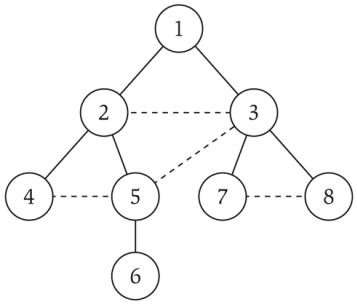
G



(a)



(b)



(c)

Implementazione di BFS con coda FIFO

- L'algoritmo BFS si presta ad essere implementate con un coda
- Ogni volta che viene scoperto un nodo u, il nodo u viene inserito nella coda
- Vengono esaminati gli archi incidenti sul nodo al front della coda

BFS(s)

1. Inizializza Q con una coda vuota
2. Poni Discovered[*s*] = true e Discovered[*v*] = false per tutti gli altri *v*
3. Inserisci s in coda a Q con una enqueue
4. While(Q non è vuota)
5. estrai il front di Q con una deque e ponilo in u
6. For each arco (u,v) incidente su u
7. If(Discovered[v]=false)
8. Poni Discovered[v]= true
9. aggiungi v in coda a Q con una enqueue
10. aggiungi (u,v) al BFS tree
11. Endif
12. Endwhile
13. Endwhile

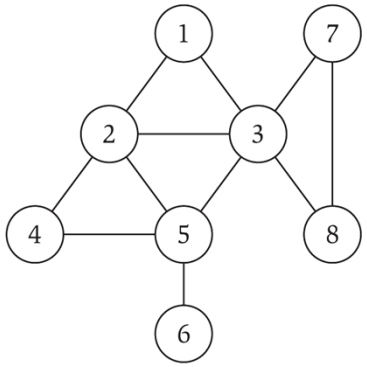
Breadth First Search Tree

- Proprietà. Sia T un albero BFS di G = (V, E), e sia (x, y) un arco di G. I livelli di x e y differiscono di al più di 1.

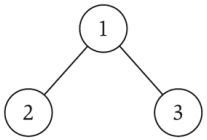
- Dim. Sia L_i il livello di x ed L_j quello di y. Supponiamo senza perdere di generalità che x venga scoperto prima di y cioè che **$i \leq j$**. Consideriamo il momento in cui l'algoritmo esamina gli archi incidenti su x.

- Caso 1. Il nodo y è stato già scoperto:
  Siccome per ipotesi y viene scoperto dopo x allora sicuramente y viene inserito o nel livello i dopo x (se adiacente a qualche nodo nel livello i-1) o nel livello i+1 (se adiacente a qualche nodo del livello i esaminato nel **For each** prima di x). Quindi in questo caso j= i o j=i+1.

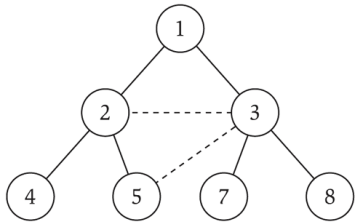
- Caso 2. Il nodo y non è stato ancora scoperto:
  Siccome tra gli archi incidenti su x c'è anche (x,y) allora y viene inserito in questo momento in L_{i+1} . Quindi in questo caso j=i+1.



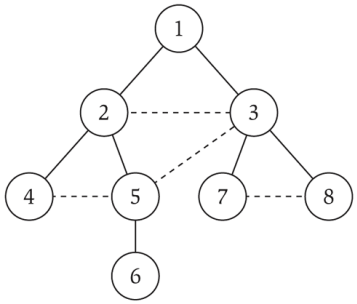
G



(a)



(b)



(c)

Depth first search (visita in profondità)

- La visita in profondità riproduce il comportamento di una persona che esplora un labirinto di camere interconnesse
  - La persona parte dalla prima camera (nodo s) e si sposta in una delle camere accessibili dalla prima (nodo adiacente ad s), di lì si sposta in una delle camere accessibili dalla seconda camera visitate e così via fino a che raggiunge una camera da cui non è possibile accedere a nessuna altra camera non ancora vistata. A questo punto torna nella camera precedentemente visitata e di lì prova a raggiungere nuove camere.

Depth first search (visita in profondità)

- La visita DFS parte dalla sorgente s e si spinge in profondità fino a che non è più possibile raggiungere nuovi nodi.
  - La visita parte da s, segue uno degli archi uscenti da s ed esplora il vertice v a cui porta l'arco.
  - Una volta in v, se c'è un arco uscente da v che porta in un vertice w non ancora esplorato allora l'algoritmo esplora w
  - Uno volta in w segue uno degli archi uscenti da w e così via fino a che non arriva in un nodo del quale sono già stati esplorati tutti i vicini.
  - A questo punto l'algoritmo fa backtrack (torna indietro) fino a che torna in un vertice a partire dal quale può visitare un vertice non ancora esplorato in precedenza.

Depth first search: pseudocodice

```
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
```

R = insieme dei vertici raggiunti

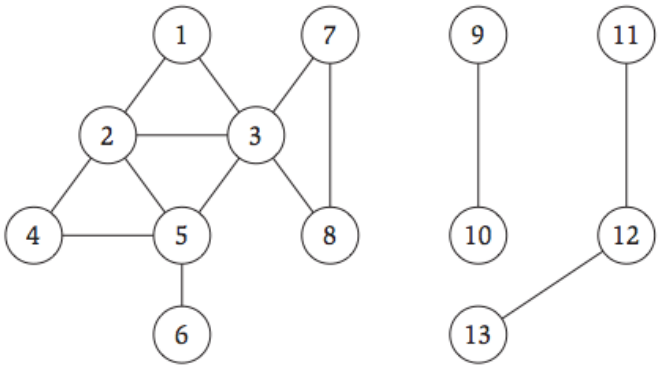
Analisi:

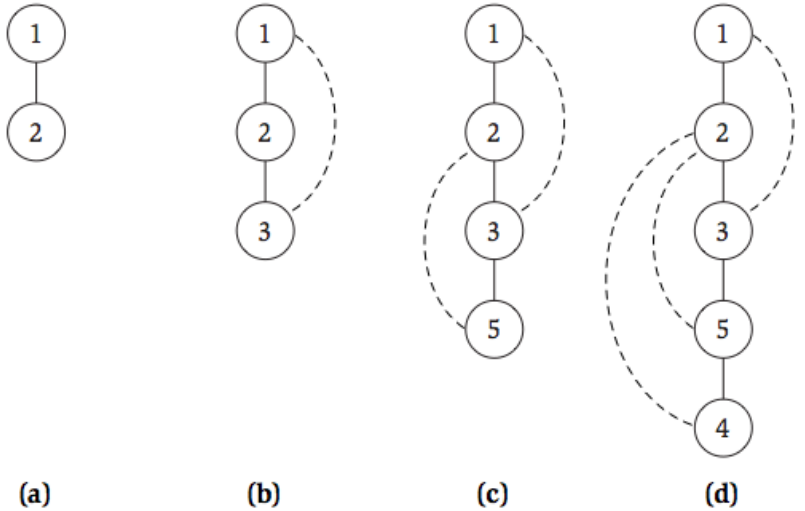
Ciascuna visita ricorsiva richiede tempo O(1) per marcare u e aggiungerlo ad R e tempo O(deg(u)) per eseguire il for.

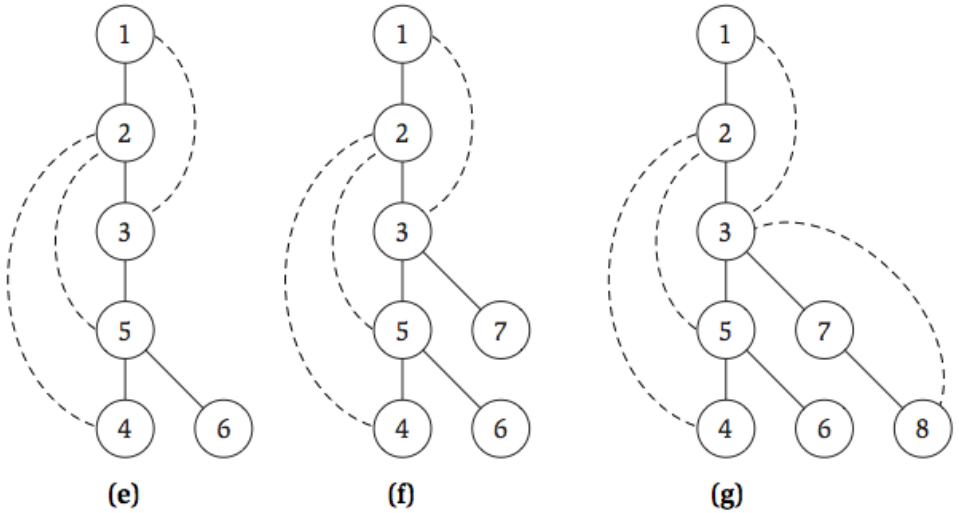
Se inizialmente invochiamo DFS su un nodo s, allora DFS viene invocata ricorsivante su tutti i nodi raggiungibili a partire da s. Il costo totale è quindi $\sum_{u \in V} O(1) + \sum_{u \in V} O(deg(u)) = O(n) + O(m) = O(n+m)$

Esempio di esecuzione di DFS

G







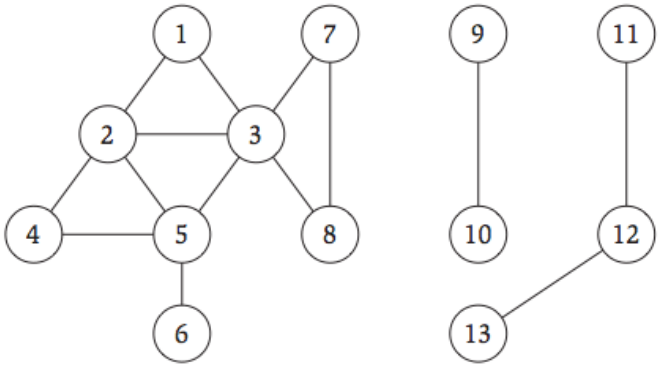
Depth First Search Tree (Albero DFS)

- Proprietà. L'algoritmo DFS produce un albero che ha come radice la sorgente s e come nodi tutti i nodi del grafo raggiungibili da s.

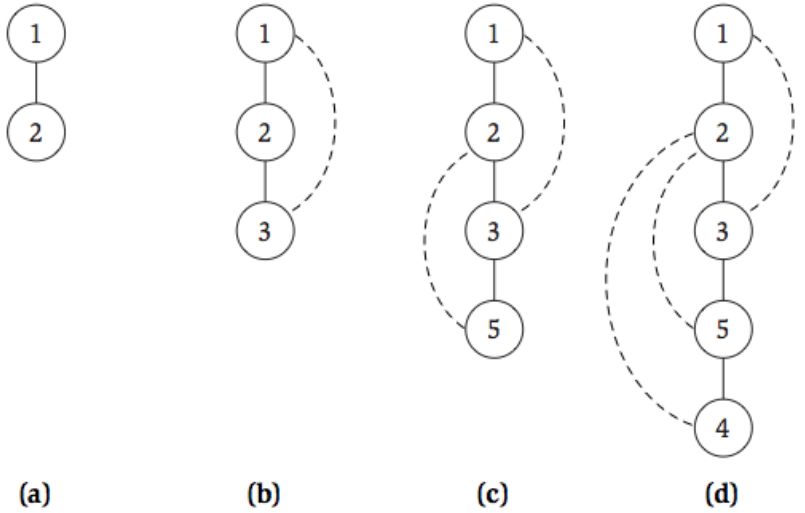
- L'albero si ottiene in questo modo:
  - Consideriamo il momento in cui viene invocata DFS(v)
    - Ciò avviene durante l'esecuzione di DFS(u) per un certo nodo u. In particolare durante l'esame dell'arco (u,v) nella chiamata DFS(u).
    - In questo momento, aggiungiamo l'arco (u,v) e il nodo v all'albero

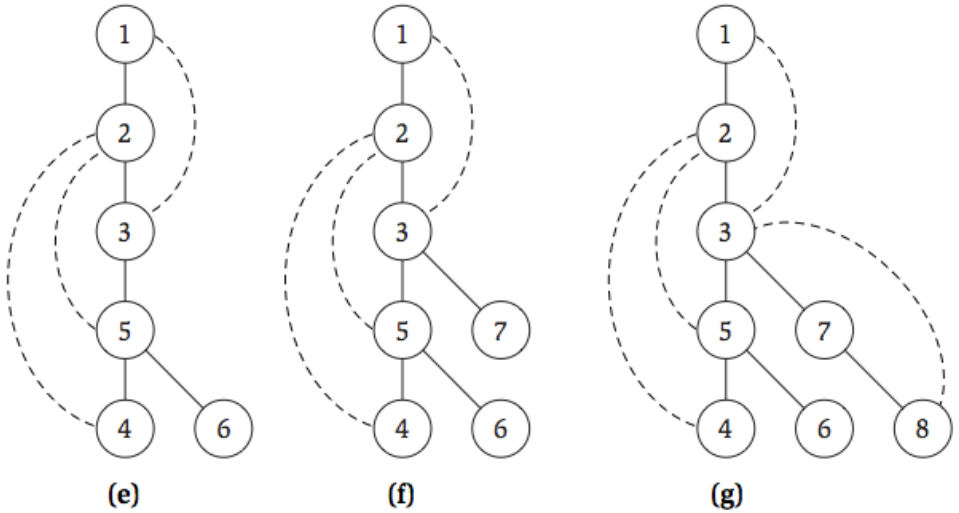
Esempio di albero DFS

G



Gli archi non tratteggiati fanno parte del DFS tree.





Albero DFS

- Proprietà 1. Per una data chiamata ricorsiva DFS(u), tutti i nodi che vengono etichettati come “Esplorati” tra l’inizio e la fine della chiamata DFS(u), sono discendenti di u nell’albero DFS.

- Proprietà 2. Sia T un albero DFS e siano x e y due nodi di T collegati dall’arco (x,y) in G. Se (x,y) non è un arco di T allora si ha che x e y sono l’uno antenato dell’altro in T.

  Dim. Supponiamo senza perdere di generalità che DFS(x) venga invocata prima di DFS(y). Ciò vuol dire che quando viene invocata DFS(x), y non è ancora etichettato come “Esplorato”.
  La chiamata DFS(x) esamina l’arco (x,y) e per ipotesi non inserisce (x,y) in T. Ciò si verifica solo se y è già stato etichettato come “Esplorato”. Siccome y non era etichettato come “Esplorato” all’inizio di DFS(x) vuol dire è stato esplorato tra l’inizio e la fine della chiamata DFS(x).
  La proprietà precedente implica che y è discendente di x.

- Proprietà 3. Sia T un albero DFS e siano x e y due nodi di T collegati dall’arco (x,y) in G. Si ha che x e y sono l’uno antenato dell’altro in T. (conseguenza immediata della proprietà)

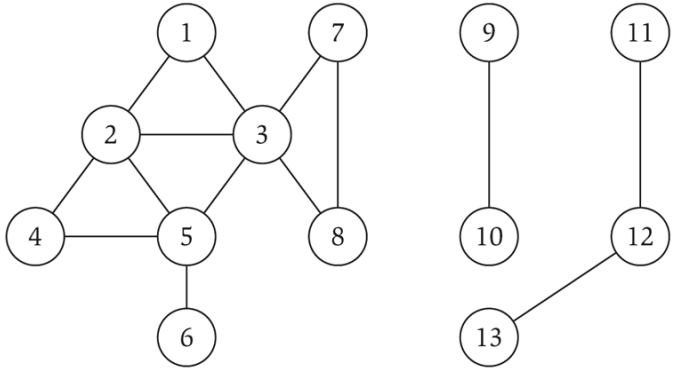
Implementazione di DFS mediante uno stack

```
DFS(s):
  Initialize S to be a stack with one element s
  While S is not empty
    Take a node u from S
    If Explored[u] = false then
       Set Explored[u] = true
       For each edge (u, v) incident to u
         Add v to the stack S
       Endfor
    Endif
  Endwhile
```

Componente connessa

- Componente connessa . Sottoinsieme di vertici tale per ciascuna coppia di vertici u e v esiste un percorso tra u e v

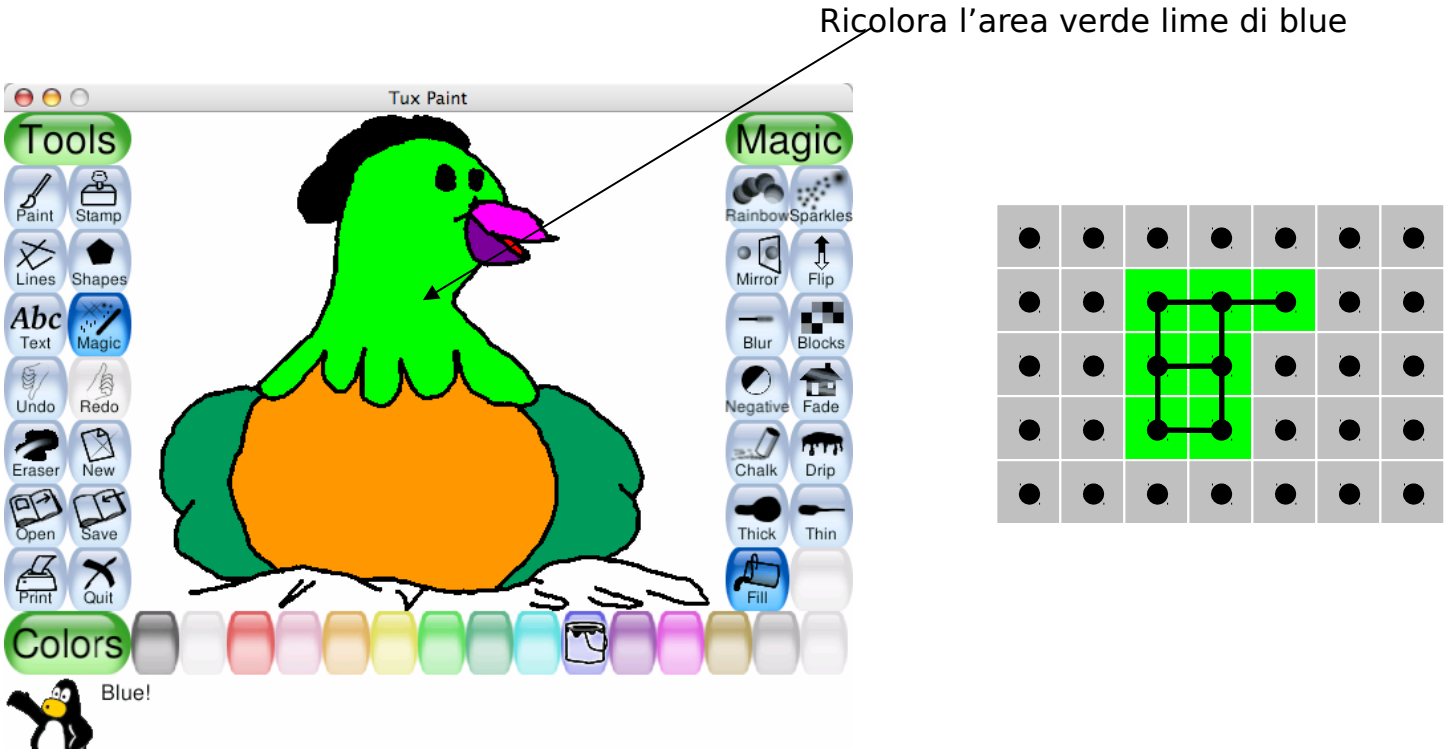
- Componente connessa contenente s. Formata da tutti i nodi raggiungibili da s



- Componente connessa contenente il nodo 1 è { 1, 2, 3, 4, 5, 6, 7, 8 }.

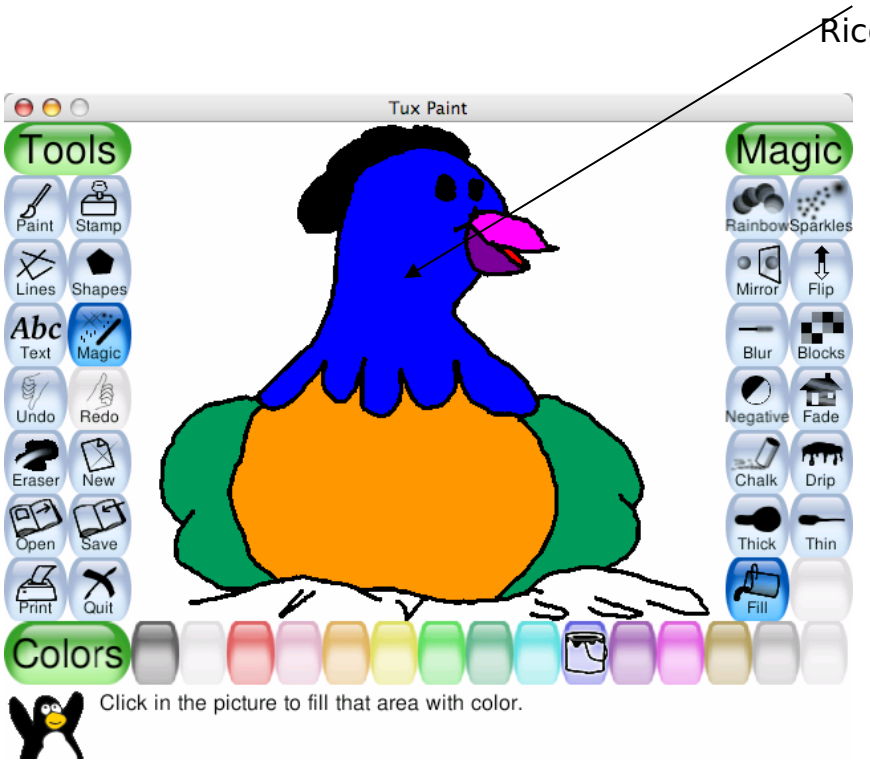
Flood Fill

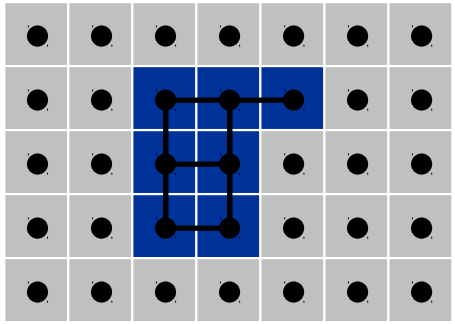
- Flood fill. Data un'immagine, cambia il colore dell'area di pixel vicini di colore verde lime in blu.
  - Nodo: pixel.
  - Arco: due pixel vicini di colore verde lime.
  - Area di pixel vicini: componente connessa di pixel di colore verde lime.



Flood Fill

- Flood fill. Data un'immagine, cambia il colore dell'area di pixel vicini di colore verde lime in blu.
  - Nodo: pixel.
  - Arco: due pixel vicini di colore verde lime.
  - Area di pixel vicini: componente connessa di pixel di colore verde lime.

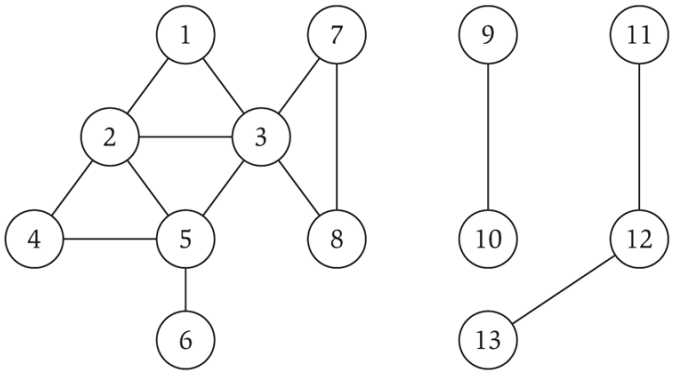




Componente connessa

- Componente connessa contenente s. Trova tutti i nodi raggiungibili da s
  - Come trovarla. Esegui BFS o DFS utilizzando s come sorgente
- Insieme di tutte le componenti connesse. Trova tutte le componenti connesse

Esempio: il grafo sottostante ha tre componenti connesse



Insieme di tutte componenti connesse

- Teorema. Per ogni due nodi s e t di un grafo, le loro componenti connesse o sono uguali o disgiunte
- Dim.
- Caso 1. Esiste un percorso tra s e t. In questo caso ogni nodo u raggiungibile da s è anche raggiungibile da t (basta andare da t ad s e da s ad u) e ogni ogni nodo u raggiungibile da t è anche raggiungibile da s (basta andare da s ad t e da t ad u). Ne consegue che un nodo u è nella componente connessa di s **se e solo** se è anche in quella di t e quindi le componenti connesse di s e t sono uguali.
- Caso 2. Non esiste un percorso tra se e t. In questo caso non può esserci un nodo che appartiene sia alla componente connessa di s che a quella di t. Se esistesse un tale nodo v questo sarebbe raggiungibile sia da s che da t e quindi potremmo andare da s a v e poi da v ad t. Ciò contraddice l'ipotesi che non c'è un percorso tra s e t.

Insieme di tutte componenti connesse

- Il teorema precedente implica che le componenti connesse di un grafo sono a due a due disgiunte.

- Algoritmo per trovare l'insieme di tutte le componenti connesse

```
AllComponents(G)
1. For each node u of G set Discovered[u]= false
2. For each node u of G
3.   If Discovered[u]= false
4.     BFS(u)
5.  Endif
6. Endfor
```

O(n+m)

- Possiamo usare al posto della BFS la DFS e al posto dell'array Discovered l'array Explored

Alcune precisazioni su BFS e DFS

1) Il tempo per eseguire BFS e DFS, escluso il tempo per l'inizializzazione dei campi Discovered di tutti i nodi del grafo, è **O(a+b)** dove a e b sono, rispettivamente, il numero di nodi e il numero di archi della componente connessa di s. Siccome a-1 ≤ b (dal momento che ognuno degli a nodi e connesso a ciascuno degli altri a-1 nodi) allora **O(a+b)=O(b).**
2) Il tempo per inizializzare tutti i campi Discovered è O(n), dove n è il numero di nodi del grafo.

- 1) e 2) implicano che il tempo per eseguire BFS e DFS è O(n+b) e siccome b≤m allora il tempo è O(n+m)

Alcune precisazioni su AllComponents

- L'algoritmo BFS (o DFS) invocato in AllComponents non deve inizializzare i campi Discovered dei nodi di G a false altrimenti l'algoritmo AllComponents invocherebbe BFS (o DFS) anche su nodi già scoperti nelle precedenti invocazioni di BFS (o DFS).
-
- Sarebbe inoltre opportuno usare un array di interi Component che associa a ciascun nodo la componente a cui il nodo appartiene. Ad esempio, Component[u]=j starebbe a significare che u si trova nella componente connessa j. Questo array potrebbe anche essere usato, al posto di Discovered, per capire se un nodo è stato scoperto.

3. Il tempo per eseguire la linea 1 di AllComponent è O(n). Per la 1) nella slide precedente, il tempo per eseguire le linee 2-6 è $\sum_{i=1}^{k} O(n_i+m_i)$, dove k è il numero di componenti connesse ed n_i ed m_i sono rispettivamente il numero di nodi e il numero di archi della componente connessa i. Il teorema visto in precedenza implica che le componenti connesse di un grafo sono a due a due disgiunte e di conseguenza $\sum_{i=1}^{k} n_i+m_i=n+m$ e il tempo di esecuzione di AllComponents è O(n+m).