

# Grafi

Progettazione di Algoritmi a.a. 2017-18

Matricole congrue a 1

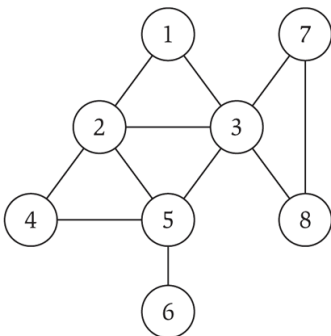
Docente: Annalisa De Bonis

1

## 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:  $n = |V|$ ,  $m = |E|$ .



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

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

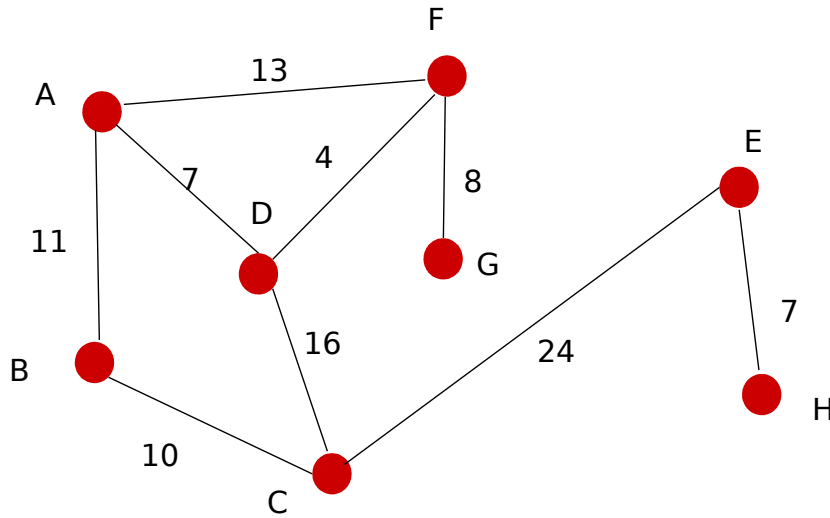
$$n = 8$$

$$m = 11$$

2

## Esempio di applicazione

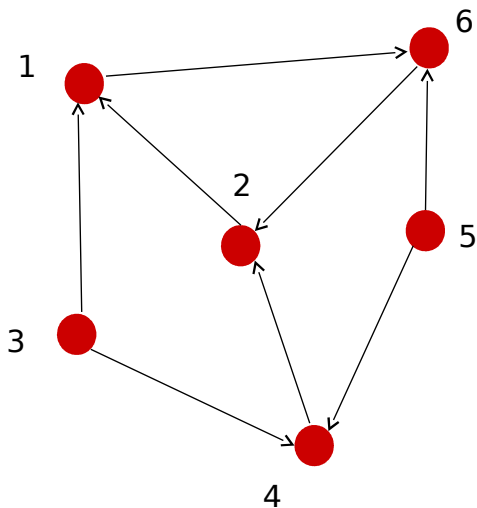
- ✂ Archi: strade (a doppio senso di circolazione)
- ✂ Nodi: intersezioni tra strade
- ✂ Pesi archi: lunghezza in km



3

## 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

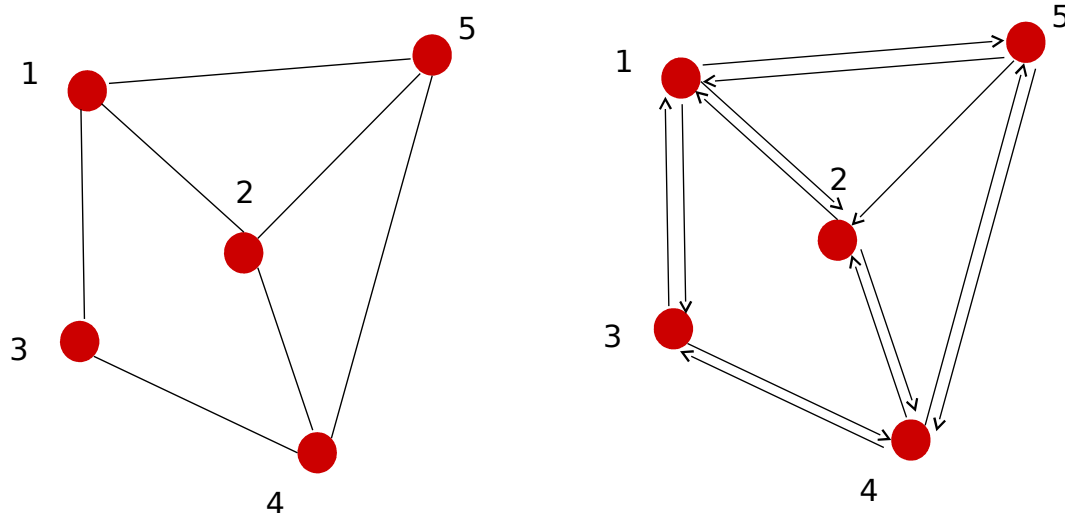


$V = \{ 1, 2, 3, 4, 5, 6 \}$   
 $E = \{ 1-6, 2-1, 3-1, 3-4, 4-2, 5-4, 5-6, 6-2 \}$   
 $n = 6$   
 $m = 8$

4

## 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  $(v,u)$

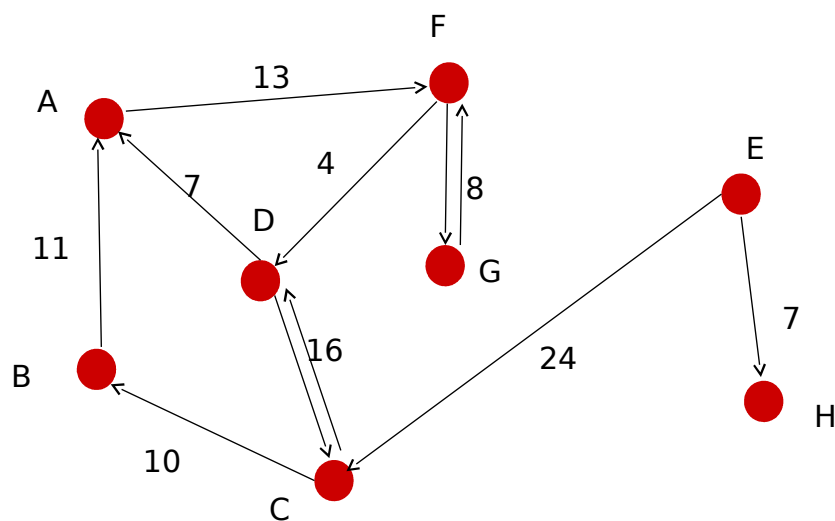


PROGETTAZIONE DI ALGORITMI A.A. 2017-18  
A. DE BONIS

5

## Esempio di applicazione

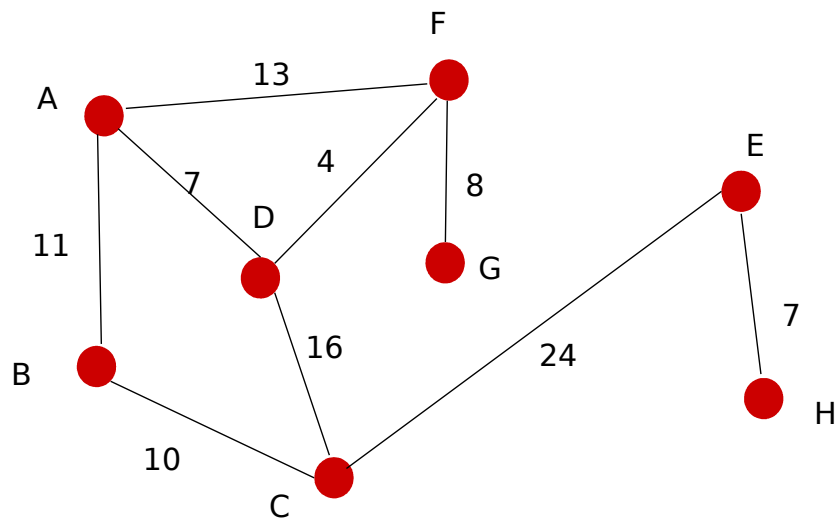
- **Archi:** strade (a senso unico di circolazione)
- **Nodi:** intersezioni tra strade
- **Pesi archi:** lunghezza in km



6

## Esempio di applicazione

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



7

## 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.

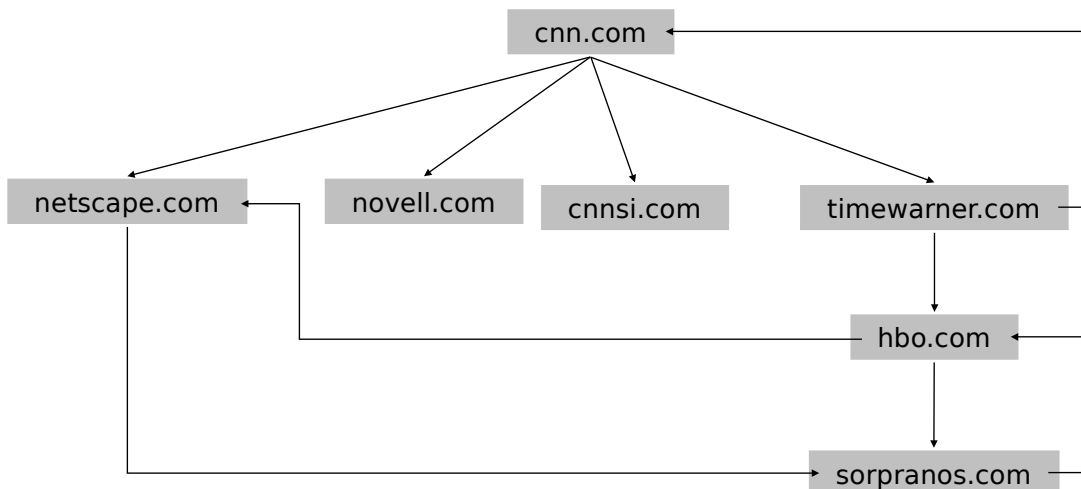
# Alcune applicazioni dei grafi

<i>Grafo</i>	<i>Nodi</i>	<i>Archi</i>
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

## World Wide Web

✂ **Web graph : grafo direzionato.**

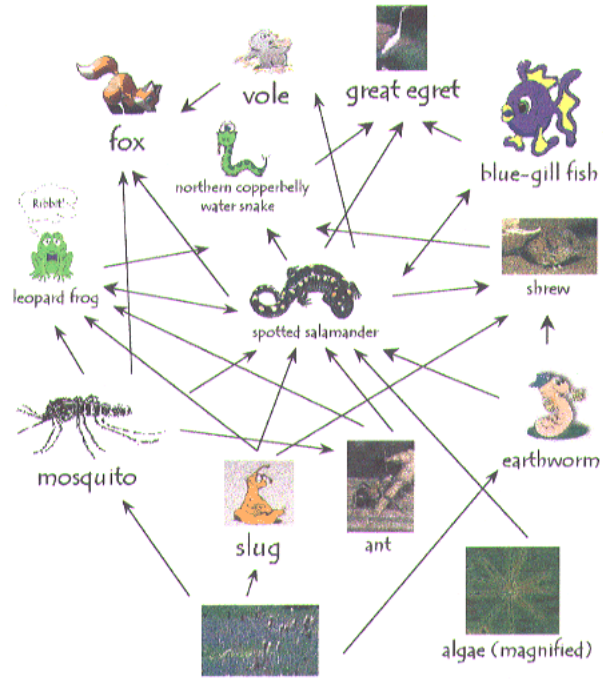
- **Nodo:** pagina web.
- **Edge:** hyperlink da una pagina all'altra.



# Ecological Food Web

## × Food web graph: grafo direzionato.

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



Reference: <http://www.twingroves.district96.k12.il.us/Wetlands/Salamander/SalGraphics/salfoodweb.gif>

PROGETTAZIONE DI ALGORITMI A.A. 2017-18  
A. DE BONIS

11

## 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  $\text{deg}(u)$

12

## 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  $2m$

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

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

- L'arco  $(u,v)$  è contato sia in  $\text{deg}(u)$  che in  $\text{deg}(v)$

2. 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  $(v,u)$

13

## 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  $n^2$

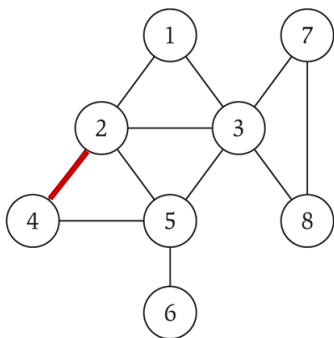
- Posso scegliere il primo nodo dell'arco in  $n$  nodi e il secondo in altri  $n$  modi (se ammettiamo archi con entrambe le estremità uguali).

14

## Rappresentazione di grafi: Matrice di Adiacenza

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

- Due rappresentazioni di ciascun arco.
- Spazio proporzionale a  $n^2$ .
- 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

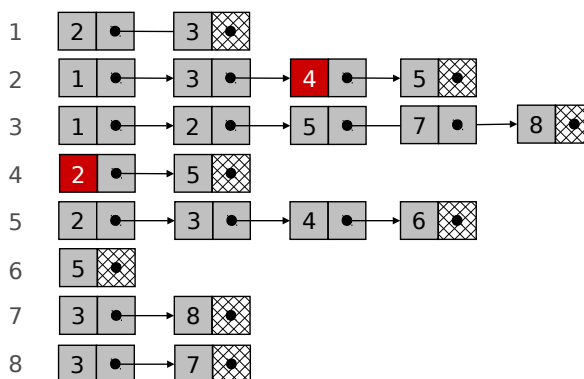
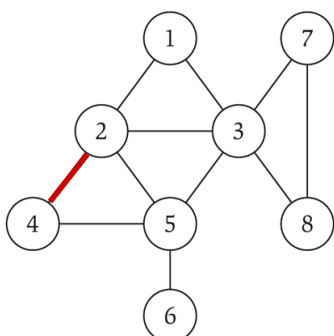
PROGETTAZIONE DI ALGORITMI A.A. 2017-18  
A. DE BONIS

15

## 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  $O(\text{deg}(u))$ . Degree (grado): numero di vicini di  $u$ .
- Individuare tutti gli archi richiede tempo  $\Theta(m + n)$ .



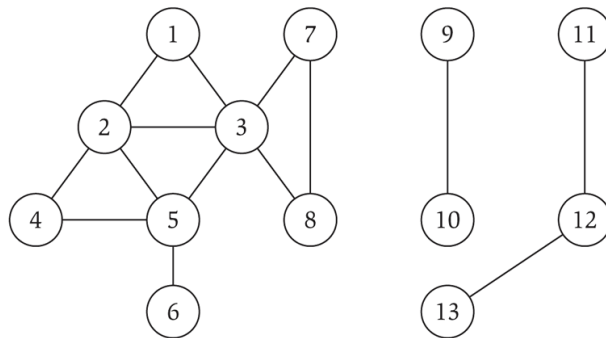
PROGETTAZIONE DI ALGORITMI A.A. 2017-18  
A. DE BONIS

16



## Percorsi e connettività

- **Def.** Un percorso in un grafo non direzionato  $G = (V, E)$  è una sequenza  $P$  di nodi  $v_1, v_2, \dots, v_{k-1}, v_k$  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  $v$ .

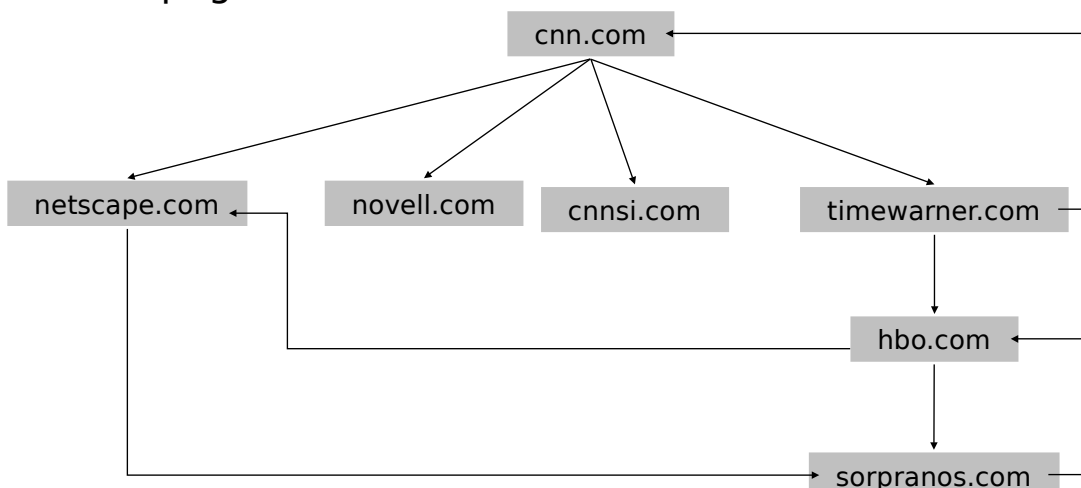


PROGETTAZIONE DI ALGORITMI A.A. 2017-18  
A. DE BONIS

17

## Applicazione del concetto di percorso

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



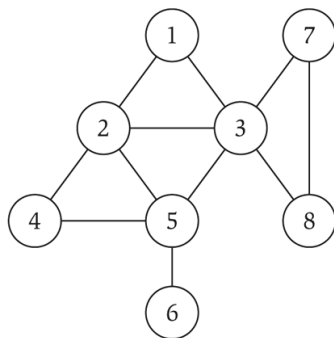
18

- 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.

19

### Cicli

**Def.** Un ciclo è un percorso  $v_1, v_2, \dots, v_{k-1}, v_k$  in cui  $v_1 = v_k$ ,  $k > 2$ , e per ogni altra coppia di nodi  $v_i, v_j$  si ha che  $v_i$  e  $v_j$  sono distinti



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

20

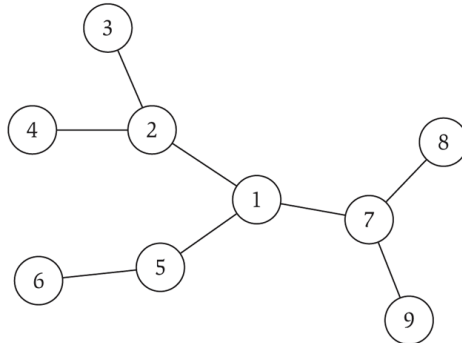
## Alberi

⌘ **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  $\implies$  3 ; 1 e 3  $\implies$  2 ; 2 e 3  $\implies$  1

1.  $G$  è connesso.
  2.  $G$  non contiene cicli.
  3.  $G$  ha  $n-1$  archi.
- } albero



PROGETTAZIONE DI ALGORITMI A.A. 2017-18  
A. DE BONIS

21

## Alberi con radice

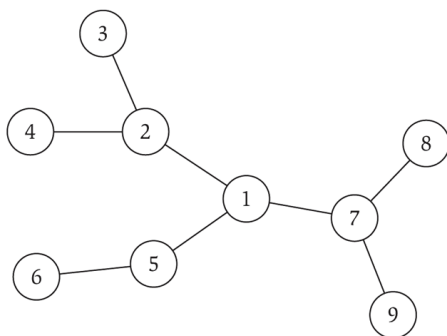
**Albero con radice.** Dato un albero  $T$ , si sceglie un nodo radice  $r$  e si considerano gli archi di  $T$  come orientati a partire da  $r$

Dato un nodo  $v$  di  $T$  si dice

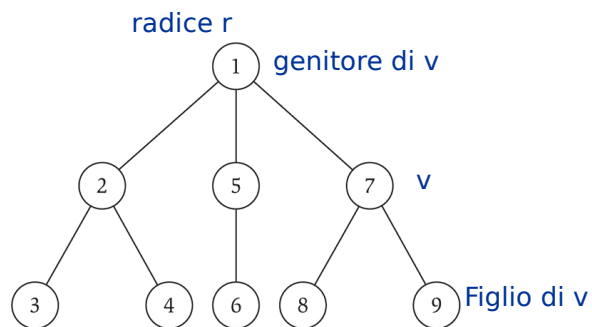
**Genitore di  $v$ :** il nodo che precede  $v$  lungo il percorso da  $r$  a  $v$

**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

PROGETTAZIONE DI ALGORITMI A.A. 2017-18  
A. DE BONIS

2

## Alberi con radice

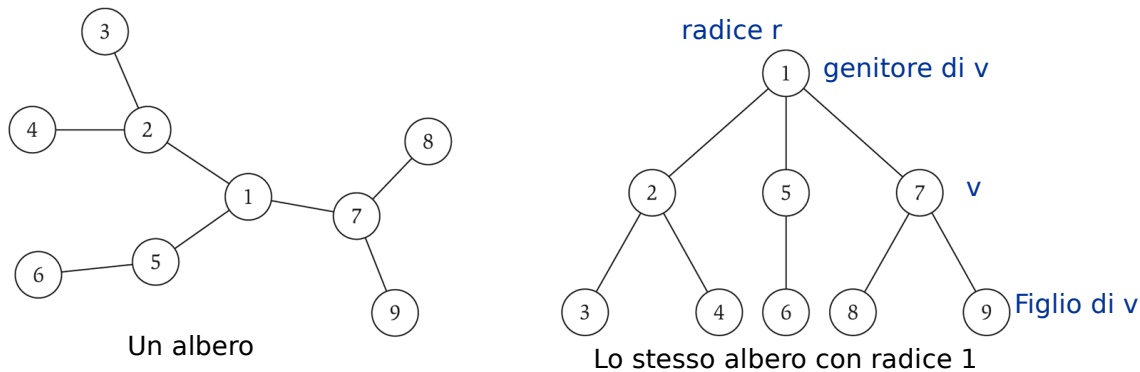
**Albero con radice.** Dato un albero  $T$ , si sceglie un nodo radice  $r$  e si considerano gli archi di  $T$  come orientati a partire da  $r$

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

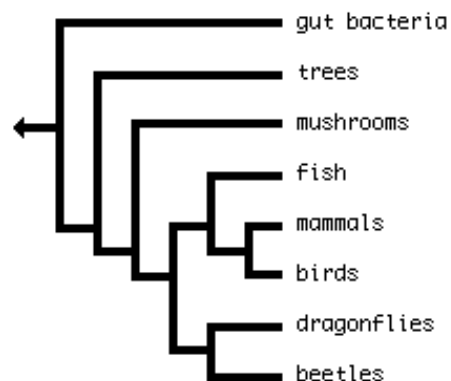


PROGETTAZIONE DI ALGORITMI A.A. 2017-18  
A. DE BONIS

2

## 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.

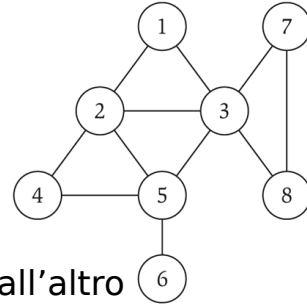
PROGETTAZIONE DI ALGORITMI A.A. 2017-18  
A. DE BONIS

24

## 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  $t$ ?

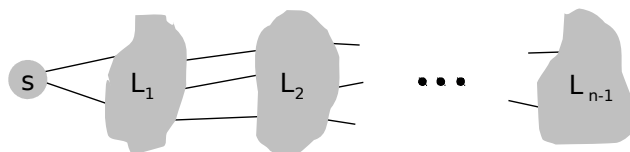


✘ Applicazioni problema percorso piu` corto.

- Minimo numero di scali per andare da un aeroporto all'altro
- 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 possibili direzioni e visitando i nodi livello per livello (N.B.: il libro li chiama layer e cioè strati).

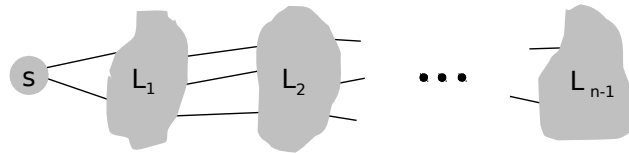


• **Algoritmo BFS.**

- $L_0 = \{ s \}$ .
- $L_1$  = 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

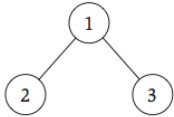
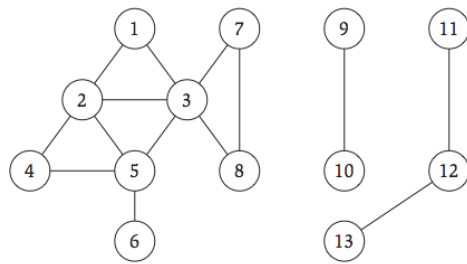
1. BFS( $s$ )
2.  $L_0 = \{ s \}$
3. **For**( $i=0; i \leq n-2; i++$ )
4.  $L_{i+1} = \emptyset$  ;
5. **For each** nodo  $u$  in  $L_i$
6.   **For each** nodo  $v$  adiacente ad  $u$
7.     **if**(  $v$  non appartiene ad  $L_0, \dots, L_i$ )
8.        $L_{i+1} = L_{i+1} \cup \{ v \}$
10. **Endfor**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

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

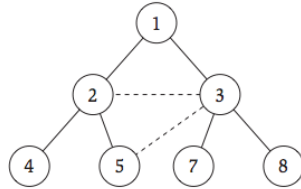
Occorre un modo per capire se un nodo è già stato visitato in precedenza. Il tempo di esecuzione dipende dal modo scelto, da come è implementato il grafo e da come sono rappresentati gli insiemi  $L_i$  che rappresentano i livelli

## Esempio di esecuzione di BFS

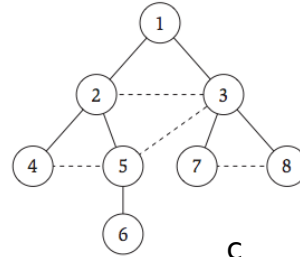
G



a



b



c

$$L_0 = \{1\}$$

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

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

$$c. L_3 = \{6\}$$

29

## Breadth First Search Tree (Albero BFS)

- **Proprietà.** L'algoritmo BFS produce un albero che ha come radice la sorgente e come nodi tutti i nodi del grafo raggiungibili da s. Un nodo v è figlio di un nodo u se la BFS visita v per la prima volta quando esamina i nodi adiacenti ad u.
- **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**  $u \in L[i]$
9.         **For each** 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**

31

## 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  $v$   $O(n)$
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  $O(1)$
6. **While**  $L[i]$  non è vuota Al più  $n-1$  volte in quanto al più  $n-1$  livelli
7.     Inizializza  $L[i+1]$  con una lista vuota
8.     **For each**  $u \in L[i]$
9.         **For each** 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**

Algoritmo è  
 $O(n+m)$

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 =

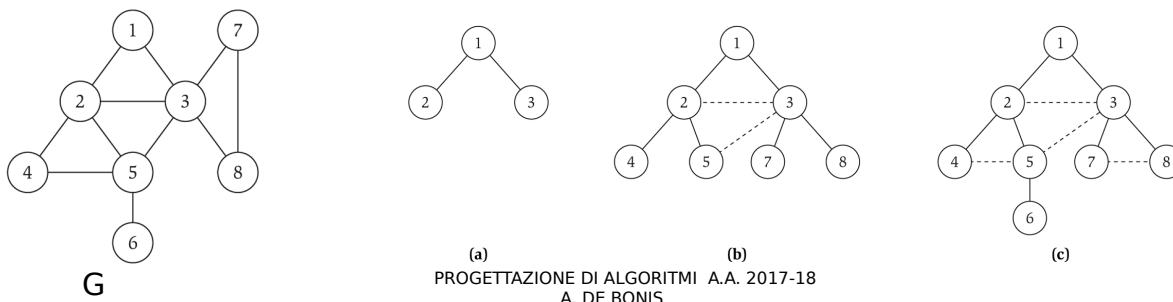
$$\sum_{i=0}^n \sum_{u \in L[i]} \text{deg}(u) \leq \sum_{u \in V} \text{deg}(u) \leq 2m$$

32



## Breadth First Search Tree

- **Proprietà.** Si consideri un'esecuzione di BFS su  $G = (V, E)$ , 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  $j = i+1$ .



33

## Implementazione di BFS con coda FIFO

- L'algoritmo BFS si presta ad essere implementate con un coda FIFO
- 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. Inizializza il BFS tree  $T$  con un albero vuoto
4. Inserisci  $s$  in coda a  $Q$  con una enqueue
5. While( $Q$  non è vuota)
  6. estrai il front di  $Q$  con una deque e ponilo in  $u$
  7. For each arco  $(u, v)$  incidente su  $u$
  8. If( $Discovered[v] = false$ )
  9. Poni  $Discovered[v] = true$
  10. aggiungi  $v$  in coda a  $Q$  con una enqueue
  11. aggiungi  $(u, v)$  al BFS tree
  12. Endif
13. Endwhile
14. Endwhile

34