

# Lezione 22

# Sommario della Lezione

- ▶ Introduzione ai grafi

# Sommario della Lezione

- ▶ Introduzione ai grafi
- ▶ Definizioni, applicazioni,...

# Sommario della Lezione

- ▶ Introduzione ai grafi
- ▶ Definizioni, applicazioni,...
- ▶ Rappresentazioni comuni di grafi

# Sommario della Lezione

- ▶ Introduzione ai grafi
- ▶ Definizioni, applicazioni,...
- ▶ Rappresentazioni comuni di grafi
- ▶ Esplorazione di grafi

## Definizione di base

Formalmente un grafo  $G$  consiste di una coppia di insiemi  $(V, E)$ , dove

- ▶  $V =$ insieme dei vertici del grafo

## Definizione di base

Formalmente un grafo  $G$  consiste di una coppia di insiemi  $(V, E)$ , dove

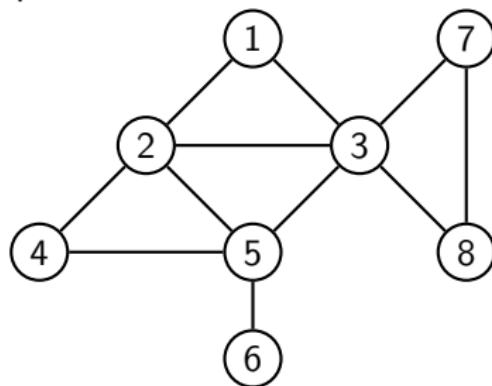
- ▶  $V$  =insieme dei vertici del grafo
- ▶  $E$  =insieme degli archi tra coppie di vertici del grafo

## Definizione di base

Formalmente un grafo  $G$  consiste di una coppia di insiemi  $(V, E)$ , dove

- ▶  $V$  = insieme dei vertici del grafo
- ▶  $E$  = insieme degli archi tra coppie di vertici del grafo

Esempio:

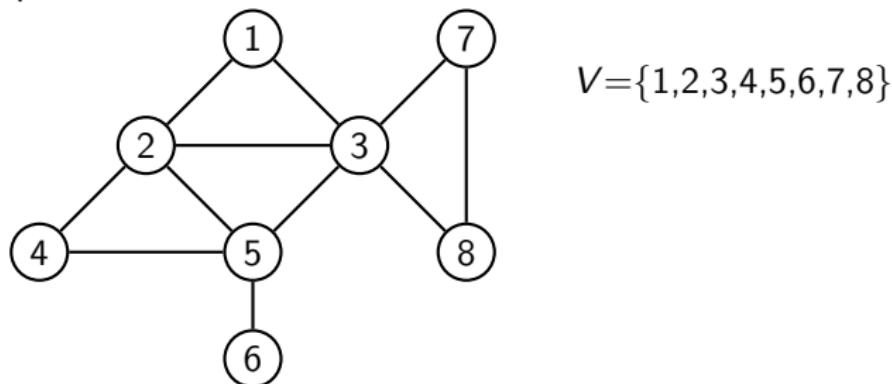


## Definizione di base

Formalmente un grafo  $G$  consiste di una coppia di insiemi  $(V, E)$ , dove

- ▶  $V$  = insieme dei vertici del grafo
- ▶  $E$  = insieme degli archi tra coppie di vertici del grafo

Esempio:

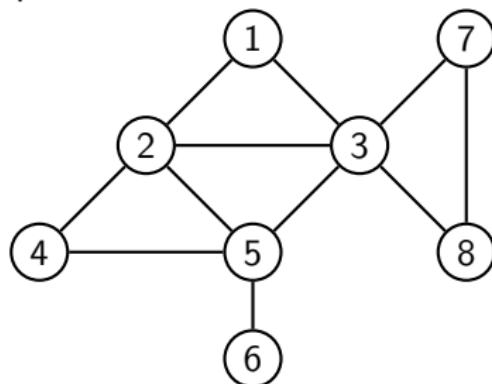


## Definizione di base

Formalmente un grafo  $G$  consiste di una coppia di insiemi  $(V, E)$ , dove

- ▶  $V$  = insieme dei vertici del grafo
- ▶  $E$  = insieme degli archi tra coppie di vertici del grafo

Esempio:



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

$$E = \{1-2, 1-3, 2-3, 2-4, 2-5,$$

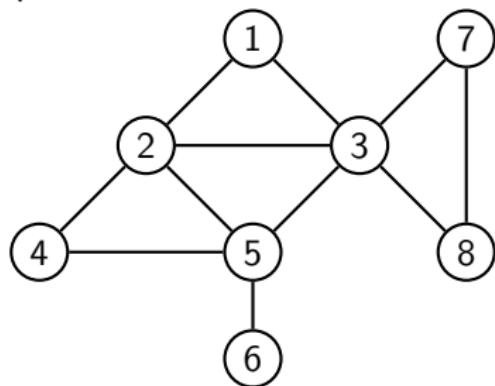
$$4-5, 5-6, 3-5, 3-7, 3-8, 7-8\}$$

# Definizione di base

Formalmente un grafo  $G$  consiste di una coppia di insiemi  $(V, E)$ , dove

- ▶  $V$  =insieme dei vertici del grafo
- ▶  $E$  =insieme degli archi tra coppie di vertici del grafo

Esempio:



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

$$E=\{1-2, 1-3, 2-3, 2-4, 2-5,$$

$$4-5, 5-6, 3-5, 3-7, 3-8, 7-8\}$$

Tipicamente gli archi descrivono “relazioni” tra coppie di “entità ” (rappresentate dai vertici).

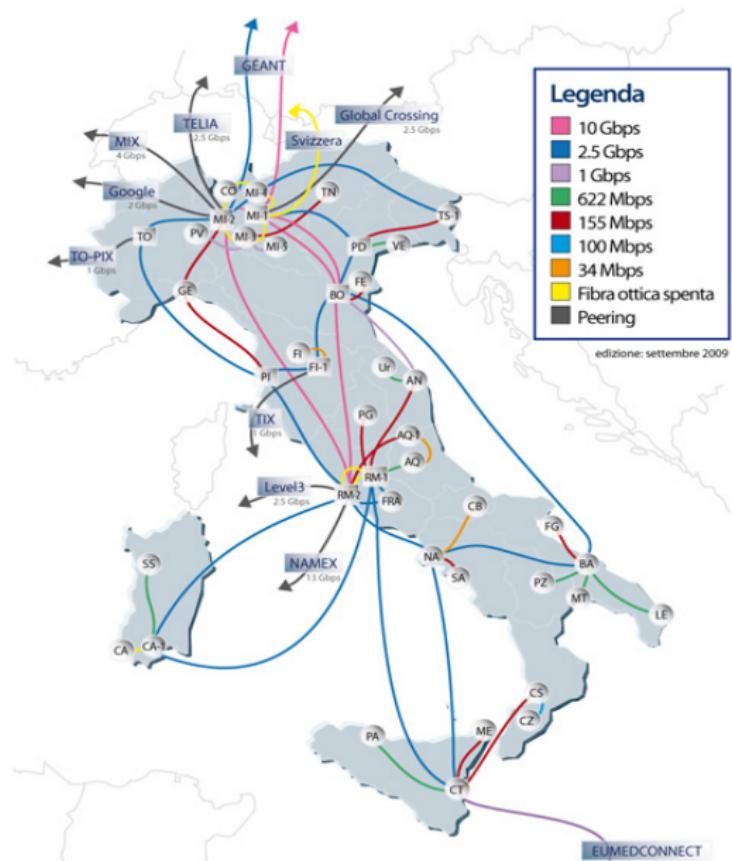
# I grafi trovano applicazioni in numerosissimi ambiti

# Reti di Comunicazione

Vertici=computers

Archi=Fibre ottiche

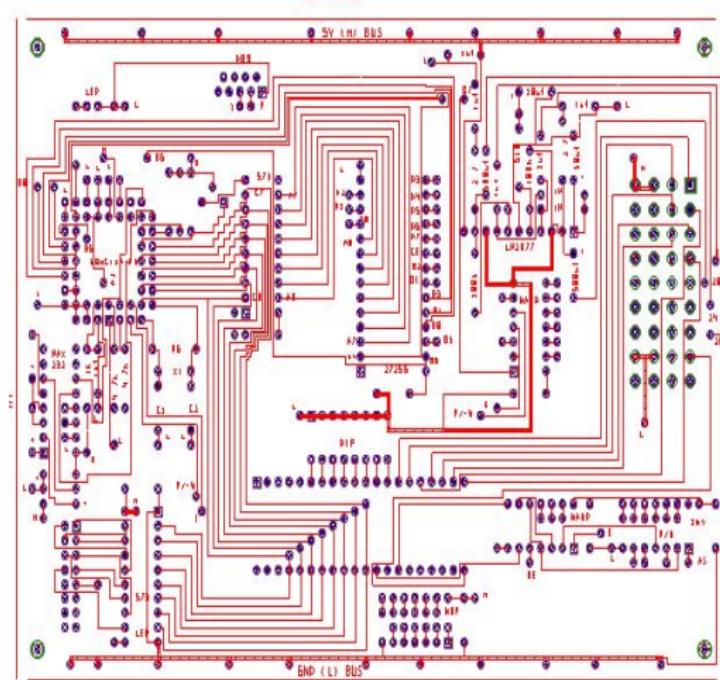
## Topologia di backbone di GARR-G



# Circuiti

**Vertici**=porte/registri/processori

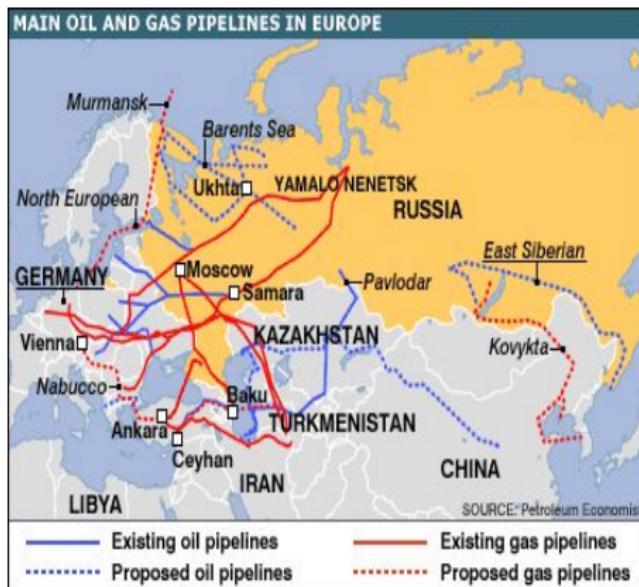
**Archi**=fili



# Gasdotti/Oleodotti

**Vertici**=depositi/stazioni di pompaggio

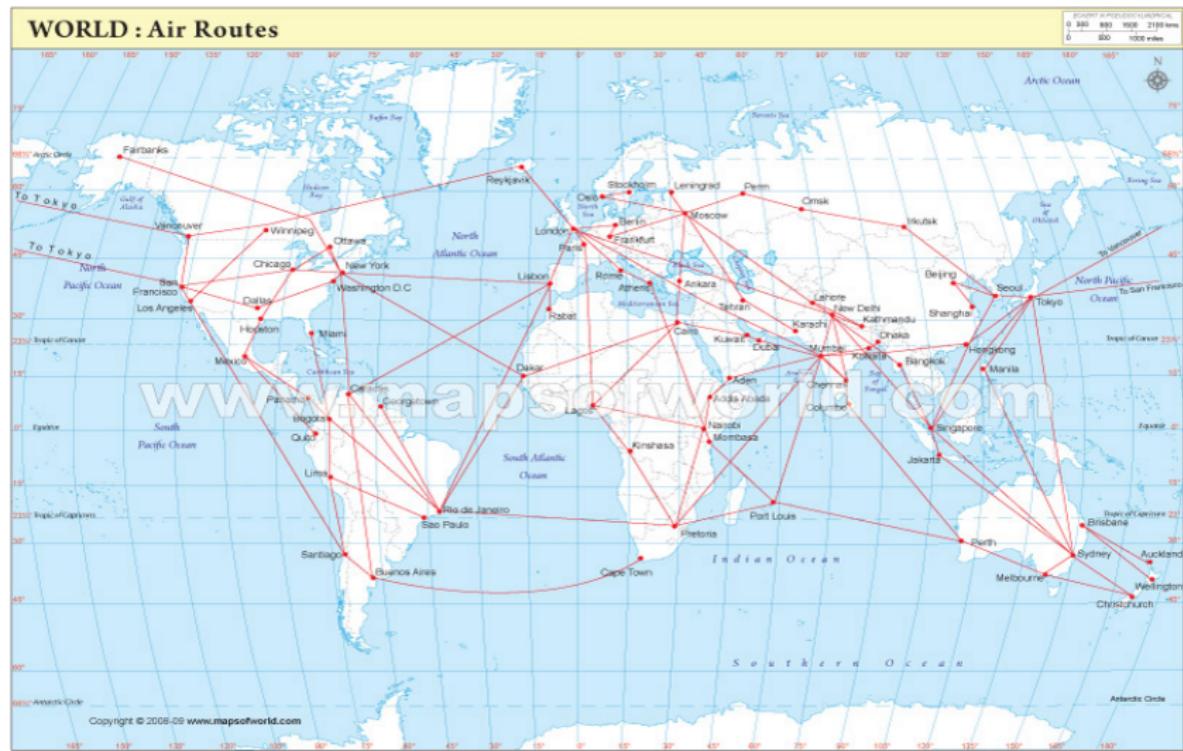
**Archi**=tubazioni



# Reti di trasporto (ad es., reti aeree)

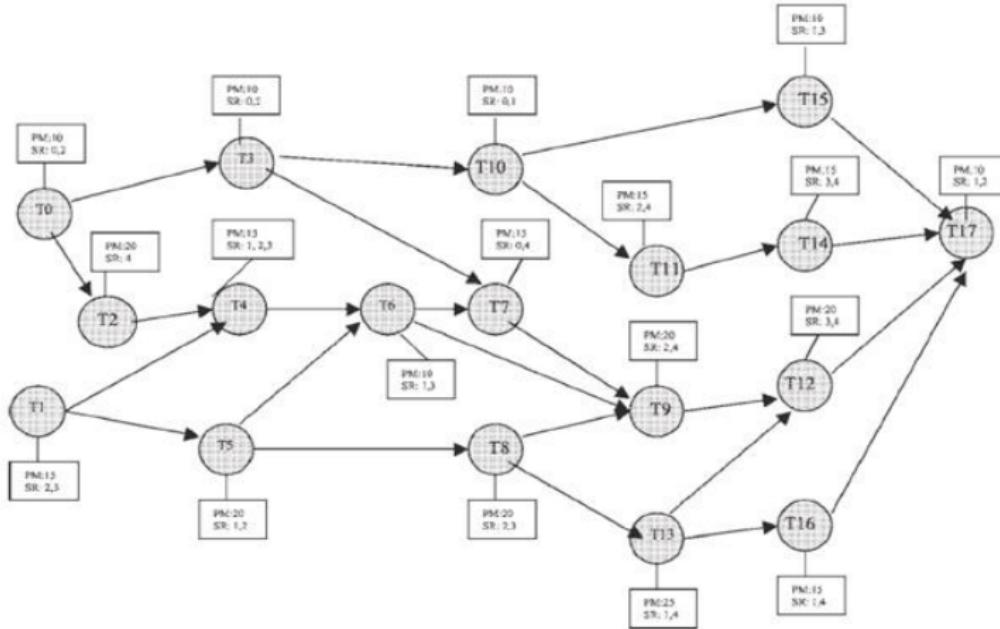
**Vertici**= aeroporti

**Archi**=tratte aeree



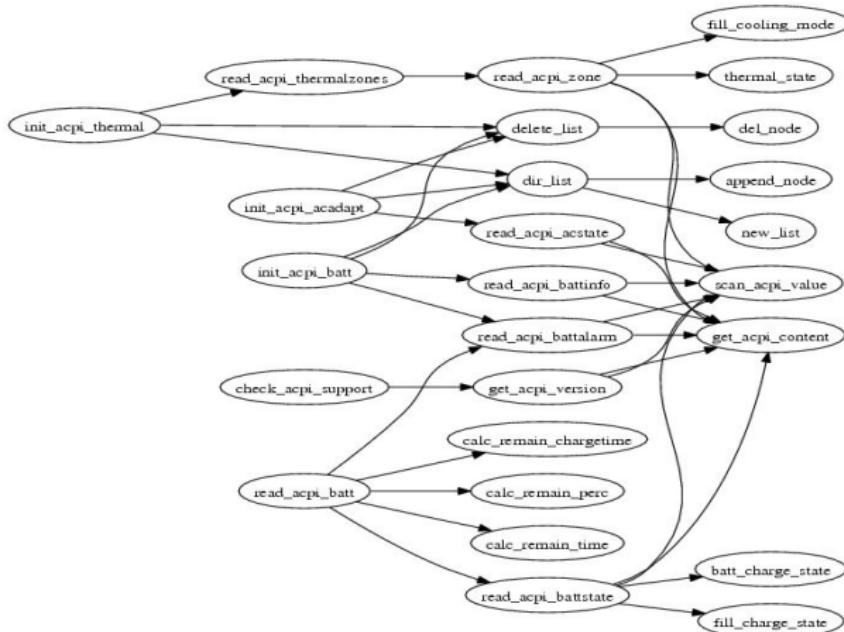
# Schedulazione di attività

**Vertici**= attività , **Archi**=vincoli di precedenza



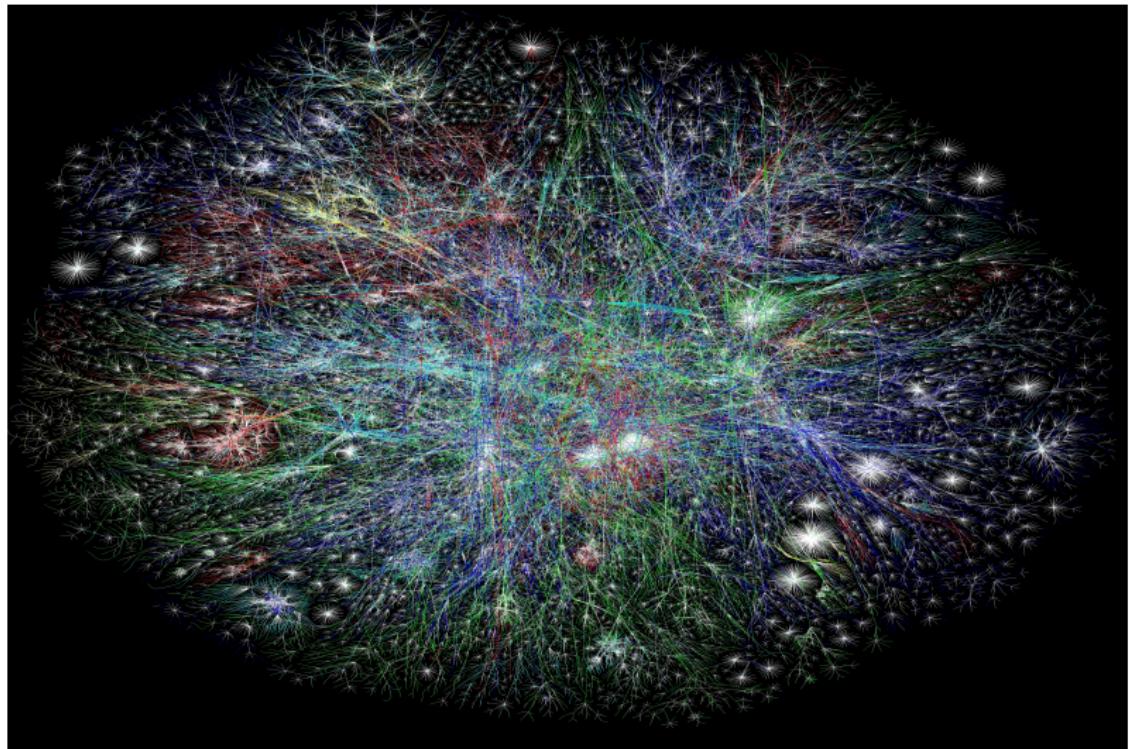
# Sistemi Software

**Vertici**= funzioni, **Archi**=chiamate tra funzioni



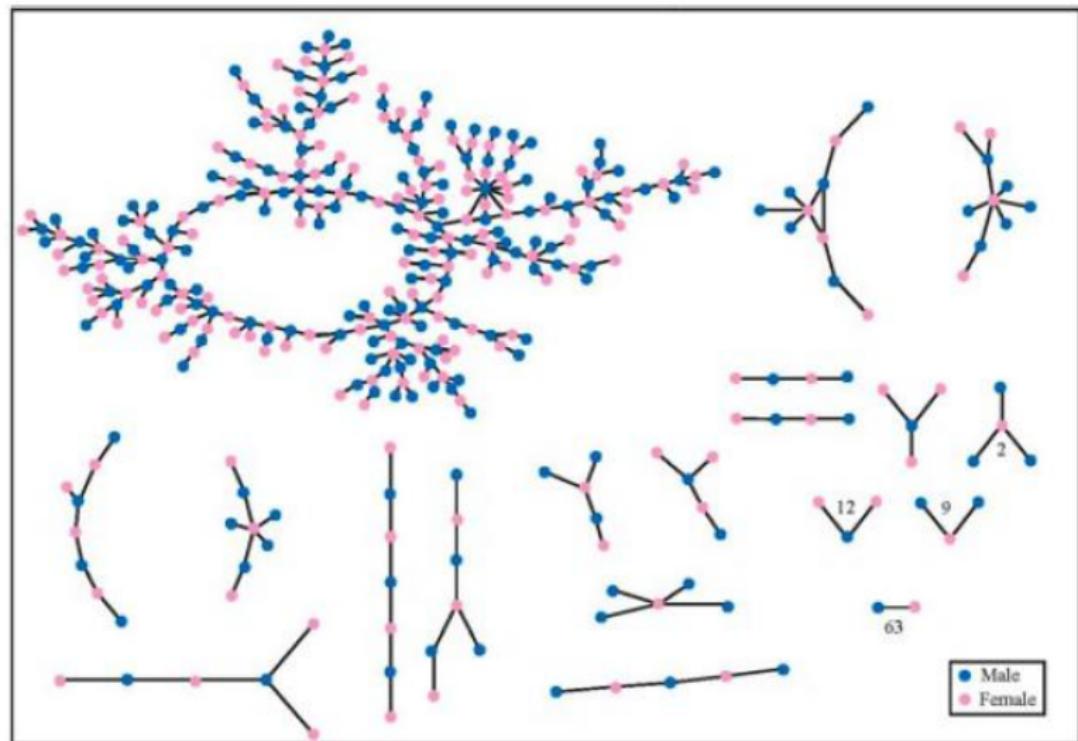
# World Wide Web

**Vertici**= pagine web, **Archi**=hyperlink

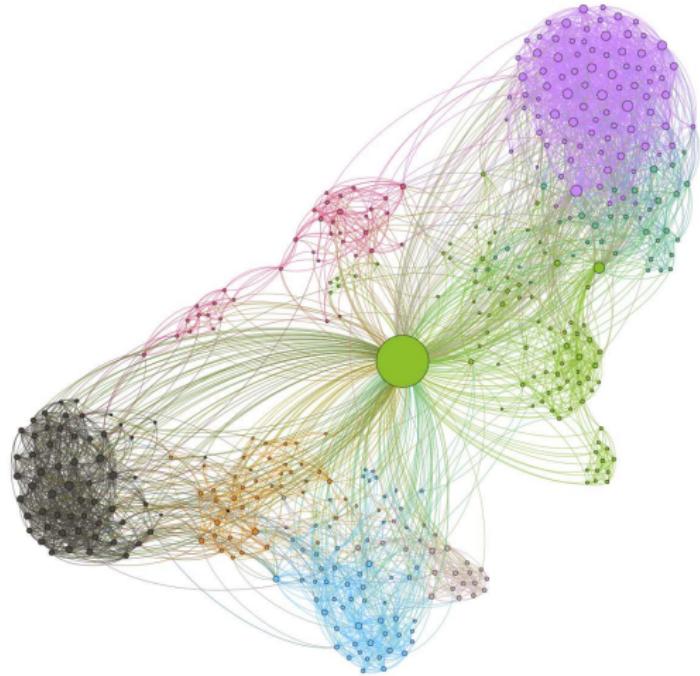


# Reti sociali

**Vertici**= persone, **Archi**=vincoli relazionali

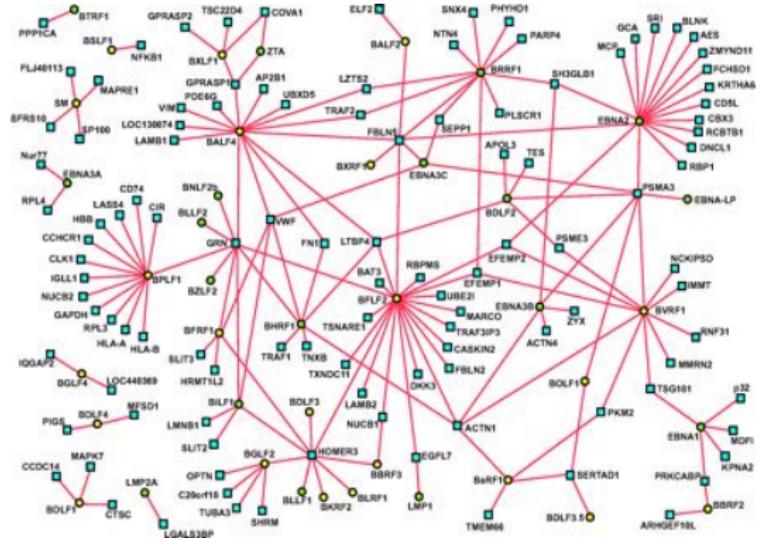


## Relazioni di amicizia di un iscritto a Facebook



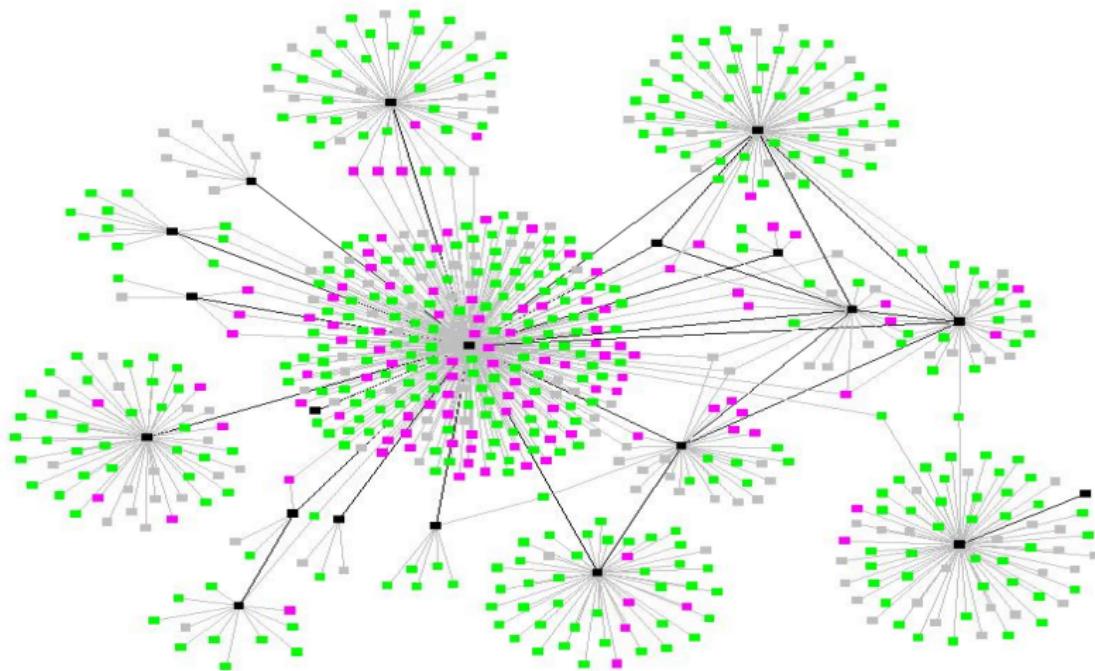
## Biologia: Interazione tra proteine

**Vertici**= proteine, **Archi**=interazione tra proteine



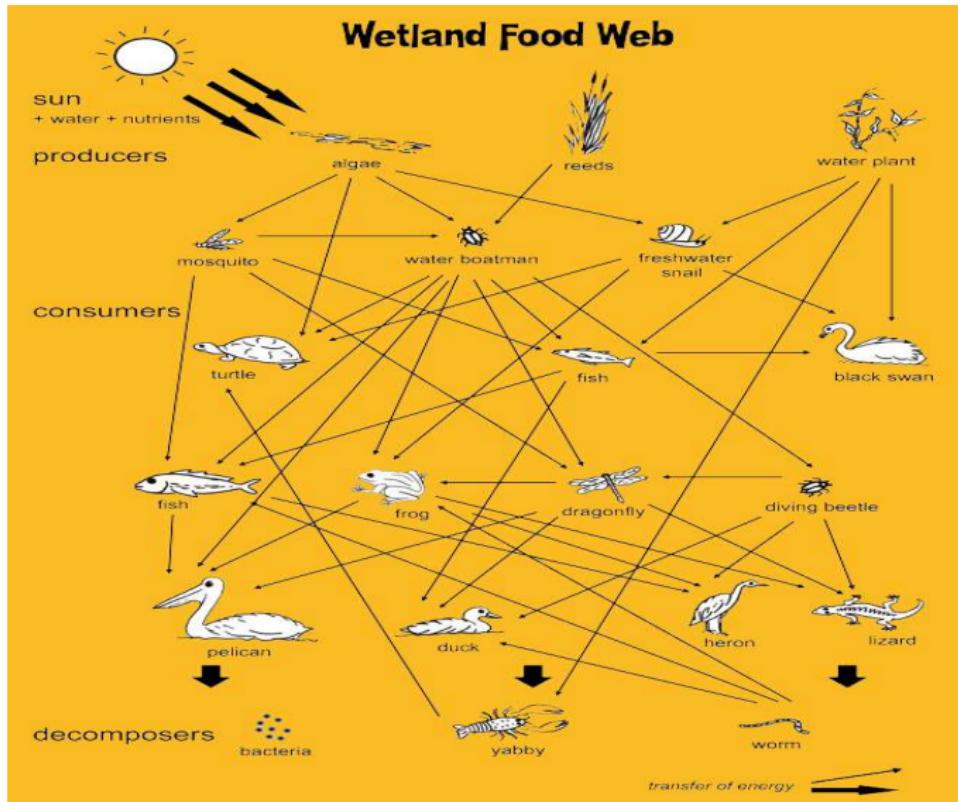
# Medicina: Diffusione di malattie

**Vertici**= persone, **Archi**=relazioni di contagio



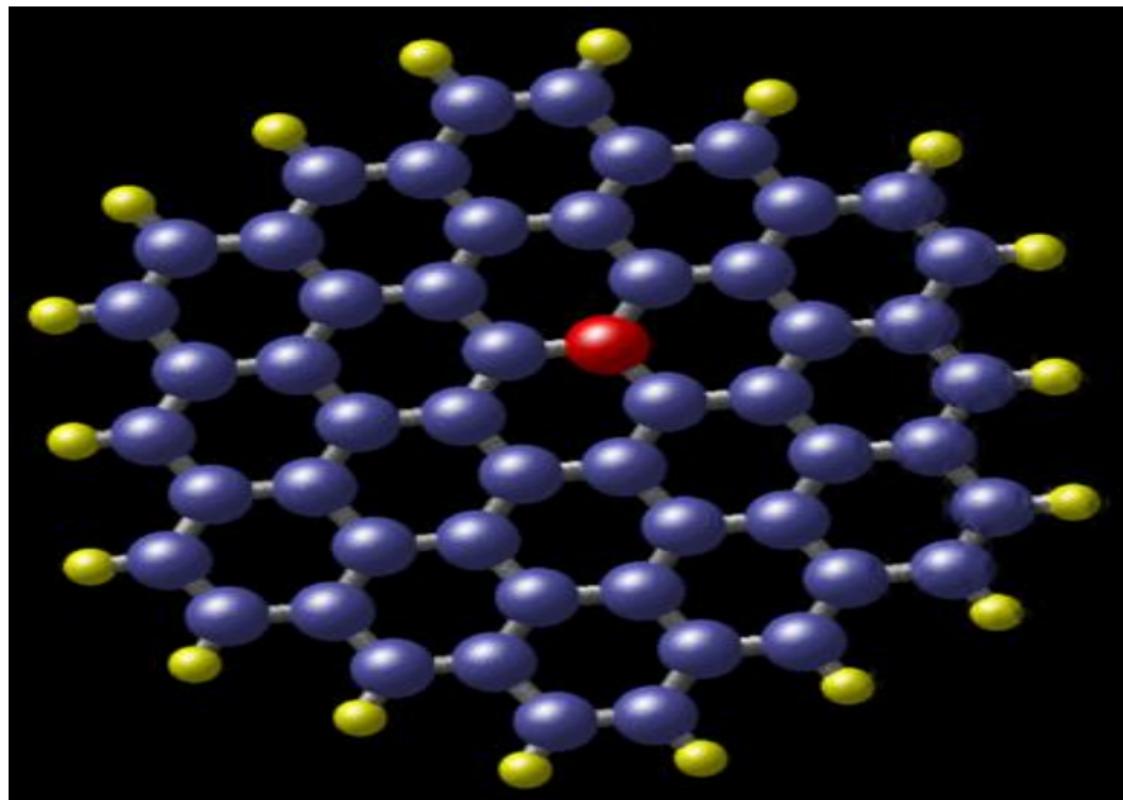
# Ecologia: reti alimentari

Vertici= specie animali, Archi=relazioni preda-predatore



# Chimica: composti

**Vertici**= molecole, **Archi**=legami chimici



## Grafi non-diretti e Grafi diretti

Dagli esempi precedenti, si vede che gli archi di un grafo possono rappresentare sia relazioni "simmetriche" tra vertici (es., nella rete GARR se Napoli è connessa a Salerno allora Salerno è ovviamente e simmetricamente connessa a Napoli),

## Grafi non-diretti e Grafi diretti

Dagli esempi precedenti, si vede che gli archi di un grafo possono rappresentare sia relazioni “simmetriche” tra vertici (es., nella rete GARR se Napoli è connessa a Salerno allora Salerno è ovviamente e simmetricamente connessa a Napoli), che relazioni “asimmetriche” (es., nel grafo alimentare, il Cigno mangia il Pesce ma *non vale* il viceversa)

- ▶ Per rappresentare relazioni simmetriche si usano **grafi non diretti**.

- ▶ Per rappresentare relazioni simmetriche si usano **grafi non diretti**. Come dice la parola, in tali grafi gli archi non hanno direzioni e *non* si fa differenza tra l'arco dal vertice  $u$  al vertice  $v$  e l'arco dal vertice  $v$  al vertice  $u$ .

- ▶ Per rappresentare relazioni simmetriche si usano **grafi non diretti**. Come dice la parola, in tali grafi gli archi non hanno direzioni e *non* si fa differenza tra l'arco dal vertice  $u$  al vertice  $v$  e l'arco dal vertice  $v$  al vertice  $u$ . Formalmente, un grafo  $G$  non diretto è rappresentato da una coppia  $G = (V, E)$ ,

- ▶ Per rappresentare relazioni simmetriche si usano **grafi non diretti**. Come dice la parola, in tali grafi gli archi non hanno direzioni e *non* si fa differenza tra l'arco dal vertice  $u$  al vertice  $v$  e l'arco dal vertice  $v$  al vertice  $u$ . Formalmente, un grafo  $G$  non diretto è rappresentato da una coppia  $G = (V, E)$ , dove  $V$  è l'insieme dei vertici,

- ▶ Per rappresentare relazioni simmetriche si usano **grafi non diretti**. Come dice la parola, in tali grafi gli archi non hanno direzioni e *non* si fa differenza tra l'arco dal vertice  $u$  al vertice  $v$  e l'arco dal vertice  $v$  al vertice  $u$ . Formalmente, un grafo  $G$  non diretto è rappresentato da una coppia  $G = (V, E)$ , dove  $V$  è l'insieme dei vertici, ed un arco  $e \in E$  tra i vertici  $u, v \in V$  consiste del sottoinsieme  $\{u, v\} \subset V$  (per cui  $\{u, v\} = \{v, u\}$ ).

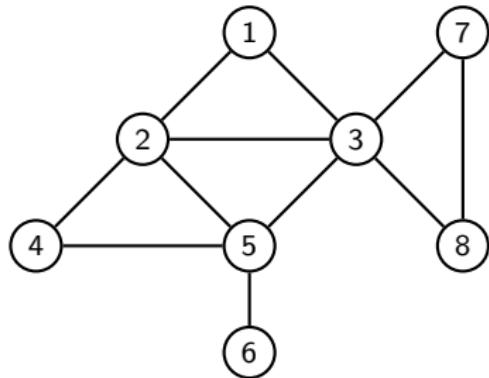
- ▶ Per rappresentare relazioni simmetriche si usano **grafi non diretti**. Come dice la parola, in tali grafi gli archi non hanno direzioni e *non* si fa differenza tra l'arco dal vertice  $u$  al vertice  $v$  e l'arco dal vertice  $v$  al vertice  $u$ . Formalmente, un grafo  $G$  non diretto è rappresentato da una coppia  $G = (V, E)$ , dove  $V$  è l'insieme dei vertici, ed un arco  $e \in E$  tra i vertici  $u, v \in V$  consiste del sottoinsieme  $\{u, v\} \subset V$  (per cui  $\{u, v\} = \{v, u\}$ ).
- ▶ Per rappresentare relazioni asimmetriche si usano **grafi diretti**.

- ▶ Per rappresentare relazioni simmetriche si usano **grafi non diretti**. Come dice la parola, in tali grafi gli archi non hanno direzioni e *non* si fa differenza tra l'arco dal vertice  $u$  al vertice  $v$  e l'arco dal vertice  $v$  al vertice  $u$ . Formalmente, un grafo  $G$  non diretto è rappresentato da una coppia  $G = (V, E)$ , dove  $V$  è l'insieme dei vertici, ed un arco  $e \in E$  tra i vertici  $u, v \in V$  consiste del sottoinsieme  $\{u, v\} \subset V$  (per cui  $\{u, v\} = \{v, u\}$ ).
- ▶ Per rappresentare relazioni asimmetriche si usano **grafi diretti**. In essi gli archi hanno “direzioni” e c’è differenza tra l’arco dal vertice  $u$  al vertice  $v$  e l’arco dal vertice  $v$  al vertice  $u$ .

- ▶ Per rappresentare relazioni simmetriche si usano **grafi non diretti**. Come dice la parola, in tali grafi gli archi non hanno direzioni e *non* si fa differenza tra l'arco dal vertice  $u$  al vertice  $v$  e l'arco dal vertice  $v$  al vertice  $u$ . Formalmente, un grafo  $G$  non diretto è rappresentato da una coppia  $G = (V, E)$ , dove  $V$  è l'insieme dei vertici, ed un arco  $e \in E$  tra i vertici  $u, v \in V$  consiste del sottoinsieme  $\{u, v\} \subset V$  (per cui  $\{u, v\} = \{v, u\}$ ).
- ▶ Per rappresentare relazioni asimmetriche si usano **grafi diretti**. In essi gli archi hanno “direzioni” e c’è differenza tra l’arco dal vertice  $u$  al vertice  $v$  e l’arco dal vertice  $v$  al vertice  $u$ . Formalmente, un grafo  $G$  diretto è rappresentato da una coppia  $G = (V, E)$ , dove  $V$  è sempre l’insieme dei vertici, ed un arco  $e \in E$  dal vertice  $u$  ad il vertice  $v$  consiste della coppia *ordinata*  $(u, v)$

- ▶ Per rappresentare relazioni simmetriche si usano **grafi non diretti**. Come dice la parola, in tali grafi gli archi non hanno direzioni e *non* si fa differenza tra l'arco dal vertice  $u$  al vertice  $v$  e l'arco dal vertice  $v$  al vertice  $u$ . Formalmente, un grafo  $G$  non diretto è rappresentato da una coppia  $G = (V, E)$ , dove  $V$  è l'insieme dei vertici, ed un arco  $e \in E$  tra i vertici  $u, v \in V$  consiste del sottoinsieme  $\{u, v\} \subset V$  (per cui  $\{u, v\} = \{v, u\}$ ).
- ▶ Per rappresentare relazioni asimmetriche si usano **grafi diretti**. In essi gli archi hanno “direzioni” e c’è differenza tra l’arco dal vertice  $u$  al vertice  $v$  e l’arco dal vertice  $v$  al vertice  $u$ . Formalmente, un grafo  $G$  diretto è rappresentato da una coppia  $G = (V, E)$ , dove  $V$  è sempre l’insieme dei vertici, ed un arco  $e \in E$  dal vertice  $u$  ad il vertice  $v$  consiste della coppia *ordinata*  $(u, v)$  (per cui  $(u, v) \neq (v, u)$ , e se nel grafo c’è l’arco  $(u, v)$  non è detto che nel grafo che ci sia anche  $(v, u)$ ).

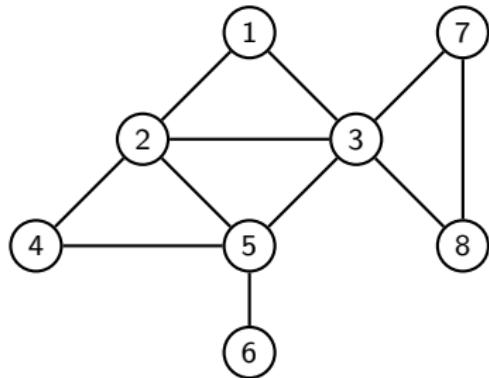
Esempio: a sinistra grafo non diretto, a destra grafo diretto



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

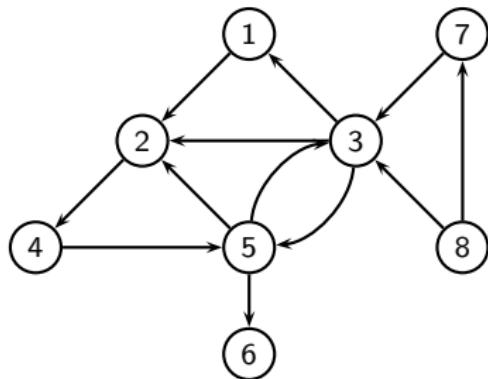
$$E = \{\{1, 2\}, \{1, 3\}, \{2, 3\}, \{2, 4\}, \{2, 5\}, \\ \{4, 5\}, \{5, 6\}, \{3, 5\}, \{3, 7\}, \{3, 8\}, \{7, 8\}\}$$

Esempio: a sinistra grafo non diretto, a destra grafo diretto



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

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



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

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

Come descrivere un grafo  $G = (V, E)$ ,  $|V| = n$ ,  $|E| = m$  ad un calcolatore?

Come descrivere un grafo  $G = (V, E)$ ,  $|V| = n$ ,  $|E| = m$  ad un calcolatore?

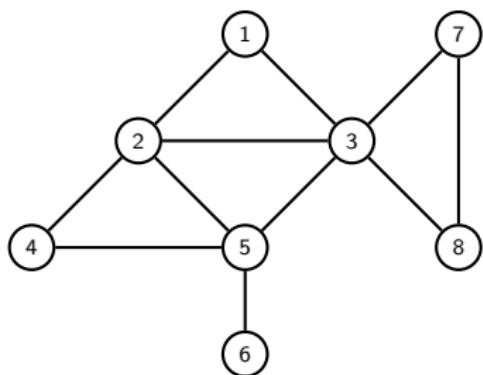
Primo Metodo – Matrice di adiacenza:

Matrice binaria con  $n$  righe ed  $n$  colonne, in cui l'elemento  $A[i, j]$  nella riga  $i$ -esima e colonna  $j$ -esima di  $A$  è uguale a **1** se e solo se esiste nel grafo  $G$  l'arco tra i vertici ***i*** e ***j***.

Come descrivere un grafo  $G = (V, E)$ ,  $|V| = n$ ,  $|E| = m$  ad un calcolatore?

Primo Metodo – Matrice di adiacenza:

Matrice binaria con  $n$  righe ed  $n$  colonne, in cui l'elemento  $A[i, j]$  nella riga  $i$ -esima e colonna  $j$ -esima di  $A$  è uguale a 1 se e solo se esiste nel grafo  $G$  l'arco tra i vertici  $i$  e  $j$ .

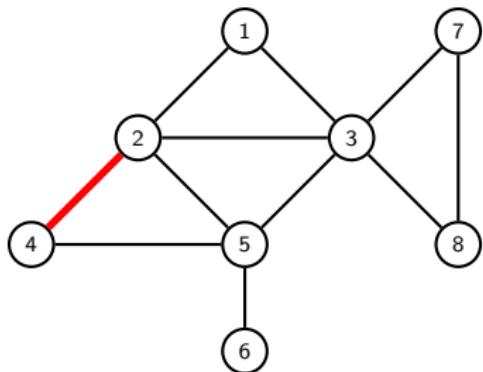


$$A = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 3 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 4 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 5 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 6 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 7 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 8 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Come descrivere un grafo  $G = (V, E)$ ,  $|V| = n$ ,  $|E| = m$  ad un calcolatore?

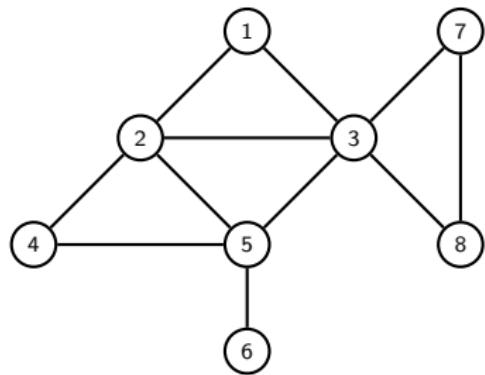
Primo Metodo – Matrice di adiacenza:

Matrice binaria con  $n$  righe ed  $n$  colonne, in cui l'elemento  $A[i, j]$  nella riga  $i$ -esima e colonna  $j$ -esima di  $A$  è uguale a 1 se e solo se esiste nel grafo  $G$  l'arco tra i vertici  $i$  e  $j$ .



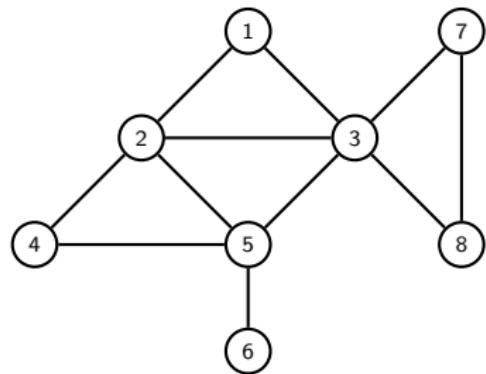
$$A = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 3 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 4 & 0 & \textcolor{red}{1} & 0 & 0 & 1 & 0 & 0 \\ 5 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 6 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 7 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 8 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

## Pregi e difetti della matrice di adiacenza di un grafo



$$A = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 3 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 4 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 5 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 6 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 7 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 8 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

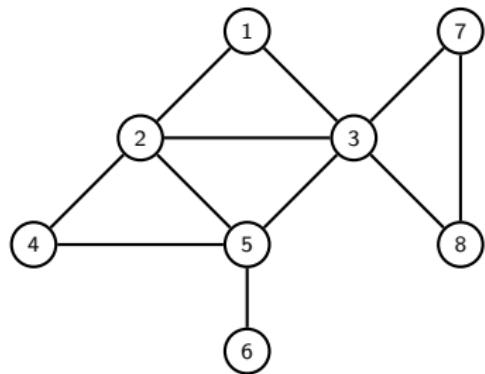
## Pregi e difetti della matrice di adiacenza di un grafo



$$A = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 3 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 4 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 5 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 6 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 7 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 8 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

- Lo spazio richiesto è  $\Theta(n^2)$ ,  $n = |V|$

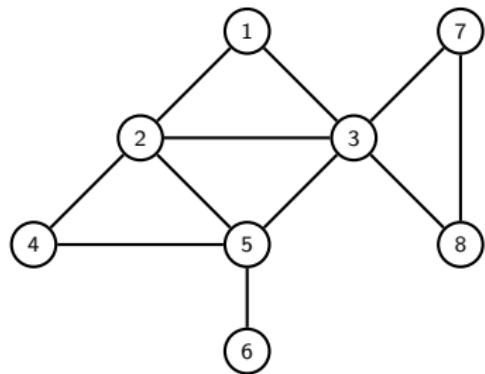
## Pregi e difetti della matrice di adiacenza di un grafo



$$A = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 3 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 4 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 5 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 6 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 7 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 8 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

- Lo spazio richiesto è  $\Theta(n^2)$ ,  $n = |V|$
- Verificare se c'è un arco tra i vertici  $i$  e  $j$  prende tempo  $\Theta(1)$

## Pregi e difetti della matrice di adiacenza di un grafo



$$A = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 3 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 4 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 5 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 6 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 7 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 8 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

- Lo spazio richiesto è  $\Theta(n^2)$ ,  $n = |V|$
- Verificare se c'è un arco tra i vertici  $i$  e  $j$  prende tempo  $\Theta(1)$
- Identificare tutti gli archi del grafo prende tempo  $\Theta(n^2)$

## Secondo Metodo: Liste di Adiacenza

## Secondo Metodo: Liste di Adiacenza

- ▶  $n$  liste a puntatori, una per ogni nodo del grafo

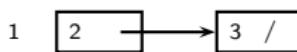
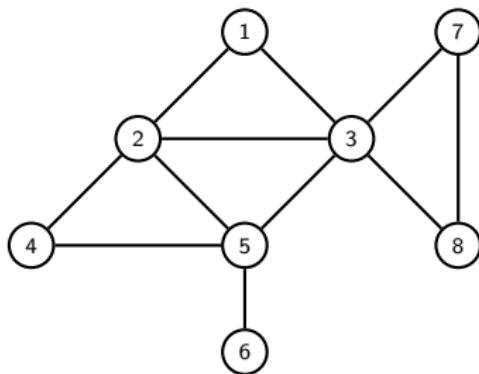
## Secondo Metodo: Liste di Adiacenza

- ▶  $n$  liste a puntatori, una per ogni nodo del grafo
- ▶ La lista associata al generico nodo  $i$  contiene *tutti* i nodi  $j$  vicini ad  $i$  nel grafo.

## Secondo Metodo: Liste di Adiacenza

- ▶  $n$  liste a puntatori, una per ogni nodo del grafo
- ▶ La lista associata al generico nodo  $i$  contiene *tutti* i nodi  $j$  vicini ad  $i$  nel grafo.

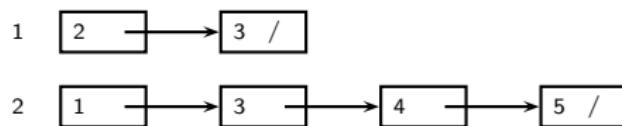
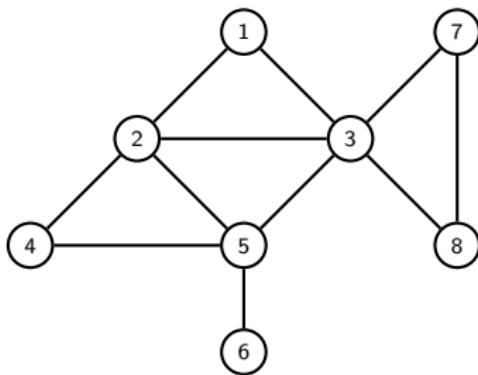
Esempio:



## Secondo Metodo: Liste di Adiacenza

- ▶  $n$  liste a puntatori, una per ogni nodo del grafo
- ▶ La lista associata al generico nodo  $i$  contiene *tutti* i nodi  $j$  vicini ad  $i$  nel grafo.

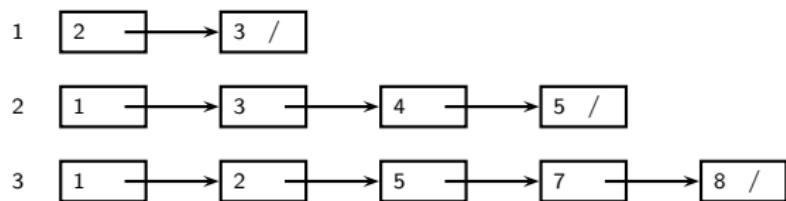
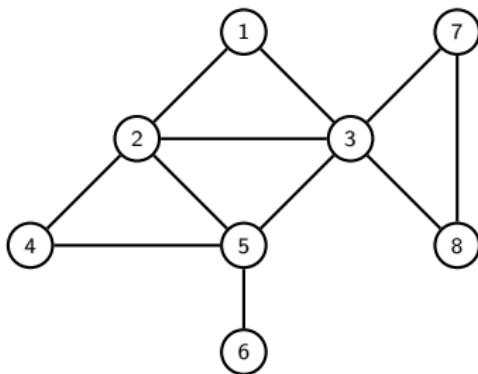
Esempio:



## Secondo Metodo: Liste di Adiacenza

- ▶  $n$  liste a puntatori, una per ogni nodo del grafo
- ▶ La lista associata al generico nodo  $i$  contiene *tutti* i nodi  $j$  vicini ad  $i$  nel grafo.

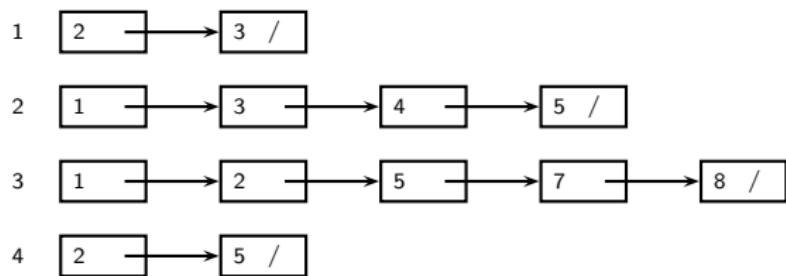
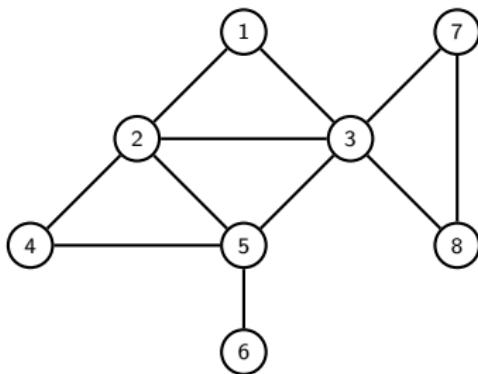
Esempio:



## Secondo Metodo: Liste di Adiacenza

- ▶  $n$  liste a puntatori, una per ogni nodo del grafo
- ▶ La lista associata al generico nodo  $i$  contiene *tutti* i nodi  $j$  vicini ad  $i$  nel grafo.

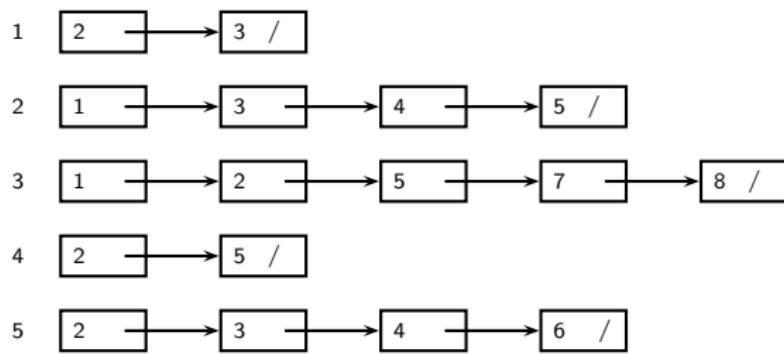
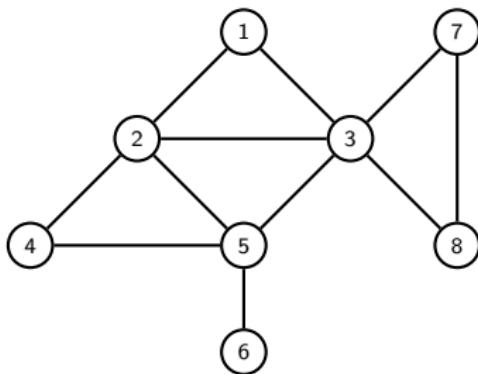
Esempio:



## Secondo Metodo: Liste di Adiacenza

- ▶  $n$  liste a puntatori, una per ogni nodo del grafo
- ▶ La lista associata al generico nodo  $i$  contiene *tutti* i nodi  $j$  vicini ad  $i$  nel grafo.

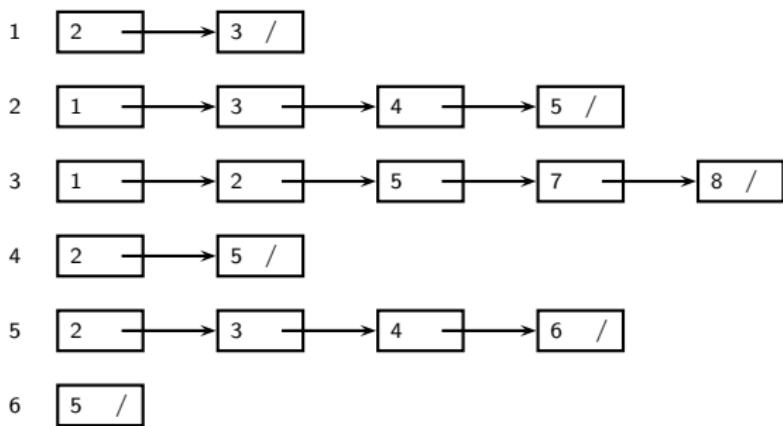
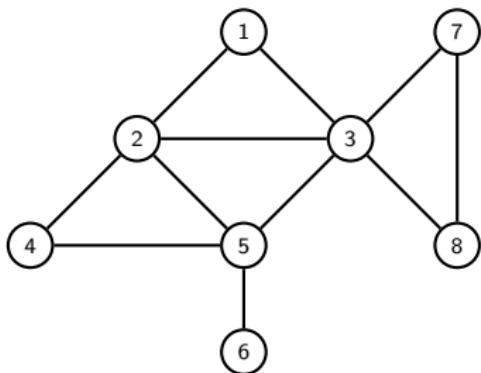
Esempio:



## Secondo Metodo: Liste di Adiacenza

- ▶  $n$  liste a puntatori, una per ogni nodo del grafo
- ▶ La lista associata al generico nodo  $i$  contiene *tutti* i nodi  $j$  vicini ad  $i$  nel grafo.

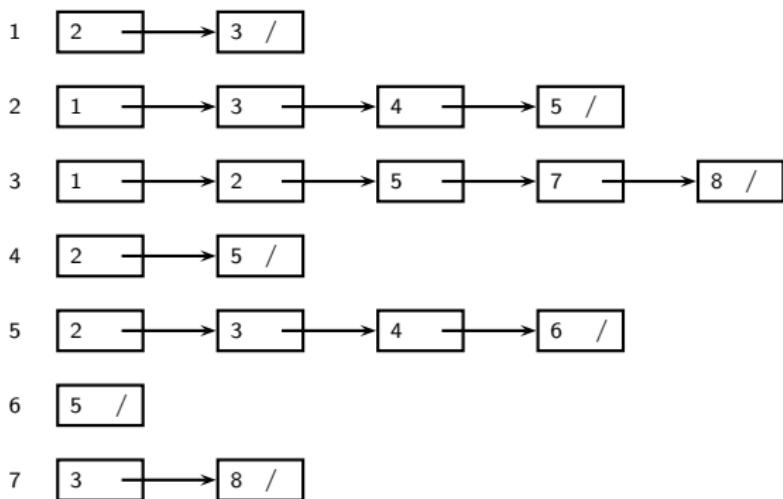
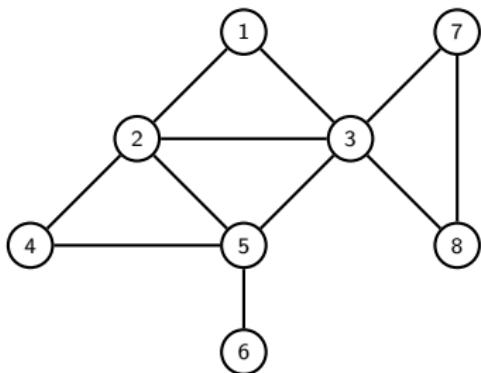
Esempio:



## Secondo Metodo: Liste di Adiacenza

- ▶  $n$  liste a puntatori, una per ogni nodo del grafo
- ▶ La lista associata al generico nodo  $i$  contiene *tutti* i nodi  $j$  vicini ad  $i$  nel grafo.

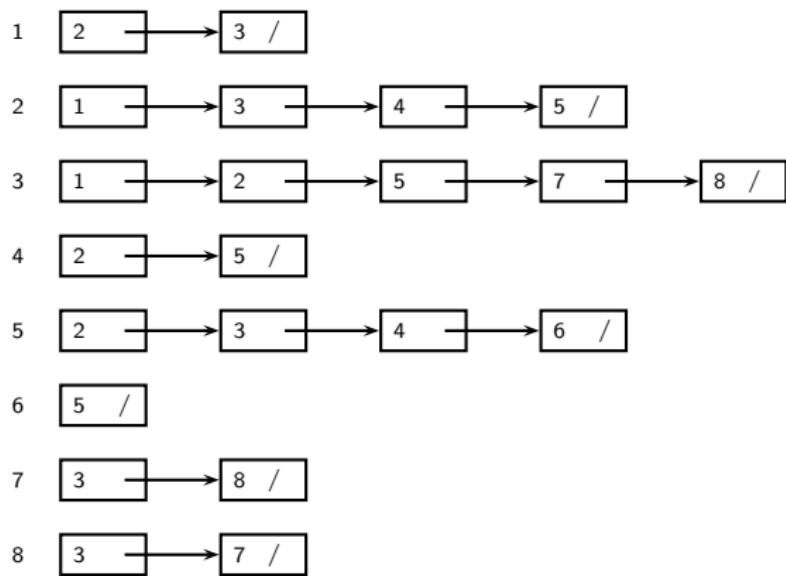
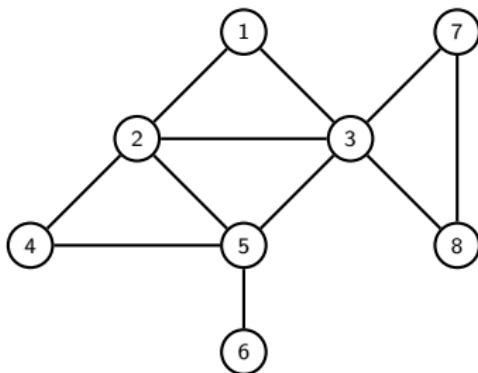
Esempio:



## Secondo Metodo: Liste di Adiacenza

- ▶  $n$  liste a puntatori, una per ogni nodo del grafo
- ▶ La lista associata al generico nodo  $i$  contiene *tutti* i nodi  $j$  vicini ad  $i$  nel grafo.

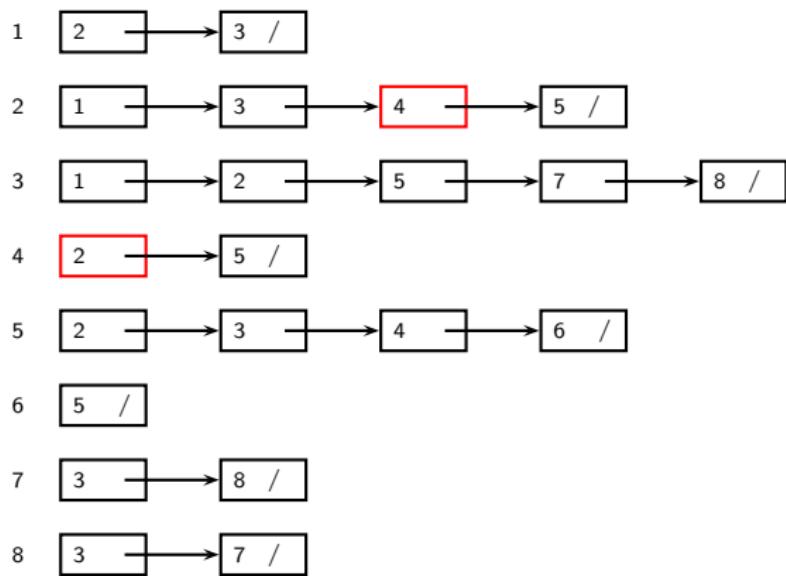
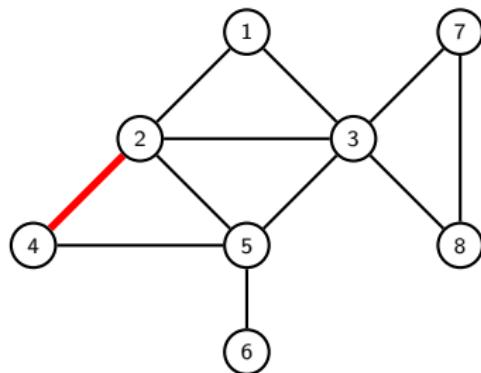
Esempio:



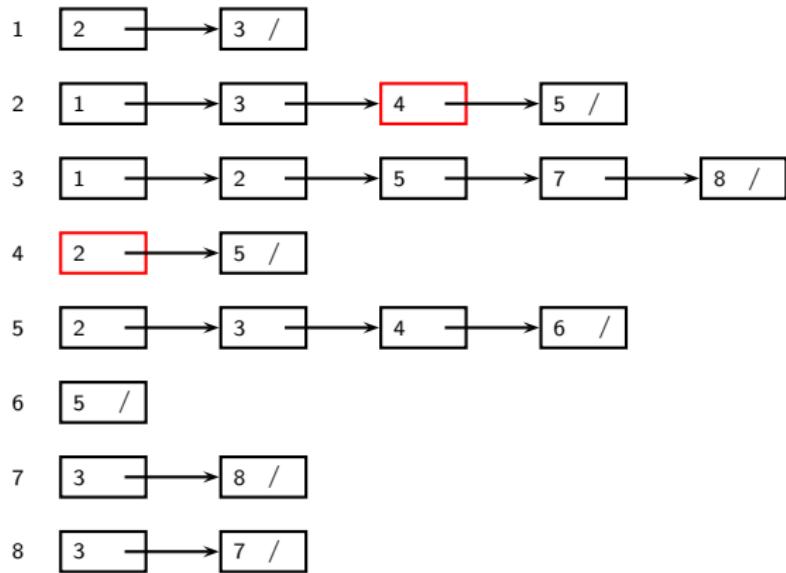
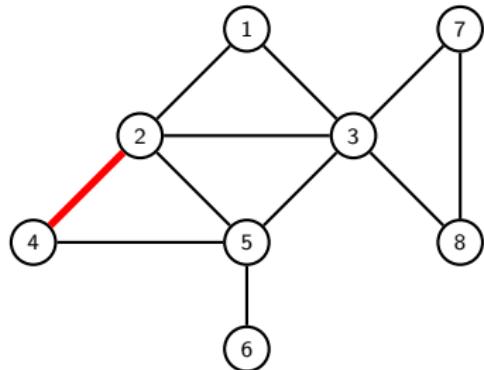
## Secondo Metodo: Liste di Adiacenza

- ▶  $n$  liste a puntatori, una per ogni nodo del grafo
- ▶ La lista associata al generico nodo  $i$  contiene *tutti* i nodi  $j$  vicini ad  $i$  nel grafo.

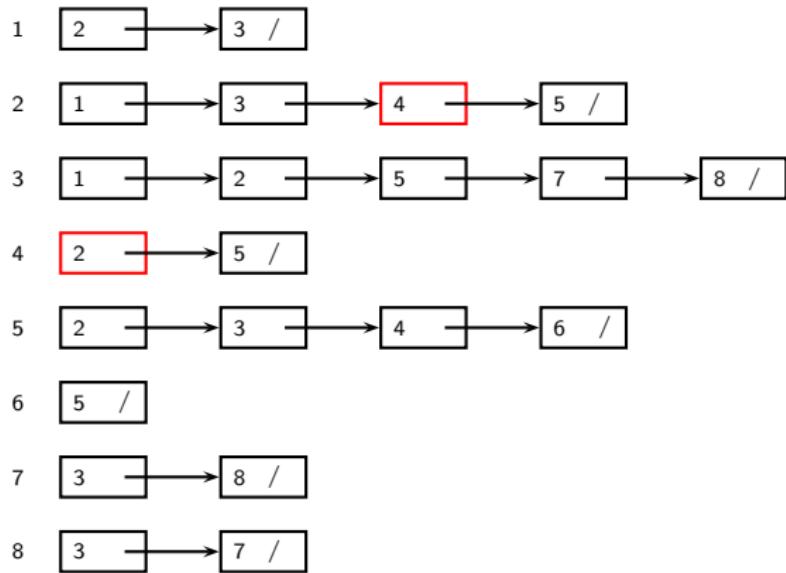
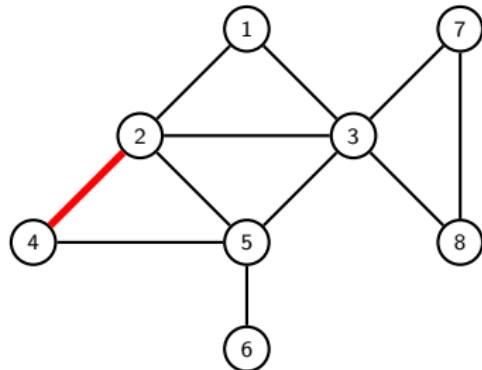
Esempio:



# Pregi e difetti della matrice di adiacenze

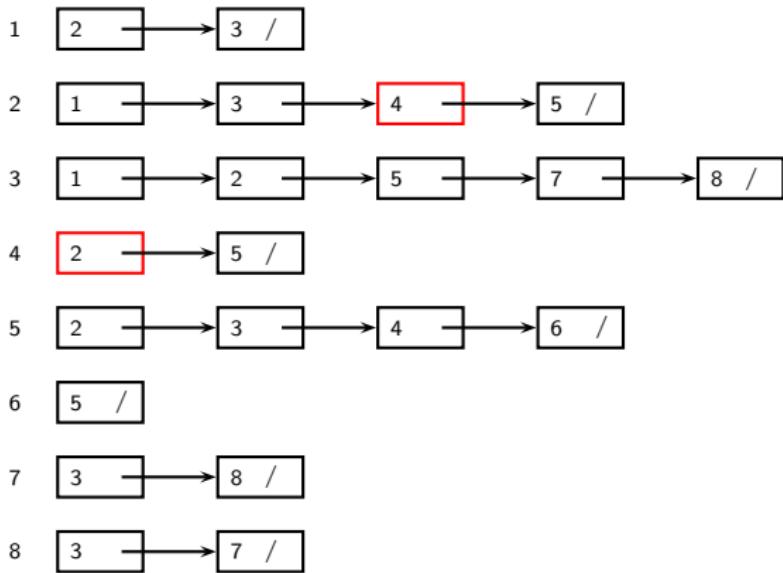
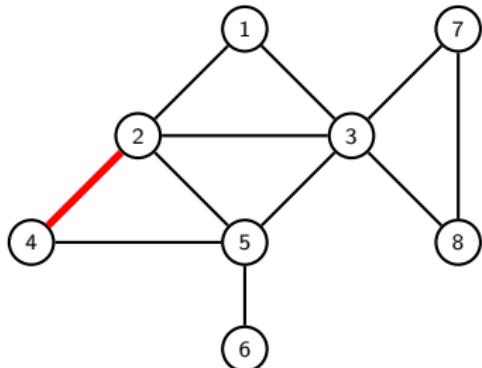


# Pregi e difetti della matrice di adiacenze



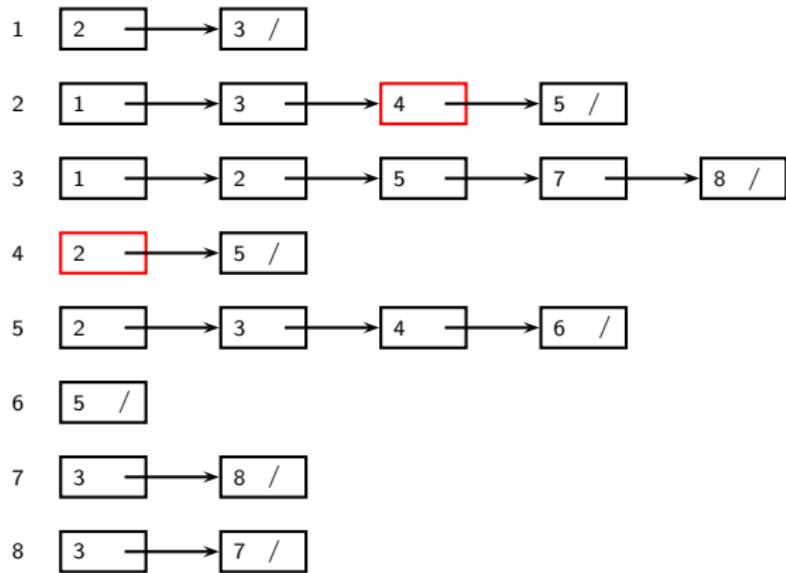
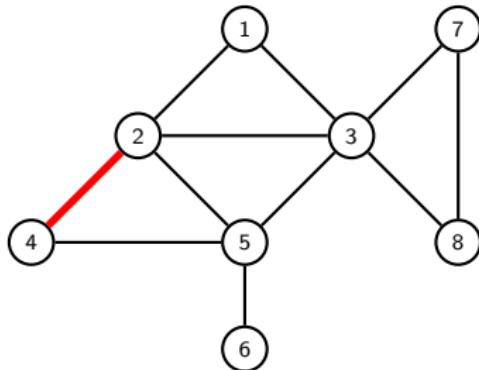
- Lo spazio richiesto è  $\Theta(n + m)$ ,  $n = |V|$ ,  $m = |E|$ ,

# Pregi e difetti della matrice di adiacenze



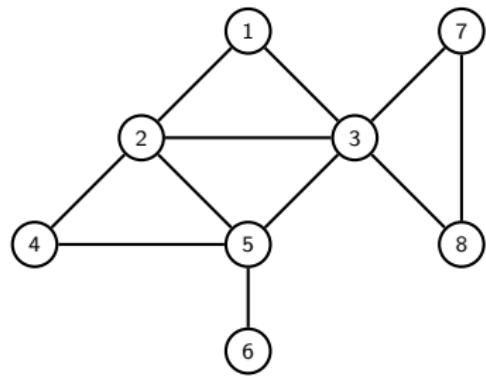
- Lo spazio richiesto è  $\Theta(n + m)$ ,  $n = |V|$ ,  $m = |E|$ ,
- Verificare se c'è un arco tra i vertici  $i$  e  $j$  prende tempo  $\Theta(\text{grado}(i))$ , dove  $\text{grado}(i) = \text{numero di nodi adiacenti a } i$ ,

# Pregi e difetti della matrice di adiacenze



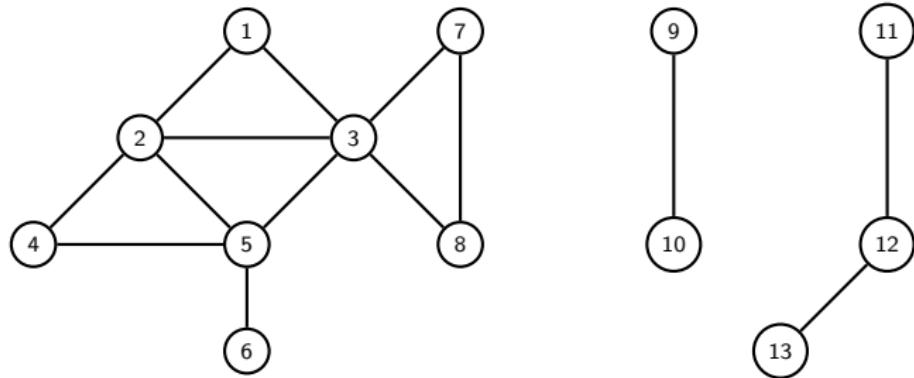
- Lo spazio richiesto è  $\Theta(n + m)$ ,  $n = |V|$ ,  $m = |E|$ ,
- Verificare se c'è un arco tra i vertici  $i$  e  $j$  prende tempo  $\Theta(\text{grado}(i))$ , dove  $\text{grado}(i) = \text{numero di nodi adiacenti a } i$ ,
- Identificare tutti gli archi del grafo prende tempo  $\Theta(n + m)$

## Un pò di nomenclatura sui grafi



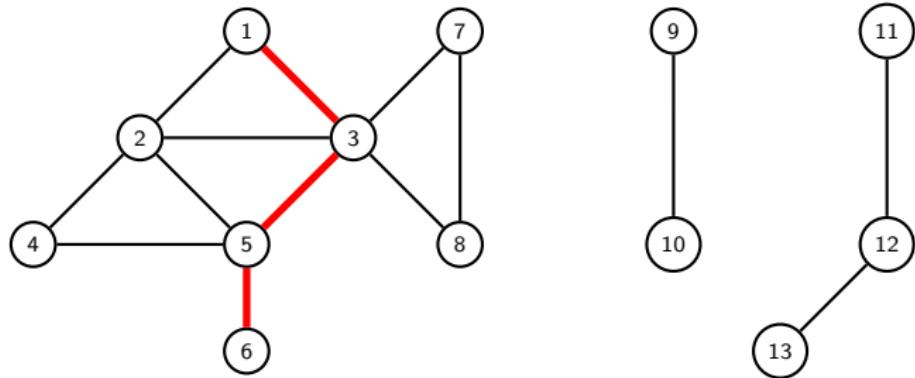
## Un pò di nomenclatura sui grafi

- ▶ Un **cammino** in un grafo  $G = (V, E)$  non diretto è una sequenza di nodi  $P = v_1 v_2 \dots v_k$  del grafo, tale  $\{v_i, v_{i+1}\} \in E$  per ogni  $i = 1, \dots, k - 1$ .



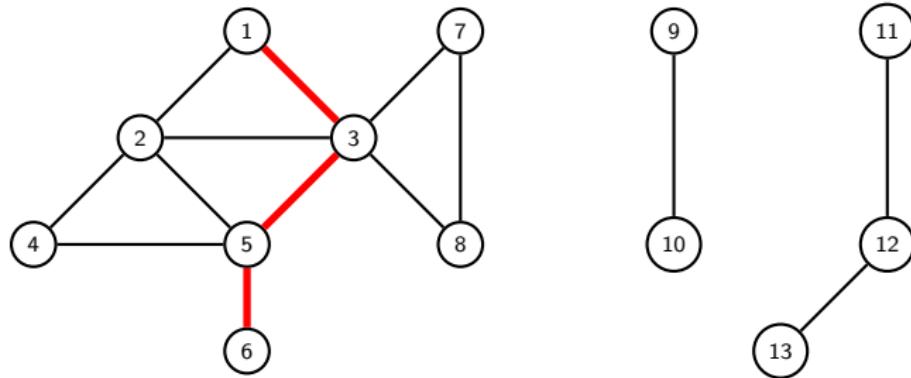
## Un pò di nomenclatura sui grafi

- ▶ Un **cammino** in un grafo  $G = (V, E)$  non diretto è una sequenza di nodi  $P = v_1 v_2 \dots v_k$  del grafo, tale  $\{v_i, v_{i+1}\} \in E$  per ogni  $i = 1, \dots, k - 1$ .



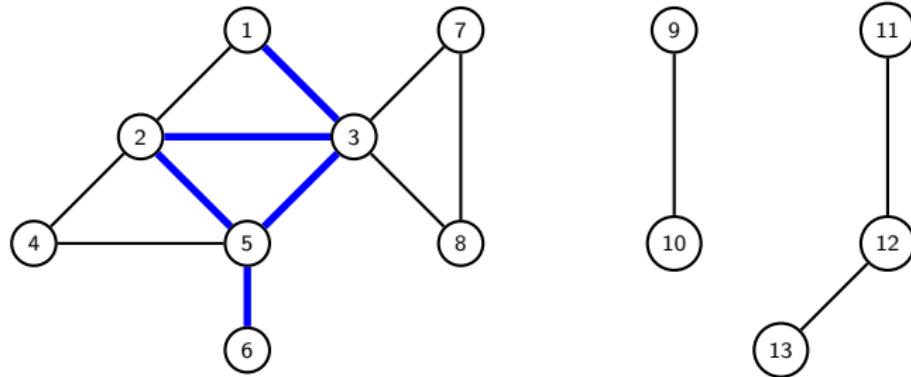
## Un pò di nomenclatura sui grafi

- ▶ Un **cammino** in un grafo  $G = (V, E)$  non diretto è una sequenza di nodi  $P = v_1 v_2 \dots v_k$  del grafo, tale  $\{v_i, v_{i+1}\} \in E$  per ogni  $i = 1, \dots, k - 1$ .
- ▶ Un cammino è **semplice** se tutti i suoi vertici sono distinti.



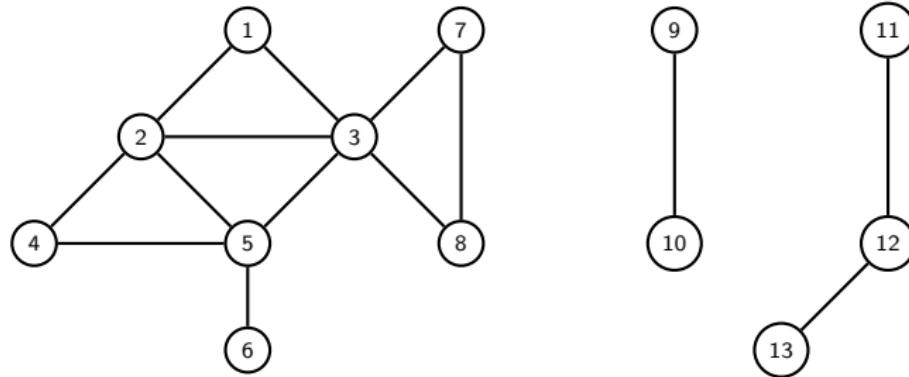
## Un pò di nomenclatura sui grafi

- ▶ Un **cammino** in un grafo  $G = (V, E)$  non diretto è una sequenza di nodi  $P = v_1 v_2 \dots v_k$  del grafo, tale  $\{v_i, v_{i+1}\} \in E$  per ogni  $i = 1, \dots, k - 1$ .
- ▶ Un cammino è **semplice** se tutti i suoi vertici sono distinti.



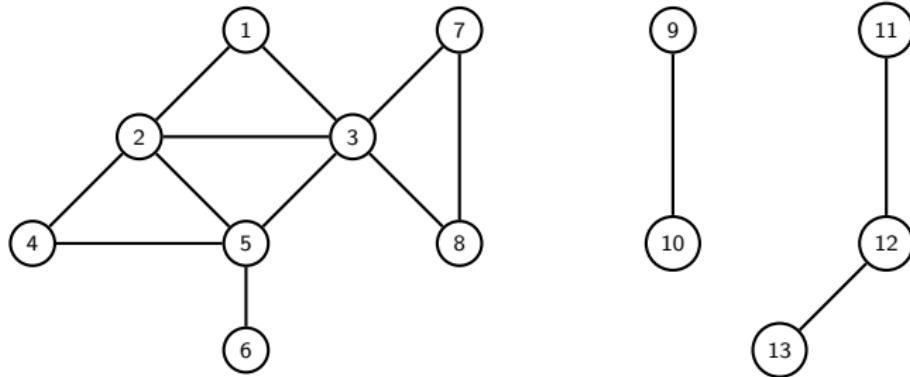
## Un pò di nomenclatura sui grafi

- ▶ Un **cammino** in un grafo  $G = (V, E)$  non diretto è una sequenza di nodi  $P = v_1 v_2 \dots v_k$  del grafo, tale  $\{v_i, v_{i+1}\} \in E$  per ogni  $i = 1, \dots, k - 1$ .
- ▶ Un cammino è **semplice** se tutti i suoi vertici sono distinti.
- ▶ Un grafo è **connesso** se per ogni coppia di vertici  $i$  e  $j$  in esso c'è un cammino tra  $i$  e  $j$ .



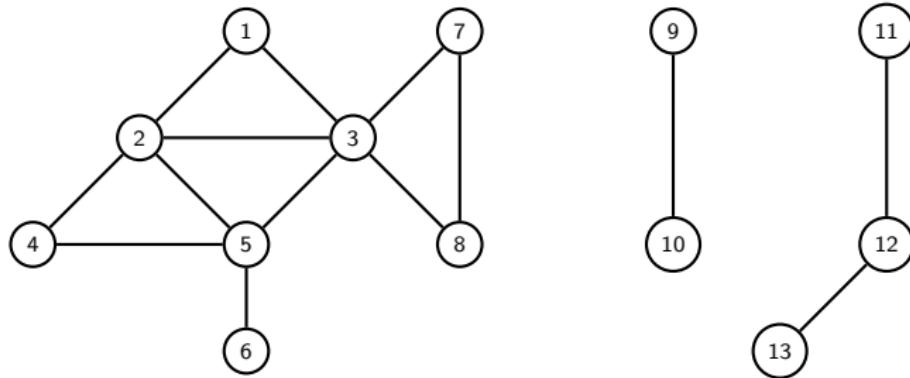
## Un pò di nomenclatura sui grafi

- ▶ Un **cammino** in un grafo  $G = (V, E)$  non diretto è una sequenza di nodi  $P = v_1 v_2 \dots v_k$  del grafo, tale  $\{v_i, v_{i+1}\} \in E$  per ogni  $i = 1, \dots, k - 1$ .
- ▶ Un cammino è **semplice** se tutti i suoi vertici sono distinti.
- ▶ Un grafo è **connesso** se per ogni coppia di vertici  $i$  e  $j$  in esso c'è un cammino tra  $i$  e  $j$ . (Es.: Il grafo in figura con insieme di vertici  $\{1, 2, \dots, 13\}$  **non** è connesso.)



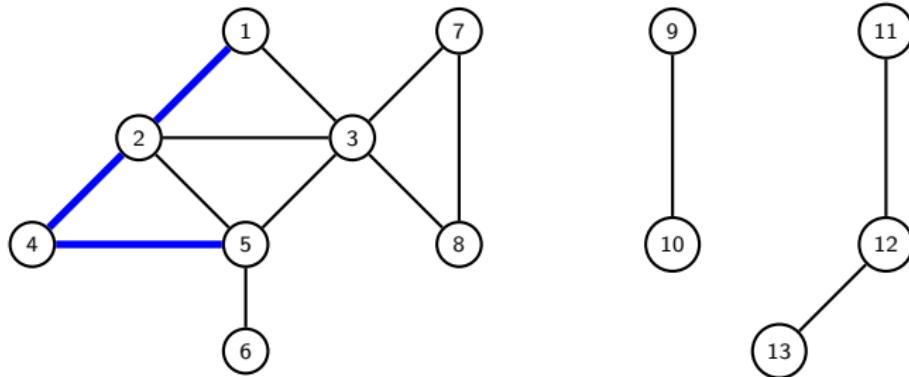
## Un pò di nomenclatura sui grafi

- ▶ Un **cammino** in un grafo  $G = (V, E)$  non diretto è una sequenza di nodi  $P = v_1 v_2 \dots v_k$  del grafo, tale  $\{v_i, v_{i+1}\} \in E$  per ogni  $i = 1, \dots, k - 1$ .
- ▶ Un cammino è **semplice** se tutti i suoi vertici sono distinti.
- ▶ Un grafo è **connesso** se per ogni coppia di vertici  $i$  e  $j$  in esso c'è un cammino tra  $i$  e  $j$ . (Es.: Il grafo in figura con insieme di vertici  $\{1, 2, \dots, 13\}$  **non** è connesso.)
- ▶ La **distanza** tra due vertici  $i$  e  $j$  è la lunghezza (misurata in numeri di archi) del più breve cammino tra  $i$  e  $j$  (se c'è).



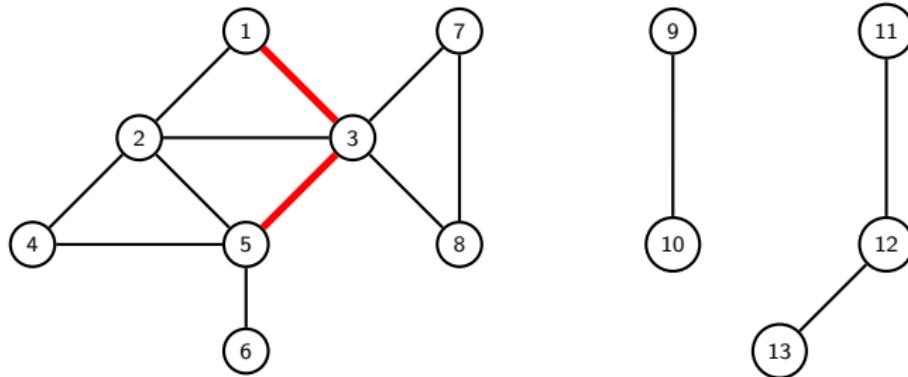
## Un pò di nomenclatura sui grafi

- ▶ Un **cammino** in un grafo  $G = (V, E)$  non diretto è una sequenza di nodi  $P = v_1 v_2 \dots v_k$  del grafo, tale  $\{v_i, v_{i+1}\} \in E$  per ogni  $i = 1, \dots, k - 1$ .
- ▶ Un cammino è **semplice** se tutti i suoi vertici sono distinti.
- ▶ Un grafo è **connesso** se per ogni coppia di vertici  $i$  e  $j$  in esso c'è un cammino tra  $i$  e  $j$ . (Es.: Il grafo in figura con insieme di vertici  $\{1, 2, \dots, 13\}$  **non** è connesso.)
- ▶ La **distanza** tra due vertici  $i$  e  $j$  è la lunghezza (misurata in numeri di archi) del più breve cammino tra  $i$  e  $j$  (se c'è).



## Un pò di nomenclatura sui grafi

- ▶ Un **cammino** in un grafo  $G = (V, E)$  non diretto è una sequenza di nodi  $P = v_1 v_2 \dots v_k$  del grafo, tale  $\{v_i, v_{i+1}\} \in E$  per ogni  $i = 1, \dots, k - 1$ .
- ▶ Un cammino è **semplice** se tutti i suoi vertici sono distinti.
- ▶ Un grafo è **connesso** se per ogni coppia di vertici  $i$  e  $j$  in esso c'è un cammino tra  $i$  e  $j$ . (Es.: Il grafo in figura con insieme di vertici  $\{1, 2, \dots, 13\}$  **non** è connesso.)
- ▶ La **distanza** tra due vertici  $i$  e  $j$  è la lunghezza (misurata in numeri di archi) del più breve cammino tra  $i$  e  $j$  (se c'è).

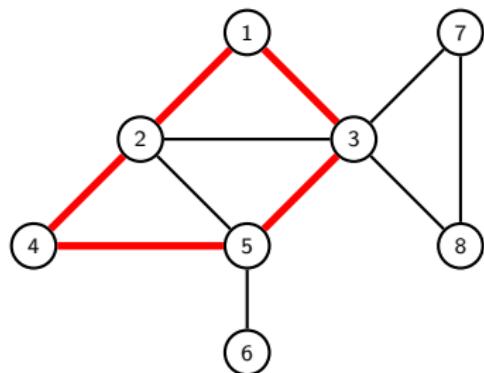


# Cicli in Grafi

- Un **ciclo** in un grafo  $G = (V, E)$  non diretto è un cammino  $P = v_1 v_2 \dots v_k$  tale  $v_1 = v_k$ ,  $k > 2$ , ed i primi  $k - 1$  nodi sono distinti.

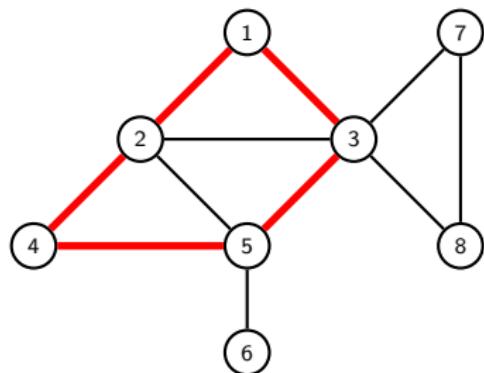
# Cicli in Grafi

- Un **ciclo** in un grafo  $G = (V, E)$  non diretto è un cammino  $P = v_1 v_2 \dots v_k$  tale  $v_1 = v_k$ ,  $k > 2$ , ed i primi  $k - 1$  nodi sono distinti.



# Cicli in Grafi

- Un **ciclo** in un grafo  $G = (V, E)$  non diretto è un cammino  $P = v_1 v_2 \dots v_k$  tale  $v_1 = v_k$ ,  $k > 2$ , ed i primi  $k - 1$  nodi sono distinti.



Ciclo=1, 2, 4, 5, 3, 1.

# Alberi

- Un grafo non diretto è detto **albero** se esso è connesso e non contiene cicli.

# Alberi

- ▶ Un grafo non diretto è detto **albero** se esso è connesso e non contiene cicli.

**Fatto:** Sia  $G$  un grafo connesso con  $n$  nodi. Ciascuna coppia delle seguenti affermazioni implica la terza (e quindi ogni coppia rappresenta una *equivalente* ed alternativa definizione di albero).

1.  $G$  è connesso.

# Alberi

- ▶ Un grafo non diretto è detto **albero** se esso è connesso e non contiene cicli.

**Fatto:** Sia  $G$  un grafo connesso con  $n$  nodi. Ciascuna coppia delle seguenti affermazioni implica la terza (e quindi ogni coppia rappresenta una *equivalente* ed alternativa definizione di albero).

1.  $G$  è connesso.
2.  $G$  non contiene cicli.

# Alberi

- ▶ Un grafo non diretto è detto **albero** se esso è connesso e non contiene cicli.

**Fatto:** Sia  $G$  un grafo connesso con  $n$  nodi. Ciascuna coppia delle seguenti affermazioni implica la terza (e quindi ogni coppia rappresenta una *equivalente* ed alternativa definizione di albero).

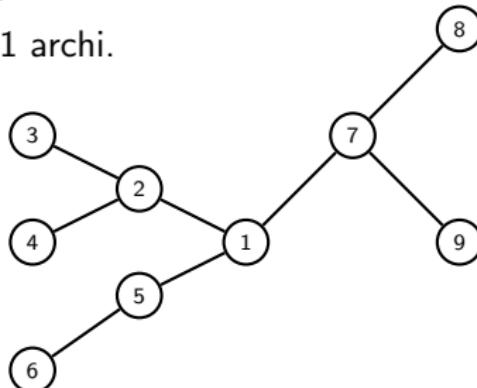
1.  $G$  è connesso.
2.  $G$  non contiene cicli.
3. ha esattamente  $n - 1$  archi.

# Alberi

- Un grafo non diretto è detto **albero** se esso è connesso e non contiene cicli.

**Fatto:** Sia  $G$  un grafo connesso con  $n$  nodi. Ciascuna coppia delle seguenti affermazioni implica la terza (e quindi ogni coppia rappresenta una *equivalente* ed alternativa definizione di albero).

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

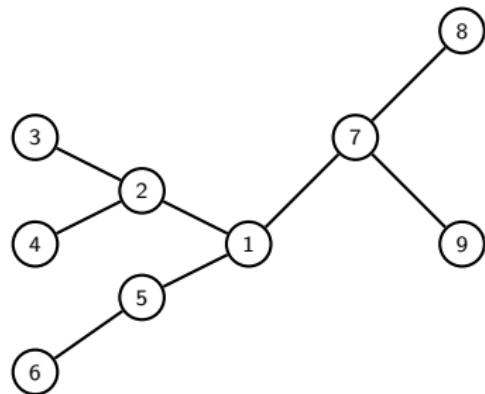


# Alberi radicati

- Dato un albero  $T$ , scegli un nodo, chiamalo radice, ed orienta gli archi “giù” dalla radice.

# Alberi radicati

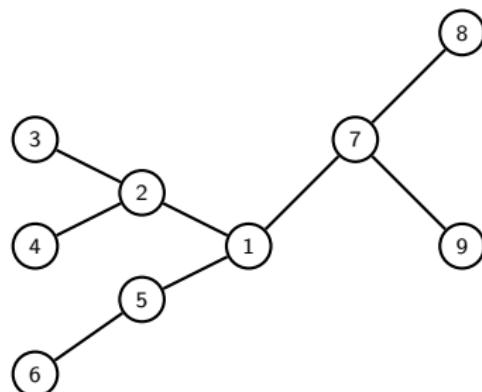
- Dato un albero  $T$ , scegli un nodo, chiamalo radice, ed orienta gli archi “giù” dalla radice.



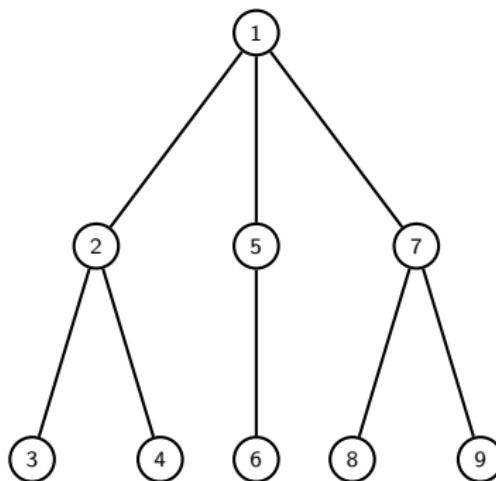
un albero

# Alberi radicati

- Dato un albero  $T$ , scegli un nodo, chiamalo radice, ed orienta gli archi “giù” dalla radice.



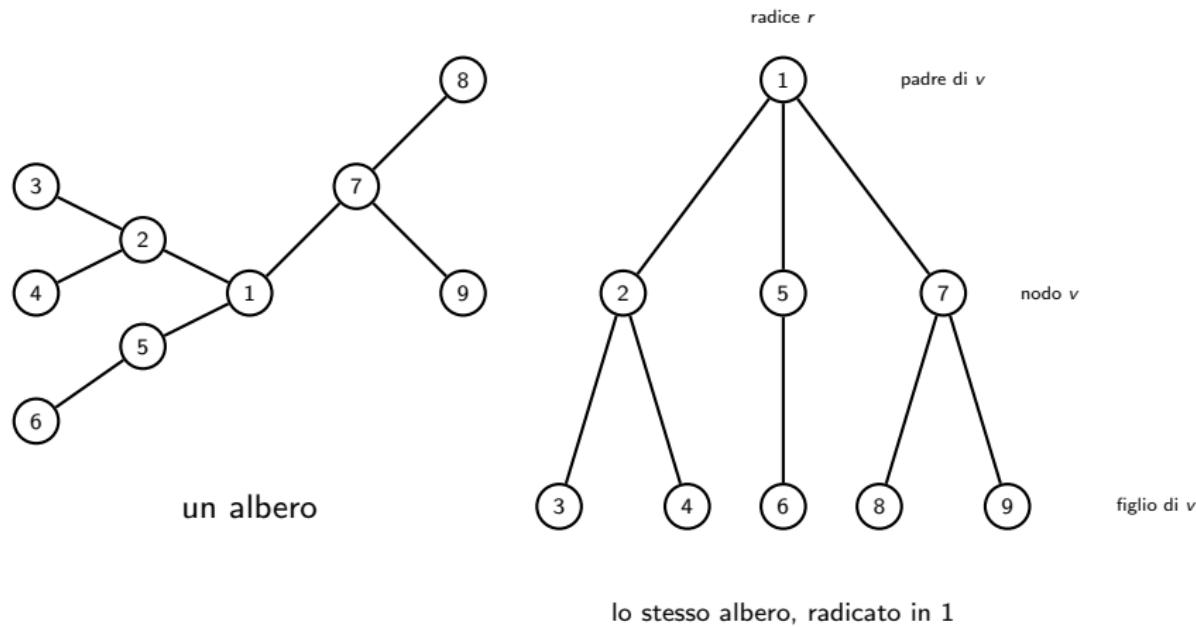
un albero



lo stesso albero, radicato in 1

# Alberi radicati

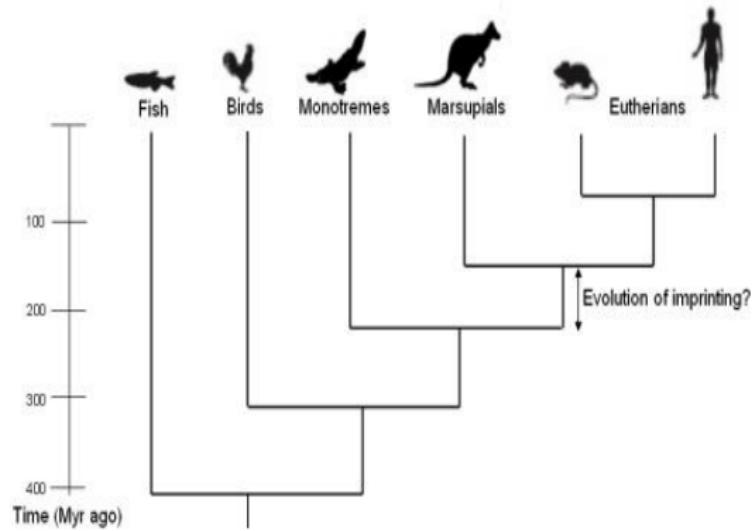
- Dato un albero  $T$ , scegli un nodo, chiamalo radice, ed orienta gli archi “giù” dalla radice.



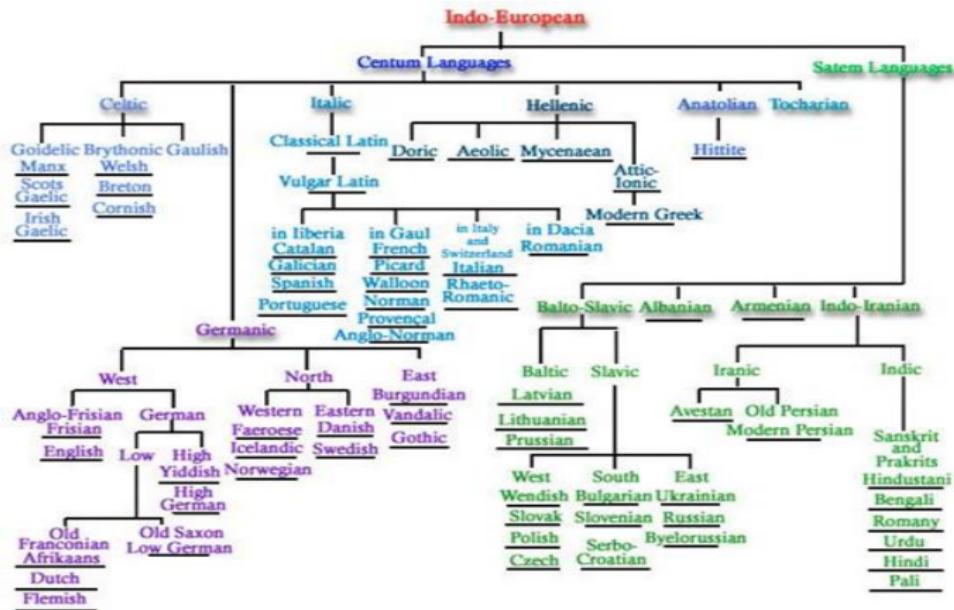
Gli alberi sono molto usati per rappresentare strutture gerarchiche

Gli alberi sono molto usati per rappresentare strutture gerarchiche

Es: **Alberi Filogenetici** per descrivere la storia evolutiva di specie



# Alberi linguistici per descrivere le relazioni tra lingue



Ora parliamo di:

Esplorazioni (altrimenti dette visite) di grafi

## Ora parliamo di:

### Esplorazioni (altrimenti dette visite) di grafi

Per esplorazione di grafi intendiamo le tecniche di attraversamento sistematico di grafi, allo scopo di acquisire informazioni sulla struttura del grafo e/o risolvere problemi algoritmici in grafi.

## Esempi di problemi algoritmici:

- ▶ Problemi di connettività di grafi

## Esempi di problemi algoritmici:

- ▶ Problemi di connettività di grafi
- ▶ **Il Problema della  $s - t$  connettività** : dati due nodi  $s$  e  $t$  in un grafo  $G$ , esiste un cammino da  $s$  a  $t$ ?

## Esempi di problemi algoritmici:

- ▶ Problemi di connettività di grafi
- ▶ Il Problema della  $s - t$  connettività : dati due nodi  $s$  e  $t$  in un grafo  $G$ , esiste un cammino da  $s$  a  $t$ ?
- ▶ Il Problema del più corto cammino da  $s - t$ : dati due nodi  $s$  e  $t$  in un grafo  $G$ , qual è la lunghezza del più corto cammino da  $s$  a  $t$ ? (ed eventualmente trovarlo).

## Esempi di problemi algoritmici:

- ▶ Problemi di connettività di grafi
- ▶ Il Problema della  $s - t$  connettività : dati due nodi  $s$  e  $t$  in un grafo  $G$ , esiste un cammino da  $s$  a  $t$ ?
- ▶ Il Problema del più corto cammino da  $s - t$ : dati due nodi  $s$  e  $t$  in un grafo  $G$ , qual è la lunghezza del più corto cammino da  $s$  a  $t$ ? (ed eventualmente trovarlo).

Le questioni hanno applicazioni in:

1. Ricerca in “reti sociali”

## Esempi di problemi algoritmici:

- ▶ Problemi di connettività di grafi
- ▶ Il Problema della  $s - t$  connettività : dati due nodi  $s$  e  $t$  in un grafo  $G$ , esiste un cammino da  $s$  a  $t$ ?
- ▶ Il Problema del più corto cammino da  $s - t$ : dati due nodi  $s$  e  $t$  in un grafo  $G$ , qual è la lunghezza del più corto cammino da  $s$  a  $t$ ? (ed eventualmente trovarlo).

Le questioni hanno applicazioni in:

1. Ricerca in “reti sociali”
2. Ricerca in spazi di soluzioni (AI)

## Esempi di problemi algoritmici:

- ▶ Problemi di connettività di grafi
- ▶ Il Problema della  $s - t$  connettività : dati due nodi  $s$  e  $t$  in un grafo  $G$ , esiste un cammino da  $s$  a  $t$ ?
- ▶ Il Problema del più corto cammino da  $s - t$ : dati due nodi  $s$  e  $t$  in un grafo  $G$ , qual è la lunghezza del più corto cammino da  $s$  a  $t$ ? (ed eventualmente trovarlo).

Le questioni hanno applicazioni in:

1. Ricerca in “reti sociali”
2. Ricerca in spazi di soluzioni (AI)
3. Percorsi di lunghezza minima in reti di comunicazione

Visita in ampiezza (Breadth First) di un grafo  $G$

## Visita in ampiezza (Breadth First) di un grafo $G$

- ▶ **Esplora** il grafo  $G$  per scoprire tutti i vertici raggiungibili dal vertice di partenza (sorgente)  $s$

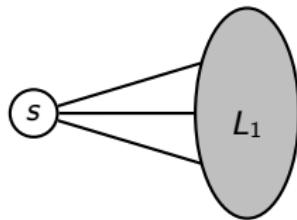
## Visita in ampiezza (Breadth First) di un grafo $G$

- ▶ **Esplora** il grafo  $G$  per scoprire tutti i vertici raggiungibili dal vertice di partenza (sorgente)  $s$
- ▶ **Calcola** la distanza (minimo numero di archi) da  $s$  ad ognuno dei vertici raggiungibili da  $s$  in  $G$

## Visita in ampiezza (Breadth First) di un grafo $G$

- ▶ **Esplora** il grafo  $G$  per scoprire tutti i vertici raggiungibili dal vertice di partenza (sorgente)  $s$
- ▶ **Calcola** la distanza (minimo numero di archi) da  $s$  ad ognuno dei vertici raggiungibili da  $s$  in  $G$

**Intuizione:** Il grafo  $G$  viene esplorato scoprendo tutti i vertici a distanza  $k$  (livello  $k$ ) prima di cominciare a scoprire quelli a distanza  $k+1$ , per  $k = 1, 2, \dots$

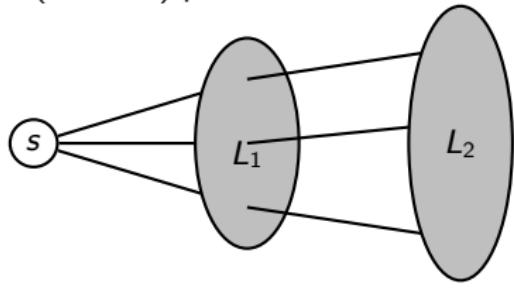


nodi a distanza 1 da  $s$

## Visita in ampiezza (Breadth First) di un grafo $G$

- ▶ **Esplora** il grafo  $G$  per scoprire tutti i vertici raggiungibili dal vertice di partenza (sorgente)  $s$
- ▶ **Calcola** la distanza (minimo numero di archi) da  $s$  ad ognuno dei vertici raggiungibili da  $s$  in  $G$

**Intuizione:** Il grafo  $G$  viene esplorato scoprendo tutti i vertici a distanza  $k$  (livello  $k$ ) prima di cominciare a scoprire quelli a distanza  $k+1$ , per  $k = 1, 2, \dots$



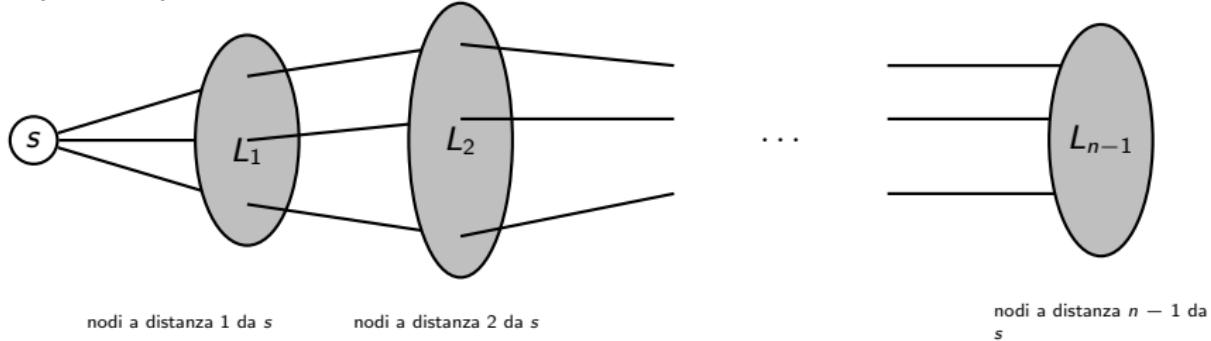
nodi a distanza 1 da  $s$

nodi a distanza 2 da  $s$

## Visita in ampiezza (Breadth First) di un grafo $G$

- ▶ **Esplora** il grafo  $G$  per scoprire tutti i vertici raggiungibili dal vertice di partenza (sorgente)  $s$
- ▶ **Calcola** la distanza (minimo numero di archi) da  $s$  ad ognuno dei vertici raggiungibili da  $s$  in  $G$

**Intuizione:** Il grafo  $G$  viene esplorato scoprendo tutti i vertici a distanza  $k$  (livello  $k$ ) prima di cominciare a scoprire quelli a distanza  $k+1$ , per  $k = 1, 2, \dots$



## BFS: l'algoritmo

$\text{BFS}(G, s)$

**1.**  $L_0 = \{s\}$

## BFS: l'algoritmo

$\text{BFS}(G, s)$

1.  $L_0 = \{s\}$
2.  $L_1 = \text{tutti i vicini di } s$

## BFS: l'algoritmo

$\text{BFS}(G, s)$

1.  $L_0 = \{s\}$

2.  $L_1 = \text{tutti i vicini di } s$

3.  $\forall i \geq 1 L_{i+1} = \text{tutti i nodi che non appartengono ad un livello } L_j \text{ precedente}$   
 $(j \leq i)$  e che sono connessi con un arco a qualche nodo in  $L_i$

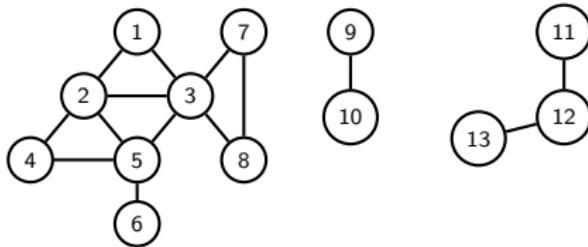
## BFS: l'algoritmo

$\text{BFS}(G, s)$

1.  $L_0 = \{s\}$

2.  $L_1 = \text{tutti i vicini di } s$

3.  $\forall i \geq 1 L_{i+1} = \text{tutti i nodi che non appartengono ad un livello } L_j \text{ precedente } (j \leq i) \text{ e che sono connessi con un arco a qualche nodo in } L_i$



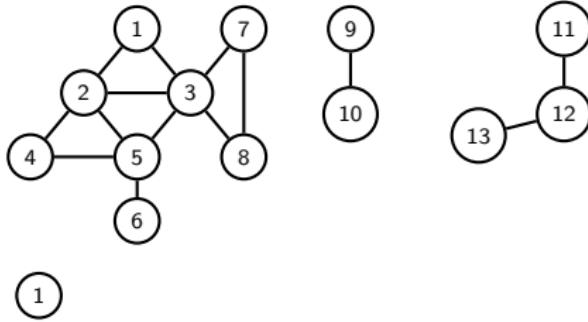
## BFS: l'algoritmo

$\text{BFS}(G, s)$

1.  $L_0 = \{s\}$

2.  $L_1 = \text{tutti i vicini di } s$

3.  $\forall i \geq 1 L_{i+1} = \text{tutti i nodi che non appartengono ad un livello } L_j \text{ precedente } (j \leq i) \text{ e che sono connessi con un arco a qualche nodo in } L_i$



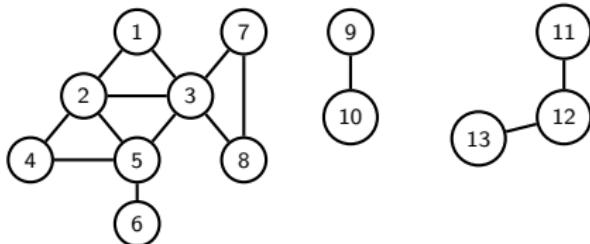
## BFS: l'algoritmo

$\text{BFS}(G, s)$

1.  $L_0 = \{s\}$

2.  $L_1 = \text{tutti i vicini di } s$

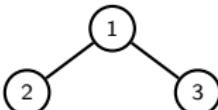
3.  $\forall i \geq 1 L_{i+1} = \text{tutti i nodi che non appartengono ad un livello } L_j \text{ precedente } (j \leq i) \text{ e che sono connessi con un arco a qualche nodo in } L_i$



$L_0$



$L_1$



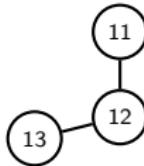
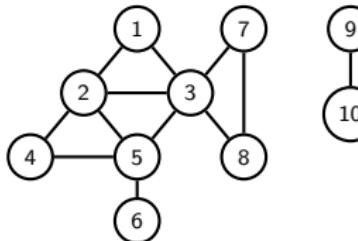
## BFS: l'algoritmo

$\text{BFS}(G, s)$

1.  $L_0 = \{s\}$

2.  $L_1 = \text{tutti i vicini di } s$

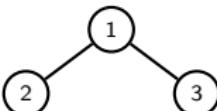
3.  $\forall i \geq 1 L_{i+1} = \text{tutti i nodi che non appartengono ad un livello } L_j \text{ precedente } (j \leq i) \text{ e che sono connessi con un arco a qualche nodo in } L_i$



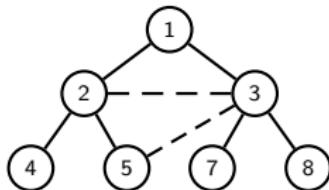
$L_0$



$L_1$



$L_2$



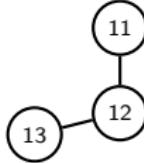
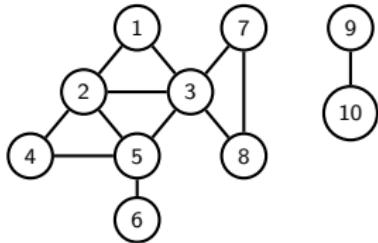
## BFS: l'algoritmo

$\text{BFS}(G, s)$

1.  $L_0 = \{s\}$

2.  $L_1 = \text{tutti i vicini di } s$

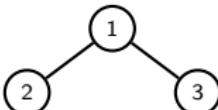
3.  $\forall i \geq 1 L_{i+1} = \text{tutti i nodi che non appartengono ad un livello } L_j \text{ precedente } (j \leq i) \text{ e che sono connessi con un arco a qualche nodo in } L_i$



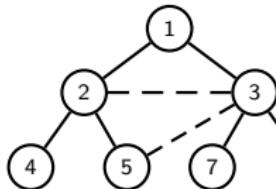
$L_0$



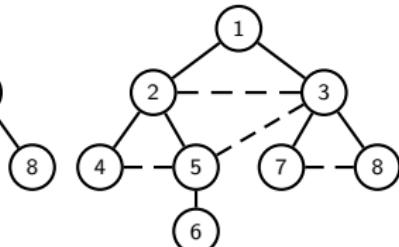
$L_1$



$L_2$



$L_3$

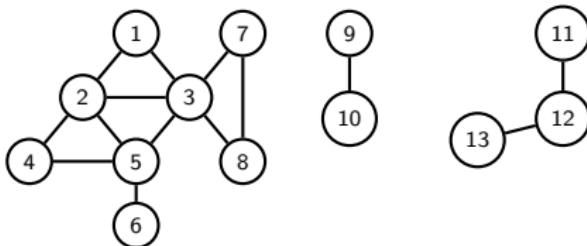


## Proprietà della BFS

**Fatto 1:** Per ciascun  $j \geq 1$ , il livello  $L_j$  di BFS consiste di tutti i nodi a distanza esattamente  $j$  da  $s$ . Esiste un cammino da  $s$  ad un nodo  $t$  se e solo se  $t$  appare in qualche livello  $L_k$

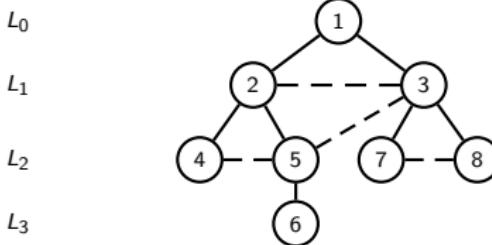
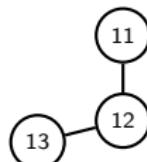
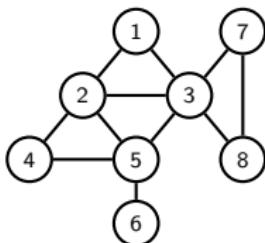
## Proprietà della BFS

**Fatto 1:** Per ciascun  $j \geq 1$ , il livello  $L_j$  di BFS consiste di tutti i nodi a distanza esattamente  $j$  da  $s$ . Esiste un cammino da  $s$  ad un nodo  $t$  se e solo se  $t$  appare in qualche livello  $L_k$



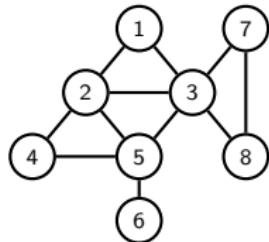
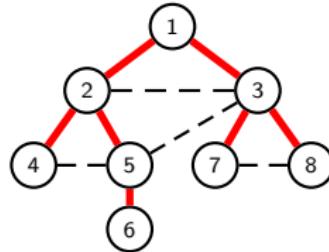
## Proprietà della BFS

**Fatto 1:** Per ciascun  $j \geq 1$ , il livello  $L_j$  di BFS consiste di tutti i nodi a distanza esattamente  $j$  da  $s$ . Esiste un cammino da  $s$  ad un nodo  $t$  se e solo se  $t$  appare in qualche livello  $L_k$

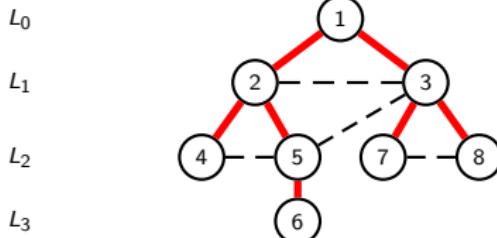
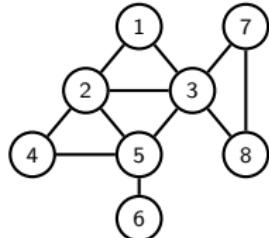


La BFS produce un albero  $T$  (albero BFS) sui nodi raggiungibili da  $s$

La BFS produce un albero  $T$  (albero BFS) sui nodi raggiungibili da  $s$

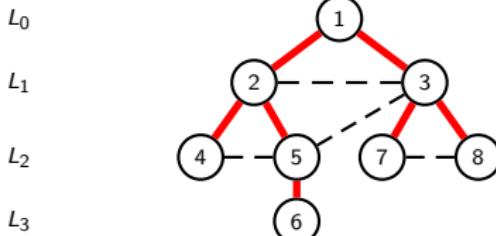
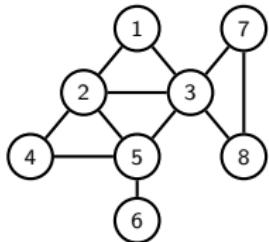
 $L_0$  $L_1$  $L_2$  $L_3$ 

La BFS produce un albero  $T$  (albero BFS) sui nodi raggiungibili da  $s$



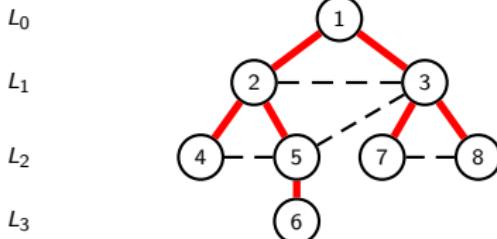
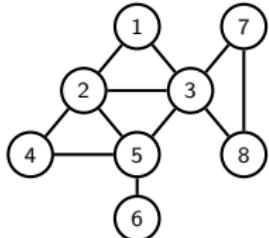
- Per ogni nodo  $v \neq s$ , consideriamo il momento in cui  $v$  è “scoperto” per la prima volta da BFS

La BFS produce un albero  $T$  (albero BFS) sui nodi raggiungibili da  $s$



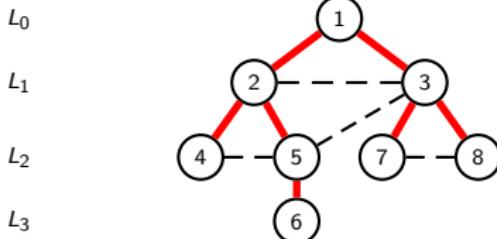
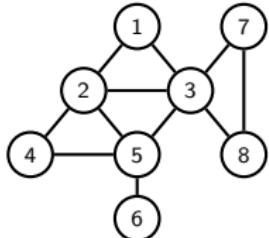
- Per ogni nodo  $v \neq s$ , consideriamo il momento in cui  $v$  è “scoperto” per la prima volta da BFS (cioè avviene quando qualche nodo  $u$  in un livello  $L_j$  viene esaminato e scopriamo che  $u$  ha un arco al nodo  $v$ ,  $v$  mai ancora visitato finora)

La BFS produce un albero  $T$  (albero BFS) sui nodi raggiungibili da  $s$



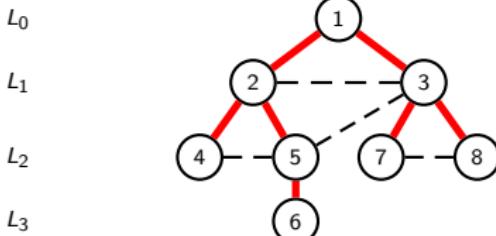
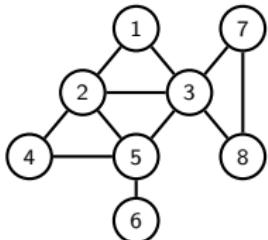
- Per ogni nodo  $v \neq s$ , consideriamo il momento in cui  $v$  è “scoperto” per la prima volta da BFS (cioè avviene quando qualche nodo  $u$  in un livello  $L_j$  viene esaminato e scopriamo che  $u$  ha un arco al nodo  $v$ ,  $v$  mai ancora visitato finora)
- In questo momento aggiungiamo l’arco  $(u, v)$  all’albero  $T$ , in modo che  $u$  diventi padre di  $v$

La BFS produce un albero  $T$  (albero BFS) sui nodi raggiungibili da  $s$



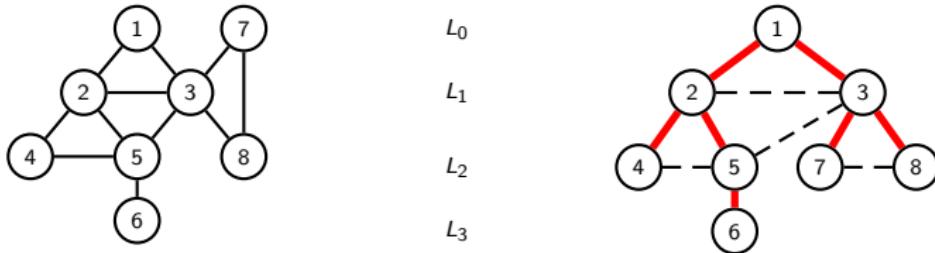
- Per ogni nodo  $v \neq s$ , consideriamo il momento in cui  $v$  è “scoperto” per la prima volta da BFS (cioè avviene quando qualche nodo  $u$  in un livello  $L_j$  viene esaminato e scopriamo che  $u$  ha un arco al nodo  $v$ ,  $v$  mai ancora visitato finora)
- In questo momento aggiungiamo l’arco  $(u, v)$  all’albero  $T$ , in modo che  $u$  diventi padre di  $v$
- Vediamo dall’esempio che ogni arco del grafo  $G$  o è anche un arco dell’albero  $T$

La BFS produce un albero  $T$  (albero BFS) sui nodi raggiungibili da  $s$



- Per ogni nodo  $v \neq s$ , consideriamo il momento in cui  $v$  è “scoperto” per la prima volta da BFS (cioè avviene quando qualche nodo  $u$  in un livello  $L_j$  viene esaminato e scopriamo che  $u$  ha un arco al nodo  $v$ ,  $v$  mai ancora visitato finora)
- In questo momento aggiungiamo l’arco  $(u, v)$  all’albero  $T$ , in modo che  $u$  diventi padre di  $v$
- Vediamo dall’esempio che ogni arco del grafo  $G$  o è anche un arco dell’albero  $T$  oppure non è un arco dell’albero e connette nodi dello stesso livello,

La BFS produce un albero  $T$  (albero BFS) sui nodi raggiungibili da  $s$



- Per ogni nodo  $v \neq s$ , consideriamo il momento in cui  $v$  è “scoperto” per la prima volta da BFS (cioè avviene quando qualche nodo  $u$  in un livello  $L_j$  viene esaminato e scopriamo che  $u$  ha un arco al nodo  $v$ ,  $v$  mai ancora visitato finora)
- In questo momento aggiungiamo l’arco  $(u, v)$  all’albero  $T$ , in modo che  $u$  diventi padre di  $v$
- Vediamo dall’esempio che ogni arco del grafo  $G$  o è anche un arco dell’albero  $T$  oppure non è un arco dell’albero e connette nodi dello stesso livello, oppure non è un arco dell’albero e connette nodi in livelli consecutivi.

Proviamolo:

**Fatto 2:** Sia  $T$  l'albero BFS sia  $x$  un nodo che appare nel livello  $L_i$  e  $y$  un nodo del livello  $L_j$ . Se  $(x, y)$  è un arco del grafo  $G$  allora o  $L_i$  e  $L_j$  sono lo stesso livello o sono due livelli *consecutivi*.

Proviamolo:

**Fatto 2:** Sia  $T$  l'albero BFS sia  $x$  un nodo che appare nel livello  $L_i$  e  $y$  un nodo del livello  $L_j$ . Se  $(x, y)$  è un arco del grafo  $G$  allora o  $L_i$  e  $L_j$  sono lo stesso livello o sono due livelli *consecutivi*.

Supponiamo per assurdo che  $i$  e  $j$  differiscano per più di 1 (ad es.  $i < j - 1$  cosicchè  $x$  viene scoperto da BFS prima di  $y$ ).

Proviamolo:

**Fatto 2:** Sia  $T$  l'albero BFS sia  $x$  un nodo che appare nel livello  $L_i$  e  $y$  un nodo del livello  $L_j$ . Se  $(x, y)$  è un arco del grafo  $G$  allora o  $L_i$  e  $L_j$  sono lo stesso livello o sono due livelli *consecutivi*.

Supponiamo per assurdo che  $i$  e  $j$  differiscano per più di 1 (ad es.  $i < j - 1$  cosicchè  $x$  viene scoperto da BFS prima di  $y$ ). Consideriamo il momento in cui gli archi incidenti su  $x$  vengono esaminati.

Proviamolo:

**Fatto 2:** Sia  $T$  l'albero BFS sia  $x$  un nodo che appare nel livello  $L_i$  e  $y$  un nodo del livello  $L_j$ . Se  $(x, y)$  è un arco del grafo  $G$  allora o  $L_i$  e  $L_j$  sono lo stesso livello o sono due livelli *consecutivi*.

Supponiamo per assurdo che  $i$  e  $j$  differiscano per più di 1 (ad es.  $i < j - 1$  cosicchè  $x$  viene scoperto da BFS prima di  $y$ ). Consideriamo il momento in cui gli archi incidenti su  $x$  vengono esaminati. Poichè  $x \in L_i$ , i nodi connessi ad  $x$  da un arco e che possiamo quindi scoprire a partire da  $x$  o sono al livello  $L_{i+1}$

Proviamolo:

**Fatto 2:** Sia  $T$  l'albero BFS sia  $x$  un nodo che appare nel livello  $L_i$  e  $y$  un nodo del livello  $L_j$ . Se  $(x, y)$  è un arco del grafo  $G$  allora o  $L_i$  e  $L_j$  sono lo stesso livello o sono due livelli *consecutivi*.

Supponiamo per assurdo che  $i$  e  $j$  differiscano per più di 1 (ad es.  $i < j - 1$  cosicchè  $x$  viene scoperto da BFS prima di  $y$ ). Consideriamo il momento in cui gli archi incidenti su  $x$  vengono esaminati. Poichè  $x \in L_i$ , i nodi connessi ad  $x$  da un arco e che possiamo quindi scoprire a partire da  $x$  o sono al livello  $L_{i+1}$  o sono già stati scoperti prima e quindi sono nel livello  $L_i$ .

Proviamolo:

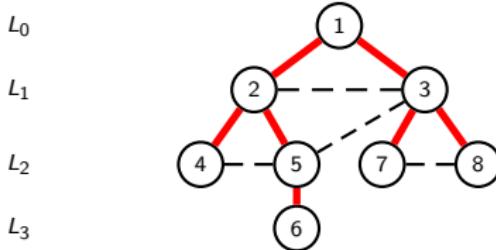
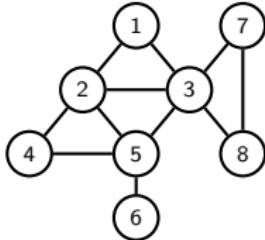
**Fatto 2:** Sia  $T$  l'albero BFS sia  $x$  un nodo che appare nel livello  $L_i$  e  $y$  un nodo del livello  $L_j$ . Se  $(x, y)$  è un arco del grafo  $G$  allora o  $L_i$  e  $L_j$  sono lo stesso livello o sono due livelli *consecutivi*.

Supponiamo per assurdo che  $i$  e  $j$  differiscano per più di 1 (ad es.  $i < j - 1$  cosicchè  $x$  viene scoperto da BFS prima di  $y$ ). Consideriamo il momento in cui gli archi incidenti su  $x$  vengono esaminati. Poichè  $x \in L_i$ , i nodi connessi ad  $x$  da un arco e che possiamo quindi scoprire a partire da  $x$  o sono al livello  $L_{i+1}$  o sono già stati scoperti prima e quindi sono nel livello  $L_i$ . Pertanto  $y \in L_j$ , con  $j \in \{i, i + 1\}$

Proviamolo:

**Fatto 2:** Sia  $T$  l'albero BFS sia  $x$  un nodo che appare nel livello  $L_i$  e  $y$  un nodo del livello  $L_j$ . Se  $(x, y)$  è un arco del grafo  $G$  allora o  $L_i$  e  $L_j$  sono lo stesso livello o sono due livelli *consecutivi*.

Supponiamo per assurdo che  $i$  e  $j$  differiscano per più di 1 (ad es.  $i < j - 1$  cosicchè  $x$  viene scoperto da BFS prima di  $y$ ). Consideriamo il momento in cui gli archi incidenti su  $x$  vengono esaminati. Poichè  $x \in L_i$ , i nodi connessi ad  $x$  da un arco e che possiamo quindi scoprire a partire da  $x$  o sono al livello  $L_{i+1}$  o sono già stati scoperti prima e quindi sono nel livello  $L_i$ . Pertanto  $y \in L_j$ , con  $j \in \{i, i + 1\}$



## Applicazione di BFS: Componenti connesse

## Applicazione di BFS: Componenti connesse

- ▶ Dato  $G = (V, E)$ ,  $A \subseteq V$  è una *componente连通的* di  $G$  se per ogni  $u, v \in A$ , esiste in  $G$  un cammino tra  $u$  e  $v$ .

## Applicazione di BFS: Componenti connesse

- ▶ Dato  $G = (V, E)$ ,  $A \subseteq V$  è una *componente连通的* di  $G$  se per ogni  $u, v \in A$ , esiste in  $G$  un cammino tra  $u$  e  $v$ .
- ▶ Conoscere le componenti connesse di un grafo è **importante** per molte applicazioni, ad es. ci permette di risolvere uno dei problemi menzionati all'inizio della lezione: *Dati i nodi  $u$  e  $v$ , esiste un cammino nel grafo tra  $u$  e  $v$ ?*

## Applicazione di BFS: Componenti connesse

- ▶ Dato  $G = (V, E)$ ,  $A \subseteq V$  è una *componente连通的* di  $G$  se per ogni  $u, v \in A$ , esiste in  $G$  un cammino tra  $u$  e  $v$ .
- ▶ Conoscere le componenti connesse di un grafo è **importante** per molte applicazioni, ad es. ci permette di risolvere uno dei problemi menzionati all'inizio della lezione: *Dati i nodi  $u$  e  $v$ , esiste un cammino nel grafo tra  $u$  e  $v$ ?*
- ▶ Per rispondere a tale domanda, basta controllare se  $u$  e  $v$  appartengono alla stessa componente connessa.

## Applicazione di BFS: Componenti connesse

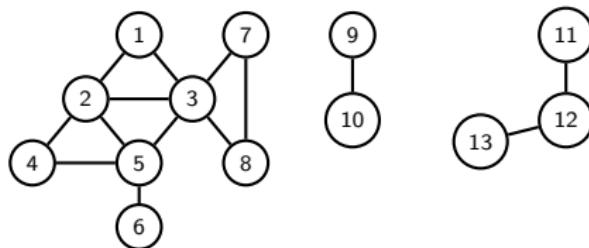
- ▶ Dato  $G = (V, E)$ ,  $A \subseteq V$  è una *componente连通的* di  $G$  se per ogni  $u, v \in A$ , esiste in  $G$  un cammino tra  $u$  e  $v$ .
- ▶ Conoscere le componenti connesse di un grafo è **importante** per molte applicazioni, ad es. ci permette di risolvere uno dei problemi menzionati all'inizio della lezione: *Dati i nodi  $u$  e  $v$ , esiste un cammino nel grafo tra  $u$  e  $v$ ?*
- ▶ Per rispondere a tale domanda, basta controllare se  $u$  e  $v$  appartengono alla stessa componente connessa.

L'insieme dei nodi dell'albero prodotto dalla BFS con partenza da un nodo  $s$  coincide con la componente connessa che contiene  $s$ .

## Applicazione di BFS: Componenti connesse

- ▶ Dato  $G = (V, E)$ ,  $A \subseteq V$  è una *componente connessa* di  $G$  se per ogni  $u, v \in A$ , esiste in  $G$  un cammino tra  $u$  e  $v$ .
- ▶ Conoscere le componenti connesse di un grafo è **importante** per molte applicazioni, ad es. ci permette di risolvere uno dei problemi menzionati all'inizio della lezione: *Dati i nodi  $u$  e  $v$ , esiste un cammino nel grafo tra  $u$  e  $v$ ?*
- ▶ Per rispondere a tale domanda, basta controllare se  $u$  e  $v$  appartengono alla stessa componente connessa.

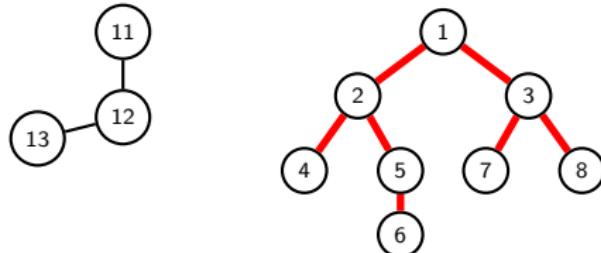
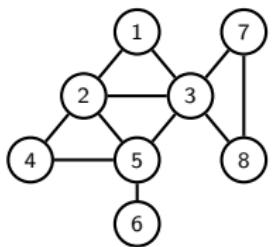
L'insieme dei nodi dell'albero prodotto dalla BFS con partenza da un nodo  $s$  coincide con la componente connessa che contiene  $s$ .



## Applicazione di BFS: Componenti connesse

- ▶ Dato  $G = (V, E)$ ,  $A \subseteq V$  è una *componente连通* di  $G$  se per ogni  $u, v \in A$ , esiste in  $G$  un cammino tra  $u$  e  $v$ .
- ▶ Conoscere le componenti connesse di un grafo è **importante** per molte applicazioni, ad es. ci permette di risolvere uno dei problemi menzionati all'inizio della lezione: *Dati i nodi  $u$  e  $v$ , esiste un cammino nel grafo tra  $u$  e  $v$ ?*
- ▶ Per rispondere a tale domanda, basta controllare se  $u$  e  $v$  appartengono alla stessa componente connessa.

L'insieme dei nodi dell'albero prodotto dalla BFS con partenza da un nodo  $s$  coincide con la componente connessa che contiene  $s$ .



In particolare....

**Fatto:** Possiamo determinare se un grafo  $G = (V, E)$  è connesso (ovvero esiste un cammino in  $G$  tra ogni coppia di suoi nodi, ovvero consta **di una sola** componente连通的) semplicemente eseguendo la visita BFS( $s$ ) a partire da un nodo arbitrario  $s$ .

In particolare....

**Fatto:** Possiamo determinare se un grafo  $G = (V, E)$  è connesso (ovvero esiste un cammino in  $G$  tra ogni coppia di suoi nodi, ovvero consta **di una sola** componente连通的) semplicemente eseguendo la visita BFS( $s$ ) a partire da un nodo arbitrario  $s$ . Se alla fine dell'algoritmo abbiamo visitato tutti i nodi di  $G$  allora otteniamo che  $G$  è connesso, altrimenti  $G$  **non** è connesso.

E perchè mai vogliamo determinare le componenti connesse di un grafo?

E perchè mai vogliamo determinare le componenti connesse di un grafo?

- ▶ Se due vertici sono connesse da un arco significa che le corrispondenti entità sono “simili”, allora le componenti connesse corrispondono a differenti gruppi di entità

E perchè mai vogliamo determinare le componenti connesse di un grafo?

- ▶ Se due vertici sono connesse da un arco significa che le corrispondenti entità sono “simili”, allora le componenti connesse corrispondono a differenti gruppi di entità
- ▶ Le componenti connesse in una rete sociale vengono usate per identificare differenti comunità che condividono caratteristiche comuni

E perchè mai vogliamo determinare le componenti connesse di un grafo?

- ▶ Se due vertici sono connesse da un arco significa che le corrispondenti entità sono “simili”, allora le componenti connesse corrispondono a differenti gruppi di entità
- ▶ Le componenti connesse in una rete sociale vengono usate per identificare differenti comunità che condividono caratteristiche comuni
- ▶ ...

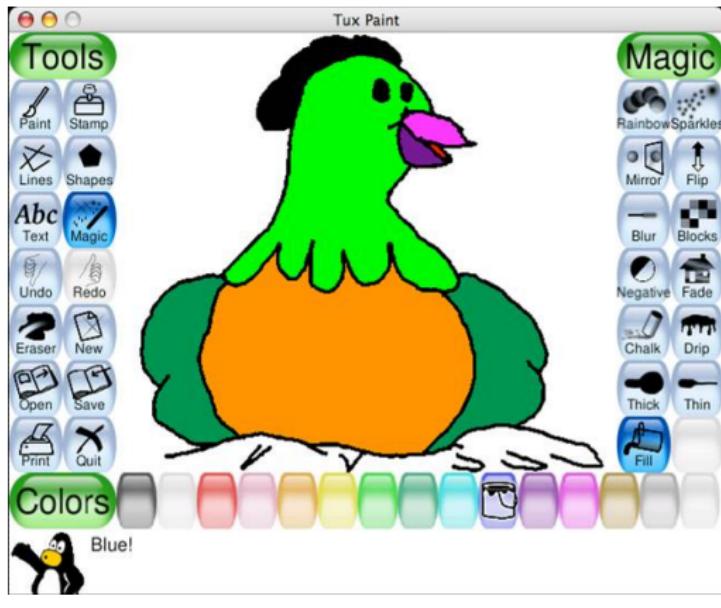
Riempimento Aree: data una regione di pixel contigui in una figura, cambiare in blocco il colore di essi (ad es. da **verde** a **blu**).

Riempimento Aree: data una regione di pixel contigui in una figura, cambiare in blocco il colore di essi (ad es. da **verde** a **blu**).

Interpretiamo i pixel come nodi di un grafo, in cui due nodi sono uniti da un arco se i corrispondenti pixel sono vicini nella figura, e quindi la regione di pixel contigui corrisponde naturalmente ad una componente连通的 del grafo

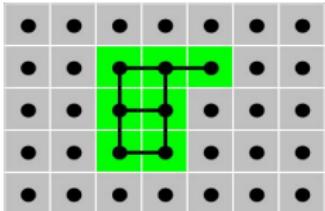
Riempimento Aree: data una regione di pixel contigui in una figura, cambiare in blocco il colore di essi (ad es. da **verde** a **blu**).

Interpretiamo i pixel come nodi di un grafo, in cui due nodi sono uniti da un arco se i corrispondenti pixel sono vicini nella figura, e quindi la regione di pixel contigui corrisponde naturalmente ad una componente连通的 del grafo



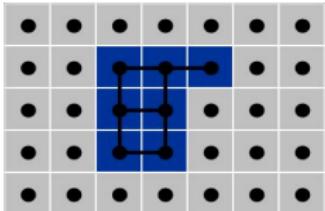
Riempimento Aree: data una regione di pixel contigui in una figura, cambiare in blocco il colore di essi (ad es. da **verde** a **blu**).

Interpretiamo i pixel come nodi di un grafo, in cui due nodi sono uniti da un arco se i corrispondenti pixel sono vicini nella figura, e quindi la regione di pixel contigui corrisponde naturalmente ad una componente连通的 del grafo



Riempimento Aree: data una regione di pixel contigui in una figura, cambiare in blocco il colore di essi (ad es. da **verde** a **blu**).

Interpretiamo i pixel come nodi di un grafo, in cui due nodi sono uniti da un arco se i corrispondenti pixel sono vicini nella figura, e quindi la regione di pixel contigui corrisponde naturalmente ad una componente连通的 del grafo



“Implementazione” dell’algoritmo  $\text{BFS}(G, s)$ .

“Implementazione” dell’algoritmo  $\text{BFS}(G, s)$ .

Ricordiamo la rappresentazione di grafi mediante liste di adiacenze:

“Implementazione” dell’algoritmo  $\text{BFS}(G, s)$ .

Ricordiamo la rappresentazione di grafi mediante liste di adiacenze:

- ▶  $n$  liste a puntatori, una per ogni nodo del grafo

“Implementazione” dell’algoritmo  $\text{BFS}(G, s)$ .

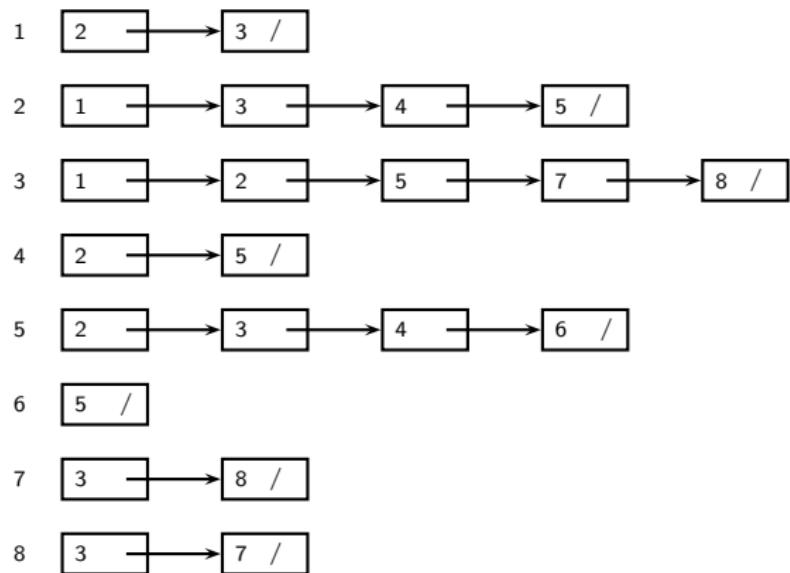
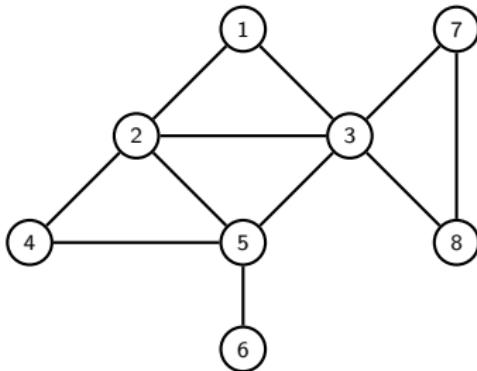
Ricordiamo la rappresentazione di grafi mediante liste di adiacenze:

- ▶  $n$  liste a puntatori, una per ogni nodo del grafo
- ▶ La lista associata al generico nodo  $i$  contiene *tutti* i nodi  $j$  vicini ad  $i$  nel grafo.

“Implementazione” dell’algoritmo  $\text{BFS}(G, s)$ .

Ricordiamo la rappresentazione di grafi mediante liste di adiacenze:

- ▶  $n$  liste a puntatori, una per ogni nodo del grafo
- ▶ La lista associata al generico nodo  $i$  contiene *tutti* i nodi  $j$  vicini ad  $i$  nel grafo.



- ▶ L'algoritmo  $\text{BFS}(G, s)$  costruisce i livelli di nodi  $L_1, L_2, \dots$ , dove  $L_i =$ insieme dei nodi a distanza  $i$  da  $s$

- ▶ L'algoritmo  $\text{BFS}(G, s)$  costruisce i livelli di nodi  $L_1, L_2, \dots$ , dove  $L_i =$ insieme dei nodi a distanza  $i$  da  $s$
- ▶ Per rappresentare ogni livello  $L_i$  useremo una *lista*  $L[i]$

- ▶ L'algoritmo  $\text{BFS}(G, s)$  costruisce i livelli di nodi  $L_1, L_2, \dots$ , dove  $L_i =$ insieme dei nodi a distanza  $i$  da  $s$
- ▶ Per rappresentare ogni livello  $L_i$  useremo una *lista*  $L[i]$
- ▶ Quando esaminiamo un nodo  $u \in L_i$ , esamineremo tutti i gli archi della forma  $(u, v)$  e se  $v \notin L_i$  allora occorrerà mettere  $v$  in  $L_{i+1}$ , quindi ci occorre un metodo veloce per stabilire questo fatto

- ▶ L'algoritmo  $\text{BFS}(G, s)$  costruisce i livelli di nodi  $L_1, L_2, \dots$ , dove  $L_i =$ insieme dei nodi a distanza  $i$  da  $s$
- ▶ Per rappresentare ogni livello  $L_i$  useremo una *lista*  $L[i]$
- ▶ Quando esaminiamo un nodo  $u \in L_i$ , esamineremo tutti i gli archi della forma  $(u, v)$  e se  $v \notin L_i$  allora occorrerà mettere  $v$  in  $L_{i+1}$ , quindi ci occorre un metodo veloce per stabilire questo fatto
- ▶ Useremo un array  $\text{Scoperto}[1 \dots n]$  e porremo  $\text{Scoperto}[i] = \text{true}$  ogni qualvolta la nostra visita trova il vertice  $i$ .

- ▶ L'algoritmo  $\text{BFS}(G, s)$  costruisce i livelli di nodi  $L_1, L_2, \dots$ , dove  $L_i =$ insieme dei nodi a distanza  $i$  da  $s$
- ▶ Per rappresentare ogni livello  $L_i$  useremo una *lista*  $L[i]$
- ▶ Quando esaminiamo un nodo  $u \in L_i$ , esamineremo tutti i gli archi della forma  $(u, v)$  e se  $v \notin L_i$  allora occorrerà mettere  $v$  in  $L_{i+1}$ , quindi ci occorre un metodo veloce per stabilire questo fatto
- ▶ Useremo un array  $\text{Scoperto}[1 \dots n]$  e porremo  $\text{Scoperto}[i] = \text{true}$  ogni qualvolta la nostra visita trova il vertice  $i$ .
- ▶ Per ogni nodo  $v$  calcoleremo un parametro  $d[v]$  e porremo  $d[v] = i$  se e solo se il nodo  $i$  viene messo nel livello  $L_i$  (ovvero se il nodo  $i$  si trova esattamente a distanza  $i$  dal nodo di partenza  $s$ )

$\text{BFS}(G, s)$

$\text{BFS}(G, s)$

1. Poni  $\text{Scoperto}[s] = \text{true}$  e  $\text{Scoperto}[v] = \text{false} \ \forall v \neq s$

$\text{BFS}(G, s)$

1. Poni  $\text{Scoperto}[s] = \text{true}$  e  $\text{Scoperto}[v] = \text{false} \quad \forall v \neq s$
2. Inizializza  $L[0]$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$

$\text{BFS}(G, s)$

1. Poni  $\text{Scoperto}[s] = \text{true}$  e  $\text{Scoperto}[v] = \text{false} \ \forall v \neq s$
2. Inizializza  $L[0]$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Poni il contatore dei livelli  $i$  a 0

$\text{BFS}(G, s)$

1. Poni  $\text{Scoperto}[s] = \text{true}$  e  $\text{Scoperto}[v] = \text{false} \ \forall v \neq s$
2. Inizializza  $L[0]$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Poni il contatore dei livelli  $i$  a 0
4. Inizializza l'albero BFS  $T$  a  $\emptyset$

$\text{BFS}(G, s)$

1. Poni  $\text{Scoperto}[s] = \text{true}$  e  $\text{Scoperto}[v] = \text{false} \ \forall v \neq s$
2. Inizializza  $L[0]$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Poni il contatore dei livelli  $i$  a 0
4. Inizializza l'albero BFS  $T$  a  $\emptyset$
5. While  $L[i]$  non è vuota

$\text{BFS}(G, s)$

1. Poni  $\text{Scoperto}[s] = \text{true}$  e  $\text{Scoperto}[v] = \text{false} \ \forall v \neq s$
2. Inizializza  $L[0]$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Poni il contatore dei livelli  $i$  a 0
4. Inizializza l'albero BFS  $T$  a  $\emptyset$
5. While  $L[i]$  non è vuota
6. Inizializza una lista vuota  $L[i + 1]$

$\text{BFS}(G, s)$

1. Poni  $\text{Scoperto}[s] = \text{true}$  e  $\text{Scoperto}[v] = \text{false} \ \forall v \neq s$
2. Inizializza  $L[0]$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Poni il contatore dei livelli  $i$  a 0
4. Inizializza l'albero BFS  $T$  a  $\emptyset$
5. While  $L[i]$  non è vuota
6. Inizializza una lista vuota  $L[i + 1]$
7. For ogni nodo  $u \in L[i]$

## BFS( $G, s$ )

1. Poni Scoperto[ $s$ ] =true e Scoperto[ $v$ ] =false  $\forall v \neq s$
2. Inizializza  $L[0]$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Poni il contatore dei livelli  $i$  a 0
4. Inizializza l'albero BFS  $T$  a  $\emptyset$
5. While  $L[i]$  non è vuota
6. Inizializza una lista vuota  $L[i + 1]$
7. For ogni nodo  $u \in L[i]$
8. Estrai  $u$  da  $L[i]$

## BFS( $G, s$ )

1. Poni Scoperto[ $s$ ] =true e Scoperto[ $v$ ] =false  $\forall v \neq s$
2. Inizializza  $L[0]$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Poni il contatore dei livelli  $i$  a 0
4. Inizializza l'albero BFS  $T$  a  $\emptyset$
5. While  $L[i]$  non è vuota
  - 6. Inizializza una lista vuota  $L[i + 1]$
  - 7. For ogni nodo  $u \in L[i]$ 
    - 8. Estrai  $u$  da  $L[i]$
    - 9. For ogni arco  $(u, v)$  incidente su  $u$

## BFS( $G, s$ )

1. Poni Scoperto[ $s$ ] =true e Scoperto[ $v$ ] =false  $\forall v \neq s$
2. Inizializza  $L[0]$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Poni il contatore dei livelli  $i$  a 0
4. Inizializza l'albero BFS  $T$  a  $\emptyset$
5. While  $L[i]$  non è vuota
  - 6. Inizializza una lista vuota  $L[i + 1]$
  - 7. For ogni nodo  $u \in L[i]$ 
    - 8. Estrai  $u$  da  $L[i]$
    - 9. For ogni arco  $(u, v)$  incidente su  $u$ 
      - 10. If Scoperto[ $v$ ]=false then

## BFS( $G, s$ )

1. Poni Scoperto[ $s$ ] =true e Scoperto[ $v$ ] =false  $\forall v \neq s$
2. Inizializza  $L[0]$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Poni il contatore dei livelli  $i$  a 0
4. Inizializza l'albero BFS  $T$  a  $\emptyset$
5. While  $L[i]$  non è vuota
  - 6. Inizializza una lista vuota  $L[i + 1]$
  - 7. For ogni nodo  $u \in L[i]$ 
    - 8. Estrai  $u$  da  $L[i]$
    - 9. For ogni arco  $(u, v)$  incidente su  $u$ 
      - 10. If Scoperto[ $v$ ]=false then
        - 11. Poni Scoperto[ $v$ ]=true

## BFS( $G, s$ )

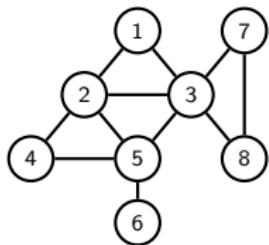
1. Poni Scoperto[ $s$ ] =true e Scoperto[ $v$ ] =false  $\forall v \neq s$
2. Inizializza  $L[0]$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Poni il contatore dei livelli  $i$  a 0
4. Inizializza l'albero BFS  $T$  a  $\emptyset$
5. While  $L[i]$  non è vuota
  - 6. Inizializza una lista vuota  $L[i + 1]$
  - 7. For ogni nodo  $u \in L[i]$ 
    - 8. Estrai  $u$  da  $L[i]$
    - 9. For ogni arco  $(u, v)$  incidente su  $u$ 
      - 10. If Scoperto[ $v$ ]=false then
        - 11. Poni Scoperto[ $v$ ]=true
        - 12. Aggiungi l'arco  $(u, v)$  all'albero  $T$

## BFS( $G, s$ )

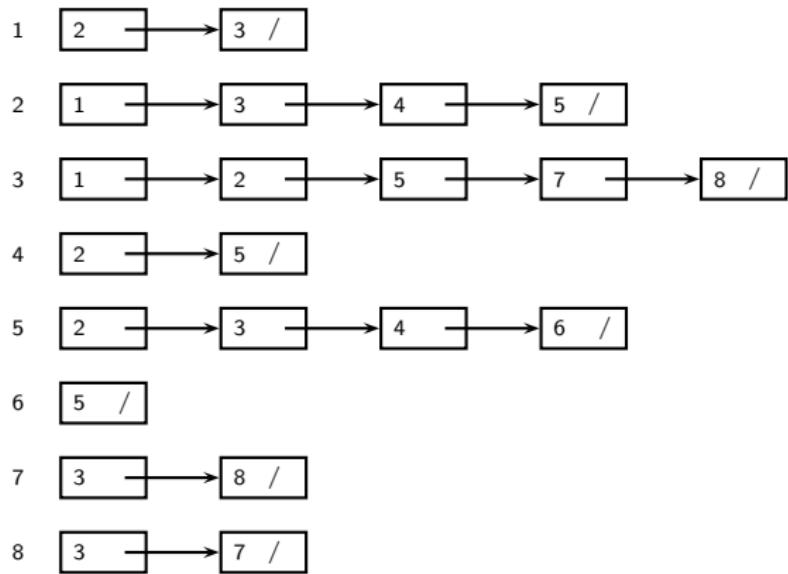
1. Poni Scoperto[ $s$ ] =true e Scoperto[ $v$ ] =false  $\forall v \neq s$
2. Inizializza  $L[0]$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Poni il contatore dei livelli  $i$  a 0
4. Inizializza l'albero BFS  $T$  a  $\emptyset$
5. While  $L[i]$  non è vuota
  - 6. Inizializza una lista vuota  $L[i + 1]$
  - 7. For ogni nodo  $u \in L[i]$ 
    - 8. Estrai  $u$  da  $L[i]$
    - 9. For ogni arco  $(u, v)$  incidente su  $u$ 
      - 10. If Scoperto[ $v$ ]=false then
        - 11. Poni Scoperto[ $v$ ]=true
        - 12. Aggiungi l'arco  $(u, v)$  all'albero  $T$
        - 13. Aggiungi  $v$  alla lista  $L[i + 1]$ ;  $d[v] \leftarrow i + 1$

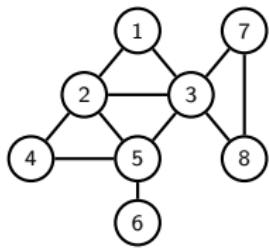
## BFS( $G, s$ )

1. Poni Scoperto[ $s$ ] =true e Scoperto[ $v$ ] =false  $\forall v \neq s$
2. Inizializza  $L[0]$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Poni il contatore dei livelli  $i$  a 0
4. Inizializza l'albero BFS  $T$  a  $\emptyset$
5. While  $L[i]$  non è vuota
  - 6. Inizializza una lista vuota  $L[i + 1]$
  - 7. For ogni nodo  $u \in L[i]$ 
    - 8. Estrai  $u$  da  $L[i]$
    - 9. For ogni arco  $(u, v)$  incidente su  $u$ 
      - 10. If Scoperto[ $v$ ]=false then
        - 11. Poni Scoperto[ $v$ ]=true
        - 12. Aggiungi l'arco  $(u, v)$  all'albero  $T$
        - 13. Aggiungi  $v$  alla lista  $L[i + 1]$ ;  $d[v] \leftarrow i + 1$
      - 14. Incrementa il contatore di livelli  $i$  di uno



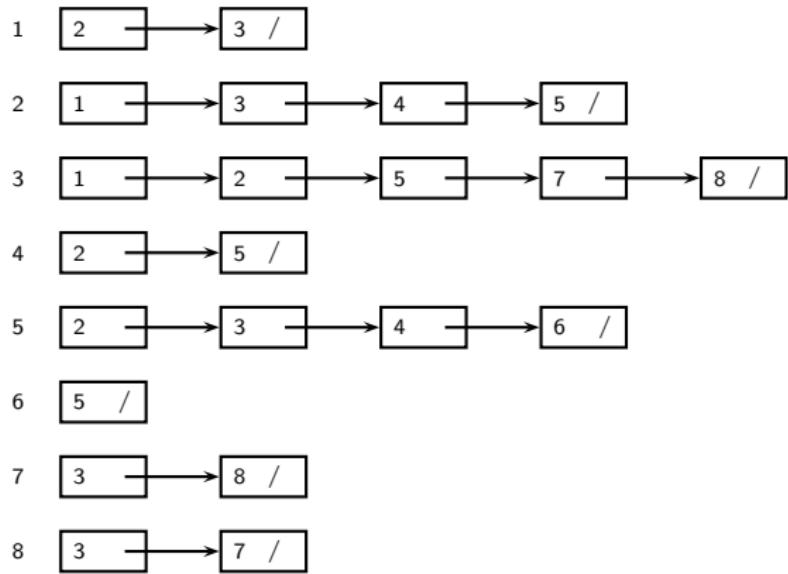
$L_0$

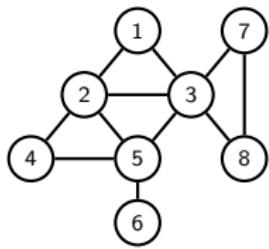




$L_0$

(1)

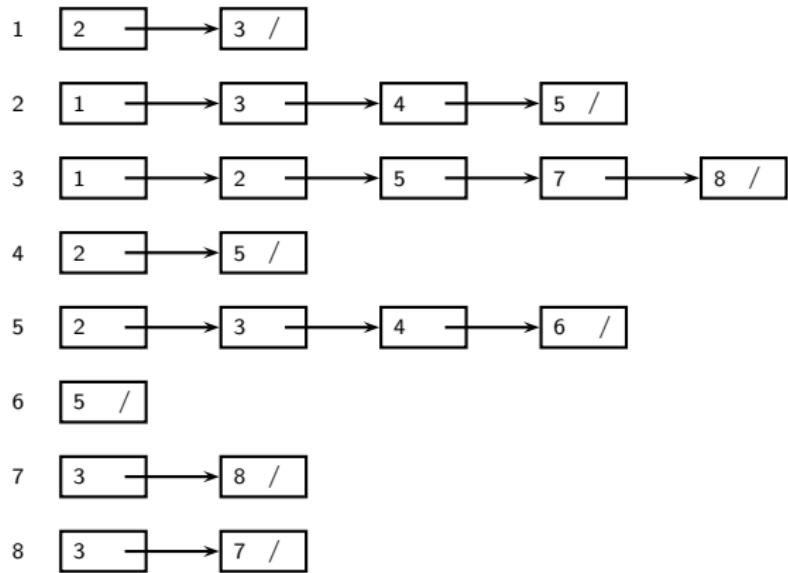


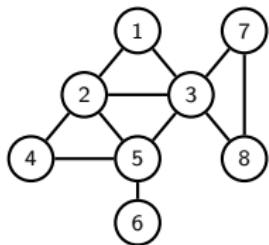


$L_0$

(1)

$L_1$



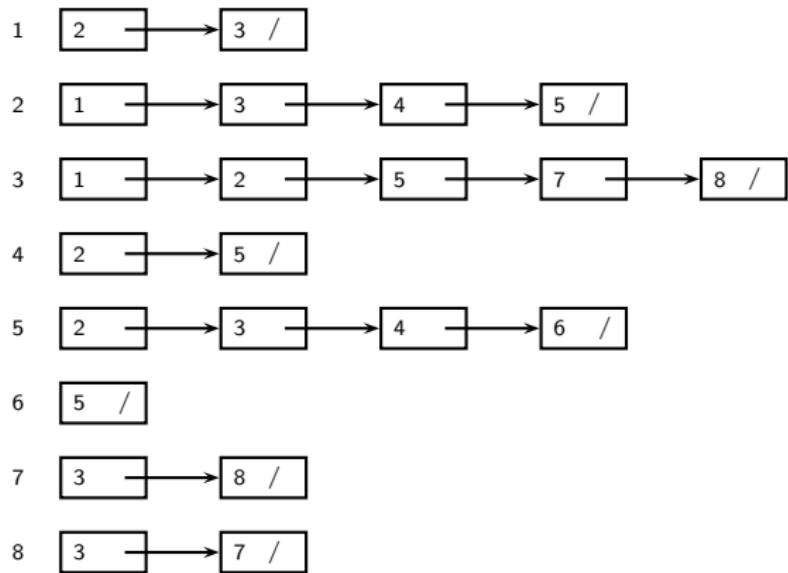


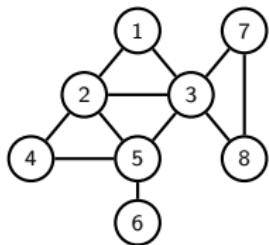
$L_0$

(1)

$L_1$

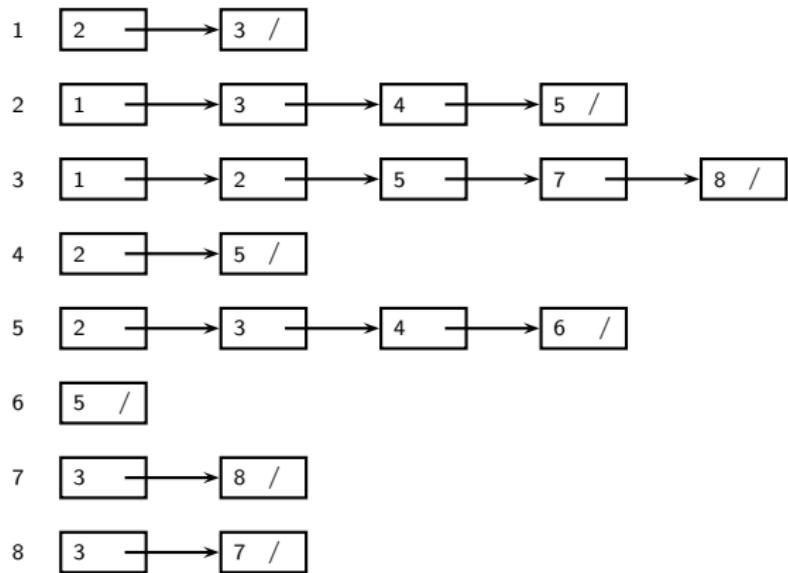
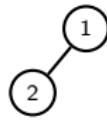
(2)

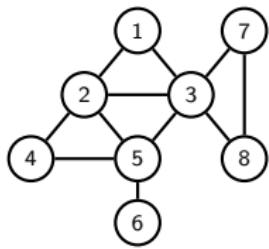




$L_0$

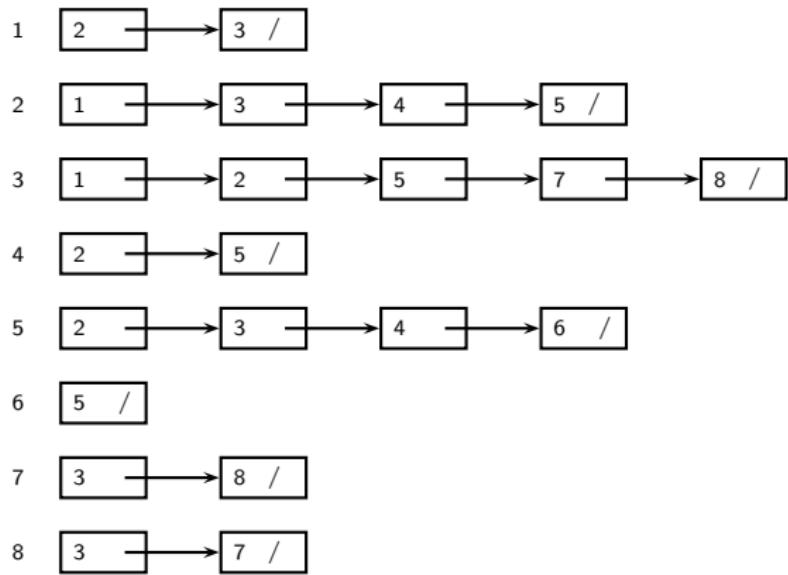
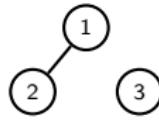
$L_1$

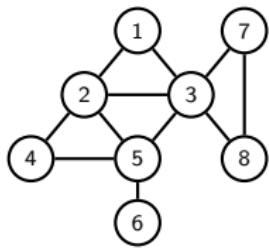




$L_0$

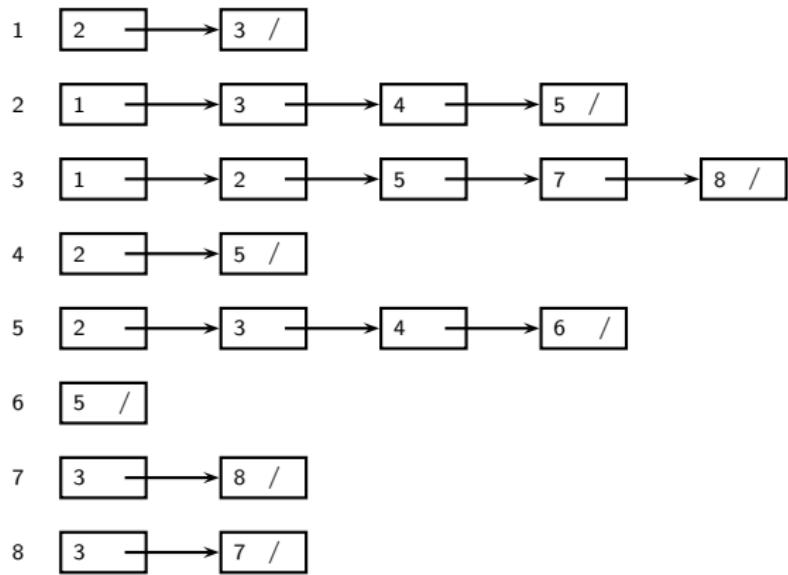
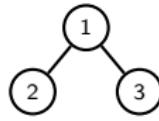
$L_1$

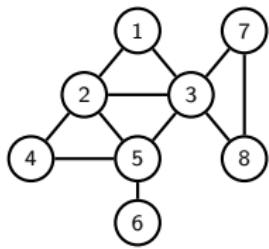




$L_0$

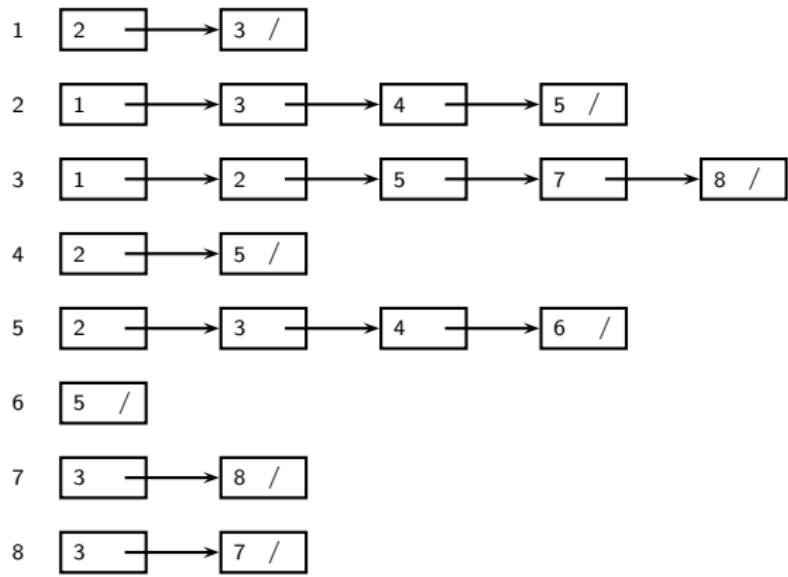
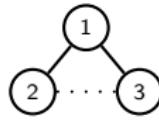
$L_1$

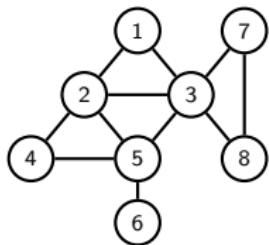




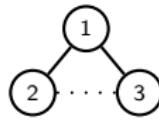
$L_0$

$L_1$



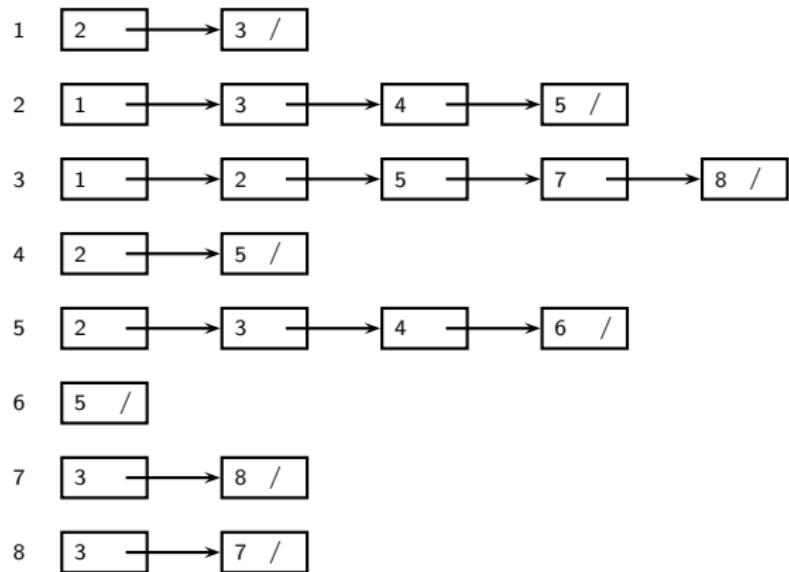


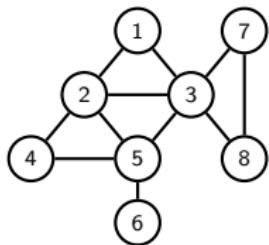
$L_0$



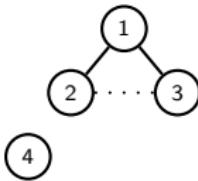
$L_1$

$L_2$



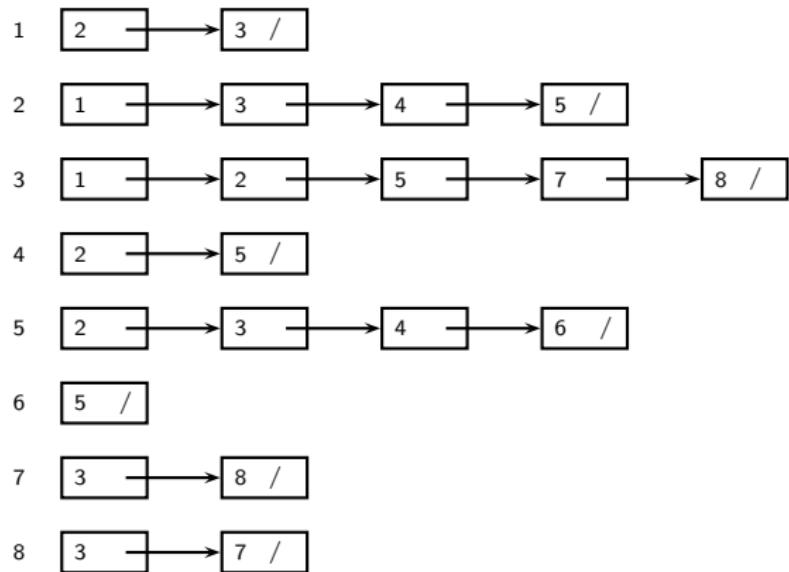


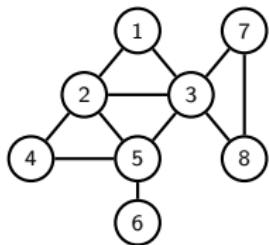
$L_0$



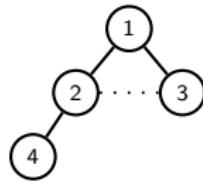
$L_1$

$L_2$



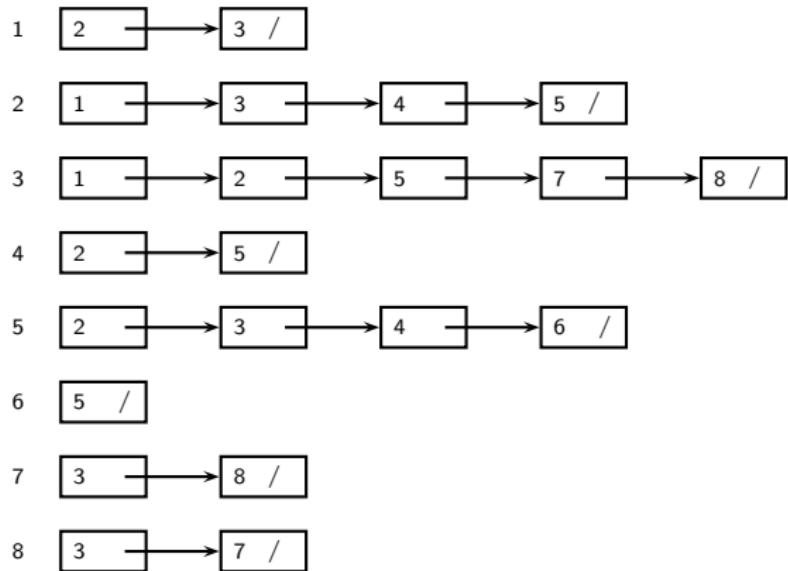


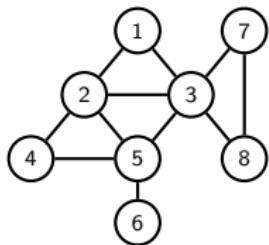
$L_0$



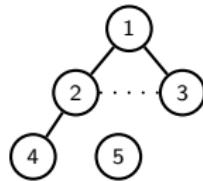
$L_1$

$L_2$



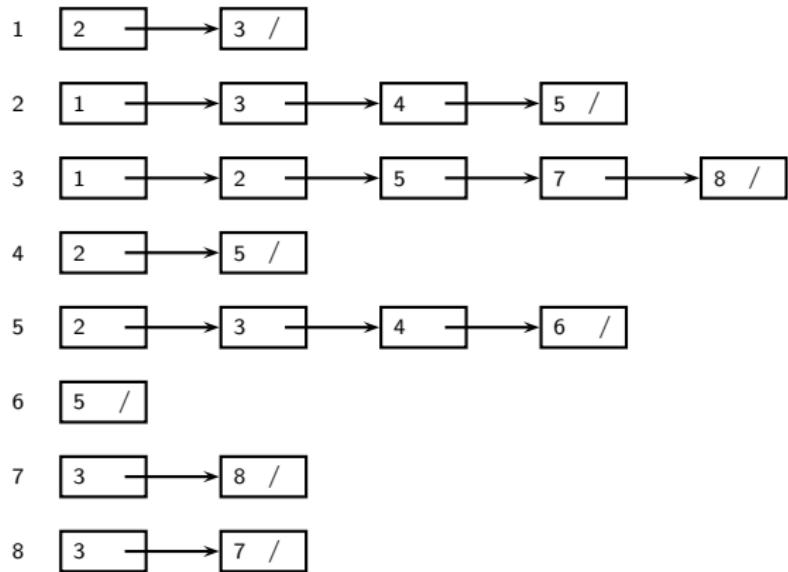


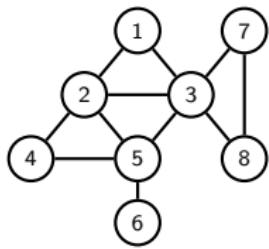
$L_0$



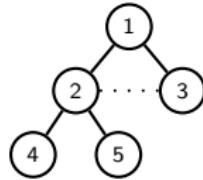
$L_1$

$L_2$



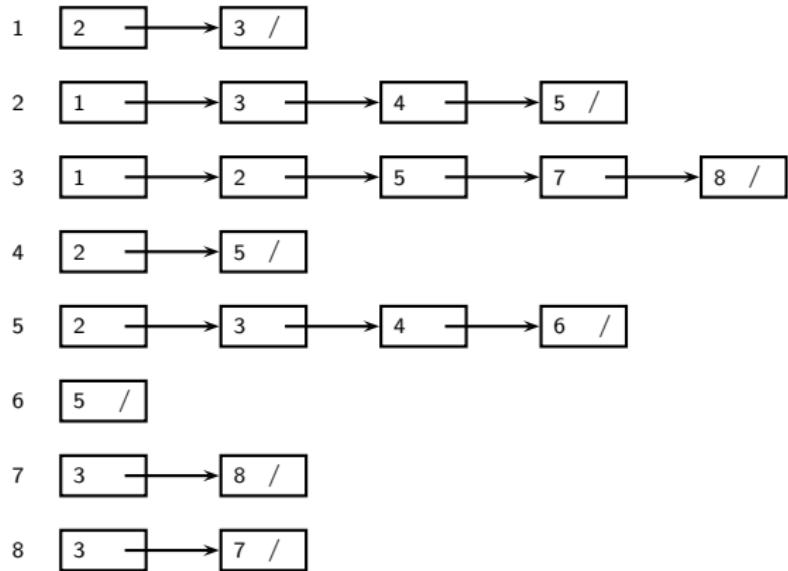


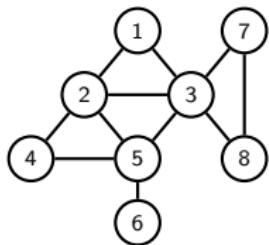
$L_0$



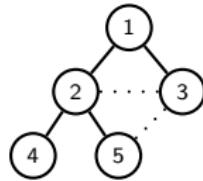
$L_1$

$L_2$



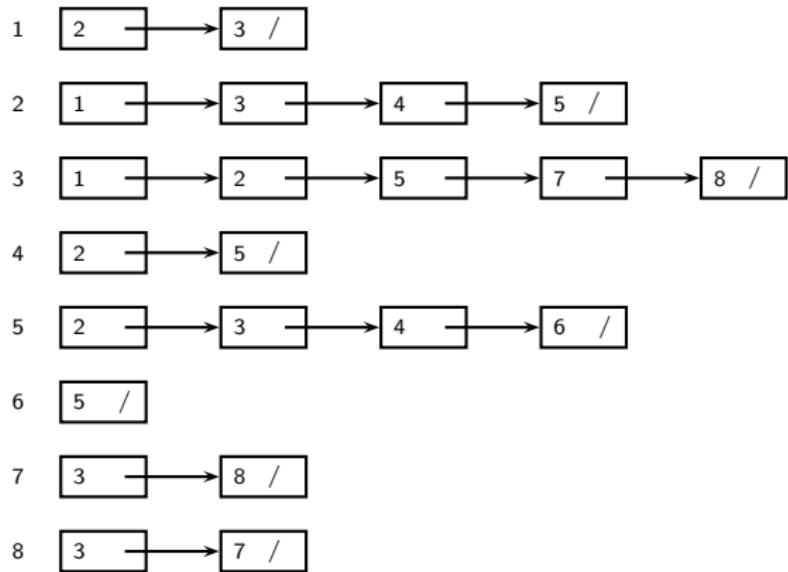


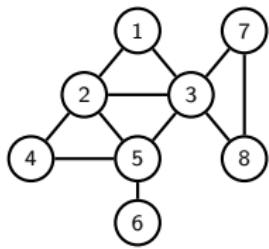
$L_0$



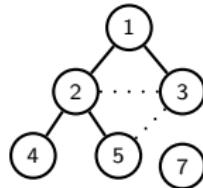
$L_1$

$L_2$



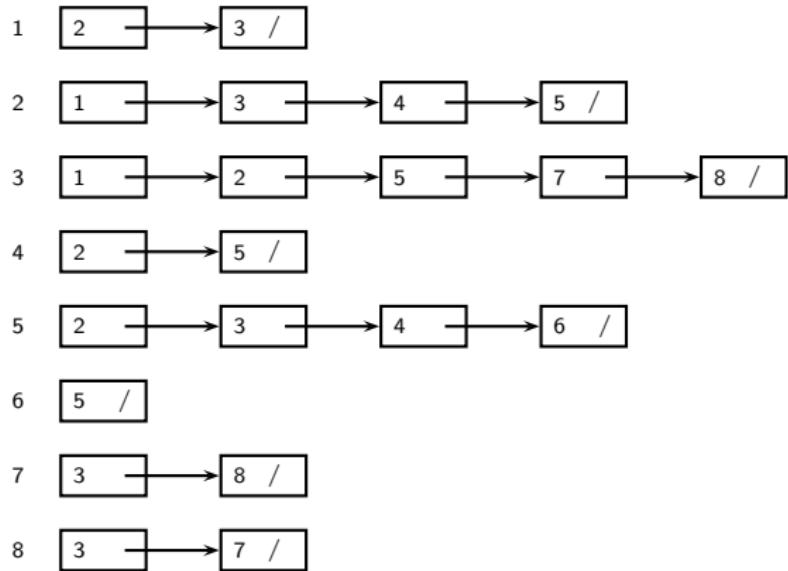


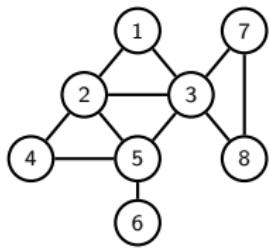
$L_0$



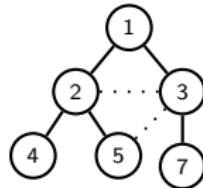
$L_1$

$L_2$



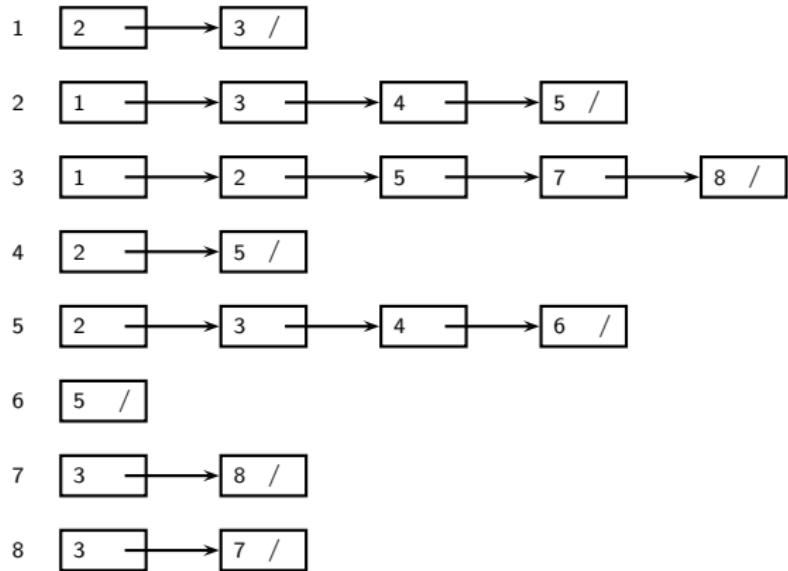


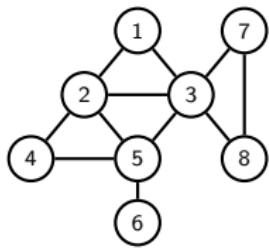
$L_0$



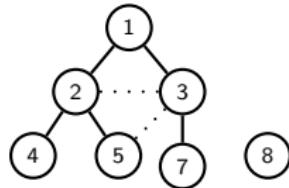
$L_1$

$L_2$



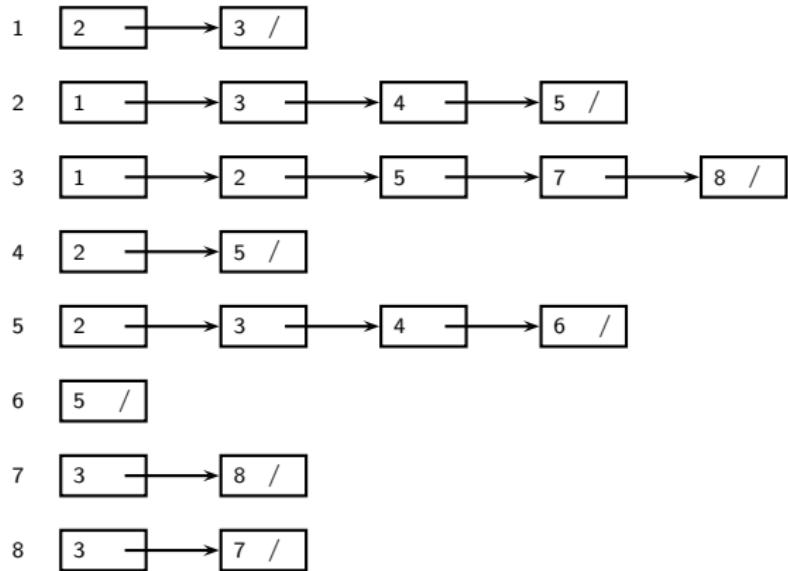


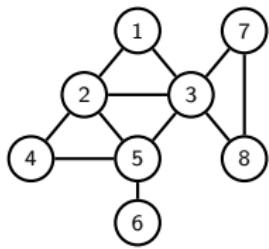
$L_0$



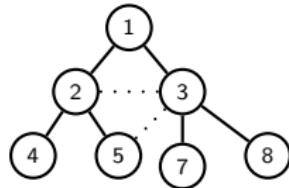
$L_1$

$L_2$



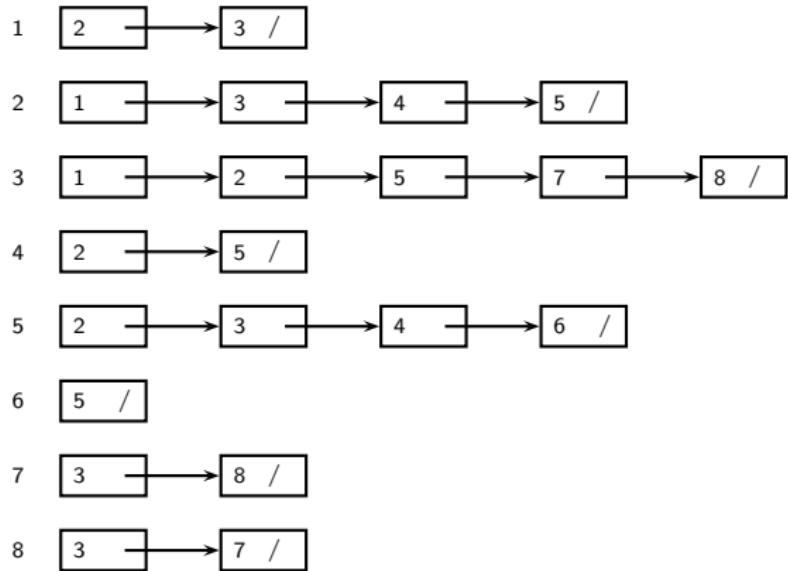


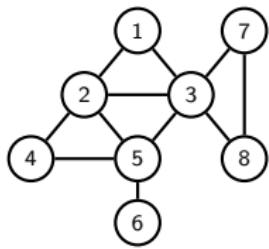
$L_0$



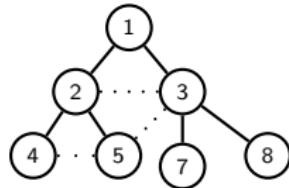
$L_1$

$L_2$



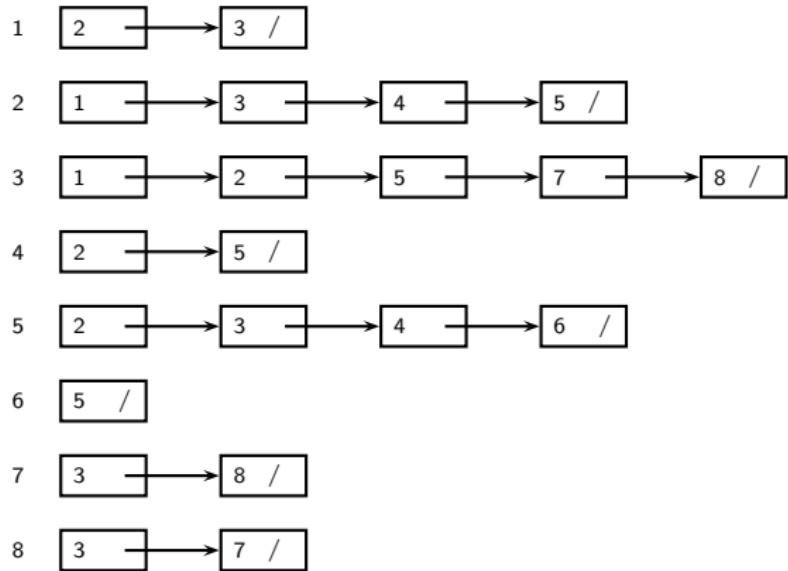


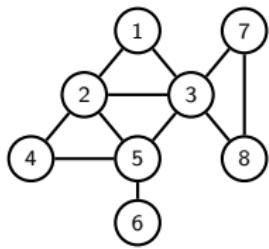
$L_0$



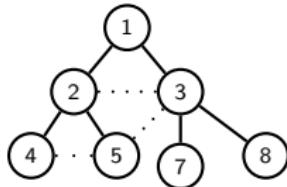
$L_1$

$L_2$





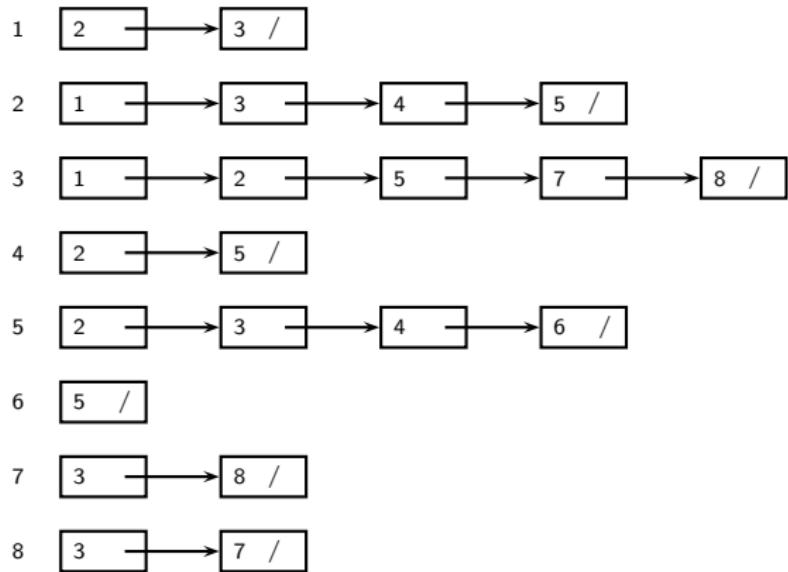
$L_0$

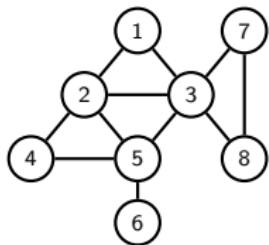


$L_1$

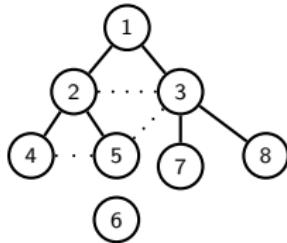
$L_2$

$L_3$





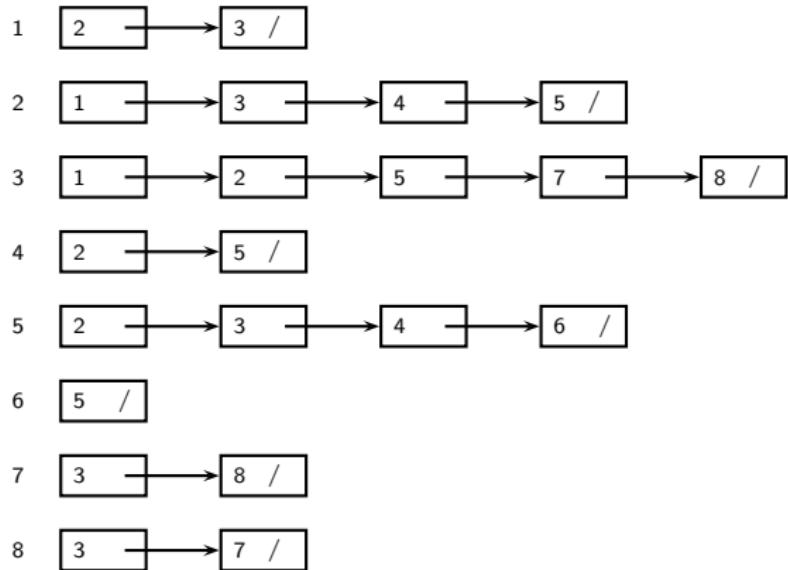
$L_0$

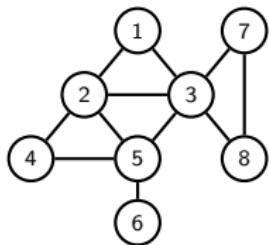


$L_1$

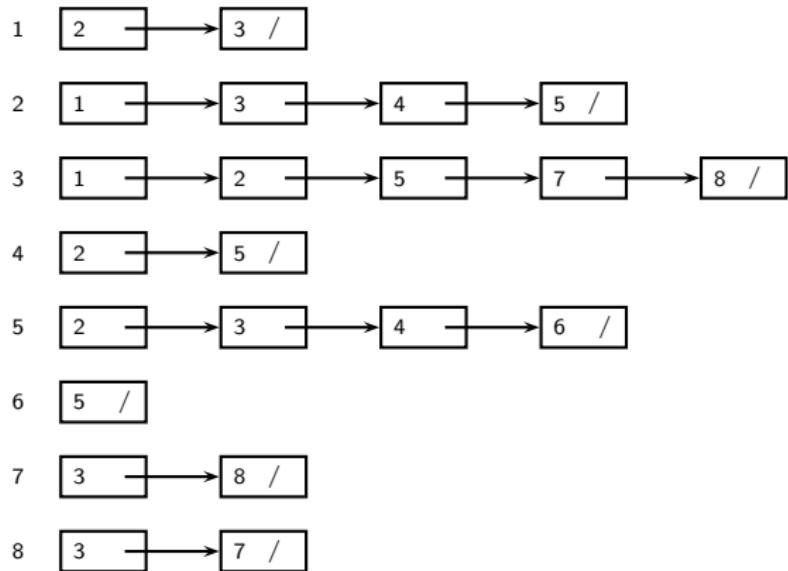
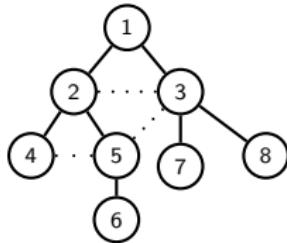
$L_2$

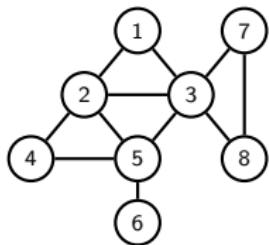
$L_3$



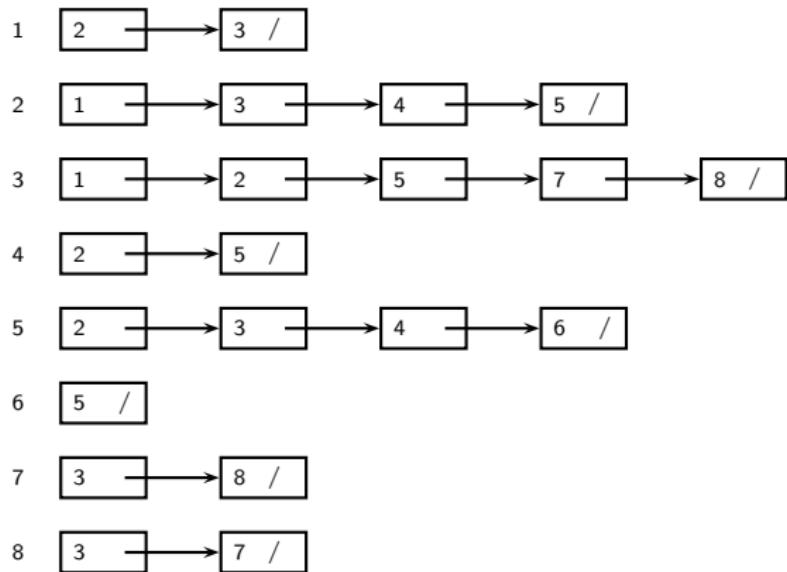
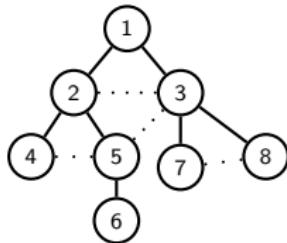


$L_0$   
 $L_1$   
 $L_2$   
 $L_3$





L<sub>0</sub>  
L<sub>1</sub>  
L<sub>2</sub>  
L<sub>3</sub>



$\text{BFS}(G, s)$

1. Poni  $\text{Scoperto}[s] = \text{true}$  e  $\text{Scoperto}[v] = \text{false} \quad \forall v \neq s$
2. Inizializza  $L[0]$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Poni il contatore dei livelli  $i$  a 0
4. Inizializza l'albero BFS  $T$  a  $\emptyset$
5. While  $L[i]$  non è vuota
  6. Inizializza una lista vuota  $L[i + 1]$
  7. For ogni nodo  $u \in L[i]$ 
    8. Estrai  $u$  da  $L[i]$
    9. For ogni arco  $(u, v)$  incidente su  $u$ 
      10. If  $\text{Scoperto}[v] = \text{false}$  then
        11. Poni  $\text{Scoperto}[v] = \text{true}$
        12. Aggiungi l'arco  $(u, v)$  all'albero  $T$
        13. Aggiungi  $v$  alla lista  $L[i + 1]$ ;  $d[v] \leftarrow i + 1$
    14. Incrementa il contatore di livelli  $i$  di uno

Le istruzioni da 1. a 4. richiedono in totale tempo  $\Theta(n)$ .

$\text{BFS}(G, s)$

1. Poni  $\text{Scoperto}[s] = \text{true}$  e  $\text{Scoperto}[v] = \text{false} \quad \forall v \neq s$
2. Inizializza  $L[0]$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Poni il contatore dei livelli  $i$  a 0
4. Inizializza l'albero BFS  $T$  a  $\emptyset$
5. While  $L[i]$  non è vuota
  6. Inizializza una lista vuota  $L[i + 1]$
  7. For ogni nodo  $u \in L[i]$ 
    8. Estrai  $u$  da  $L[i]$
    9. For ogni arco  $(u, v)$  incidente su  $u$ 
      10. If  $\text{Scoperto}[v] = \text{false}$  then
        11. Poni  $\text{Scoperto}[v] = \text{true}$
        12. Aggiungi l'arco  $(u, v)$  all'albero  $T$
        13. Aggiungi  $v$  alla lista  $L[i + 1]$ ;  $d[v] \leftarrow i + 1$
    14. Incrementa il contatore di livelli  $i$  di uno

Le istruzioni da 1. a 4. richiedono in totale tempo  $\Theta(n)$ . Ogni nodo viene processato (ed i relativi archi su di esso incidenti esaminati) una volta sola.

$\text{BFS}(G, s)$

1. Poni  $\text{Scoperto}[s] = \text{true}$  e  $\text{Scoperto}[v] = \text{false} \quad \forall v \neq s$
2. Inizializza  $L[0]$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Poni il contatore dei livelli  $i$  a 0
4. Inizializza l'albero BFS  $T$  a  $\emptyset$
5. While  $L[i]$  non è vuota
  6. Inizializza una lista vuota  $L[i + 1]$
  7. For ogni nodo  $u \in L[i]$ 
    8. Estrai  $u$  da  $L[i]$
    9. For ogni arco  $(u, v)$  incidente su  $u$ 
      10. If  $\text{Scoperto}[v] = \text{false}$  then
        11. Poni  $\text{Scoperto}[v] = \text{true}$
        12. Aggiungi l'arco  $(u, v)$  all'albero  $T$
        13. Aggiungi  $v$  alla lista  $L[i + 1]$ ;  $d[v] \leftarrow i + 1$
  14. Incrementa il contatore di livelli  $i$  di uno

Le istruzioni da 1. a 4. richiedono in totale tempo  $\Theta(n)$ . Ogni nodo viene processato (ed i relativi archi su di esso incidenti esaminati) una volta sola. Infatti, dopo che è stato posto  $\text{Scoperto}[v] = \text{true}$  il nodo  $v$  viene in seguito ignorato.

$\text{BFS}(G, s)$

1. Poni  $\text{Scoperto}[s] = \text{true}$  e  $\text{Scoperto}[v] = \text{false} \quad \forall v \neq s$
2. Inizializza  $L[0]$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Poni il contatore dei livelli  $i$  a 0
4. Inizializza l'albero BFS  $T$  a  $\emptyset$
5. While  $L[i]$  non è vuota
  6. Inizializza una lista vuota  $L[i + 1]$
  7. For ogni nodo  $u \in L[i]$ 
    8. Estrai  $u$  da  $L[i]$
    9. For ogni arco  $(u, v)$  incidente su  $u$ 
      10. If  $\text{Scoperto}[v] = \text{false}$  then
        11. Poni  $\text{Scoperto}[v] = \text{true}$
        12. Aggiungi l'arco  $(u, v)$  all'albero  $T$
        13. Aggiungi  $v$  alla lista  $L[i + 1]$ ;  $d[v] \leftarrow i + 1$
  14. Incrementa il contatore di livelli  $i$  di uno

Le istruzioni da 1. a 4. richiedono in totale tempo  $\Theta(n)$ . Ogni nodo viene processato (ed i relativi archi su di esso incidenti esaminati) una volta sola. Infatti, dopo che è stato posto  $\text{Scoperto}[v] = \text{true}$  il nodo  $v$  viene in seguito ignorato. Quindi il lavoro totale per ogni nodo  $v$  è  $O(\deg(v))$ , dove  $\deg(v) = \text{numero di archi incidenti su } v$ .

$\text{BFS}(G, s)$

1. Poni  $\text{Scoperto}[s] = \text{true}$  e  $\text{Scoperto}[v] = \text{false} \quad \forall v \neq s$
2. Inizializza  $L[0]$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Poni il contatore dei livelli  $i$  a 0
4. Inizializza l'albero BFS  $T$  a  $\emptyset$
5. While  $L[i]$  non è vuota
  6. Inizializza una lista vuota  $L[i + 1]$
  7. For ogni nodo  $u \in L[i]$ 
    8. Estrai  $u$  da  $L[i]$
    9. For ogni arco  $(u, v)$  incidente su  $u$ 
      10. If  $\text{Scoperto}[v] = \text{false}$  then
        11. Poni  $\text{Scoperto}[v] = \text{true}$
        12. Aggiungi l'arco  $(u, v)$  all'albero  $T$
        13. Aggiungi  $v$  alla lista  $L[i + 1]$ ;  $d[v] \leftarrow i + 1$
    14. Incrementa il contatore di livelli  $i$  di uno

Le istruzioni da 1. a 4. richiedono in totale tempo  $\Theta(n)$ . Ogni nodo viene processato (ed i relativi archi su di esso incidenti esaminati) una volta sola. Infatti, dopo che è stato posto  $\text{Scoperto}[v] = \text{true}$  il nodo  $v$  viene in seguito ignorato. Quindi il lavoro totale per ogni nodo  $v$  è  $O(\deg(v))$ , dove  $\deg(v) = \text{numero di archi incidenti su } v$ .

In totale, il lavoro è  $\Theta(n + \sum_v \deg(v))$

E quant'è  $\Theta(n + \sum_v \deg(v))$ ?

E quant'è  $\Theta(n + \sum_v \deg(v))$ ?

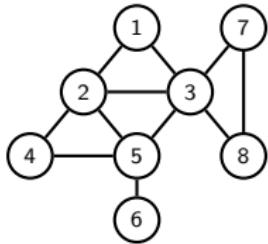
$\deg(v)$  = numero di archi incidenti su  $v$

E quant'è  $\Theta(n + \sum_v \deg(v))$ ?

$\deg(v)$  = numero di archi incidenti su  $v \implies \sum_v \deg(v) = 2|E|$  in quanto nella somma ogni arco  $(u, v)$  viene contato *due volte*, quando sommiamo  $\deg(u)$  e quando sommiamo  $\deg(v)$

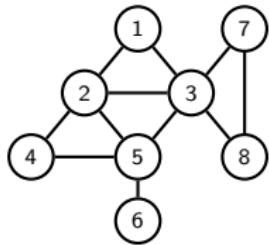
E quant'è  $\Theta(n + \sum_v \deg(v))$ ?

$\deg(v)$  = numero di archi incidenti su  $v \implies \sum_v \deg(v) = 2|E|$  in quanto nella somma ogni arco  $(u, v)$  viene contato *due volte*, quando sommiamo  $\deg(u)$  e quando sommiamo  $\deg(v)$



E quant'è  $\Theta(n + \sum_v \deg(v))$ ?

$\deg(v)$  = numero di archi incidenti su  $v \implies \sum_v \deg(v) = 2|E|$  in quanto nella somma ogni arco  $(u, v)$  viene contato *due volte*, quando sommiamo  $\deg(u)$  e quando sommiamo  $\deg(v)$



In totale, la complessità di  $\text{BFS}(G, s)$ , con  $G = (V, E)$ ,  $|V| = n$ ,  $|E| = m$ , è quindi  $\Theta(n + m)$

Abbiamo descritto l'algoritmo BFS facente uso di  $n$  liste separate  $L[i]$ , una per ogni livello  $L_i$ .

Abbiamo descritto l'algoritmo BFS facente uso di  $n$  liste separate  $L[i]$ , una per ogni livello  $L_i$ . Se ne può fare a meno ed usare una sola lista, che gestiremo come una coda (*first-in, first-out*).

Abbiamo descritto l'algoritmo BFS facente uso di  $n$  liste separate  $L[i]$ , una per ogni livello  $L_i$ . Se ne può fare a meno ed usare una sola lista, che gestiremo come una coda (*first-in, first-out*).

- ▶ In questo modo i nodi sono processati nell'ordine in cui essi vengono scoperti: ogni volta che un nodo viene scoperto per la prima volta viene aggiunto alla fine della coda e l'algoritmo processa sempre il nodo che si trova alla testa della coda.

Abbiamo descritto l'algoritmo BFS facente uso di  $n$  liste separate  $L[i]$ , una per ogni livello  $L_i$ . Se ne può fare a meno ed usare una sola lista, che gestiremo come una coda (*first-in, first-out*).

- ▶ In questo modo i nodi sono processati nell'ordine in cui essi vengono scoperti: ogni volta che un nodo viene scoperto per la prima volta viene aggiunto alla fine della coda e l'algoritmo processa sempre il nodo che si trova alla testa della coda.
- ▶ Se manteniamo i nodi scoperti in quest'ordine, allora i nodi nel livello  $L_i$  vengono esaminati chiaramente prima dei nodi del livello  $L_{i+1}$

Abbiamo descritto l'algoritmo BFS facente uso di  $n$  liste separate  $L[i]$ , una per ogni livello  $L_i$ . Se ne può fare a meno ed usare una sola lista, che gestiremo come una coda (*first-in, first-out*).

- ▶ In questo modo i nodi sono processati nell'ordine in cui essi vengono scoperti: ogni volta che un nodo viene scoperto per la prima volta viene aggiunto alla fine della coda e l'algoritmo processa sempre il nodo che si trova alla testa della coda.
- ▶ Se manteniamo i nodi scoperti in quest'ordine, allora i nodi nel livello  $L_i$  vengono esaminati chiaramente prima dei nodi del livello  $L_{i+1}$ .
- ▶ Quindi i nodi del livello  $L_i$  formano una sequenza di nodi consecutivi nella coda, e sono seguiti dalla sequenza consecutiva dei nodi nel livello  $L_{i+1}$ , e così via...

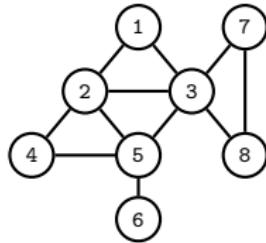
Abbiamo descritto l'algoritmo BFS facente uso di  $n$  liste separate  $L[i]$ , una per ogni livello  $L_i$ . Se ne può fare a meno ed usare una sola lista, che gestiremo come una coda (*first-in, first-out*).

- ▶ In questo modo i nodi sono processati nell'ordine in cui essi vengono scoperti: ogni volta che un nodo viene scoperto per la prima volta viene aggiunto alla fine della coda e l'algoritmo processa sempre il nodo che si trova alla testa della coda.
- ▶ Se manteniamo i nodi scoperti in quest'ordine, allora i nodi nel livello  $L_i$  vengono esaminati chiaramente prima dei nodi del livello  $L_{i+1}$ .
- ▶ Quindi i nodi del livello  $L_i$  formano una sequenza di nodi consecutivi nella coda, e sono seguiti dalla sequenza consecutiva dei nodi nel livello  $L_{i+1}$ , e così via...

Pertanto quest'implementazione con un'unica coda produrrà lo stesso risultato dell'implementazione della BFS prima descritta.

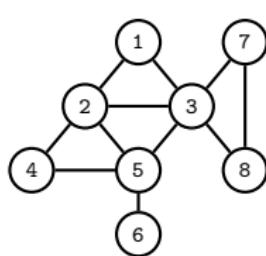
BFS( $G, s$ )

1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



BFS( $G, s$ )

1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$

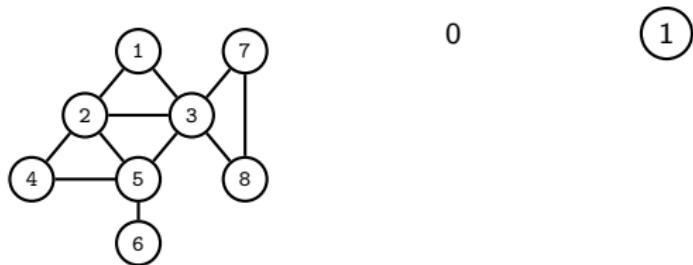


1

$$Q = \{1\}$$

$\text{BFS}(G, s)$

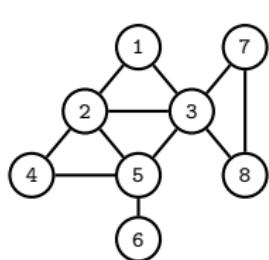
1. Poni  $\text{Scoperto}[s] = \text{true}$  e  $\text{Scoperto}[v] = \text{false} \quad \forall v \neq s$
  2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
  3. Inizializza l'albero BFS  $T$  a  $\emptyset$
  4. While  $Q$  non è vuota
    5. Estrai il nodo  $u$  dalla testa della lista  $Q$
    6. Considera ciascun arco  $(u, v)$  incidente su  $u$
    7. If  $\text{Scoperto}[v] = \text{false}$  then
      8. Poni  $\text{Scoperto}[v] = \text{true}$
      9. Aggiungi l'arco  $(u, v)$  all'albero  $T$
      10. Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



$$Q = \{1\}$$

BFS( $G, s$ )

1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



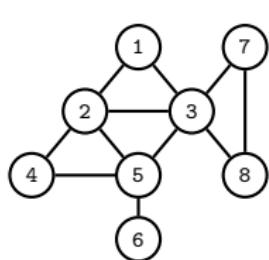
0

1

$$Q = \{1\} \quad Q = \emptyset$$

BFS( $G, s$ )

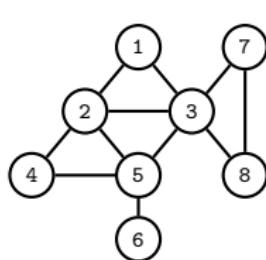
1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



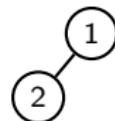
$$Q = \{1\} \quad Q = \emptyset$$

BFS( $G, s$ )

1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



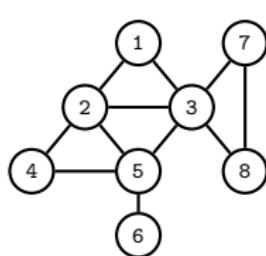
0



$$Q = \{1\} \quad Q = \emptyset$$

BFS( $G, s$ )

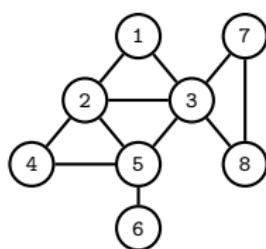
1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



$$Q = \{1\} \quad Q = \emptyset \quad Q = \{2\}$$

BFS( $G, s$ )

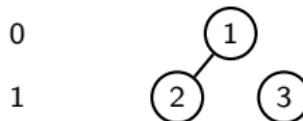
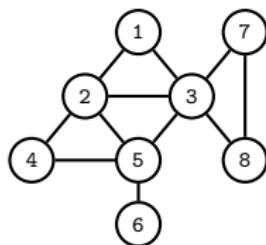
1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



$$Q = \{1\} \quad Q = \emptyset \quad Q = \{2\}$$

BFS( $G, s$ )

1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



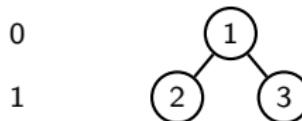
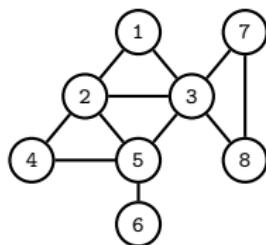
$$Q = \{1\}$$

$$Q = \emptyset$$

$$Q = \{2\}$$

BFS( $G, s$ )

1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



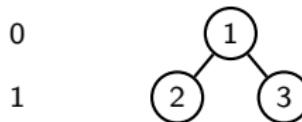
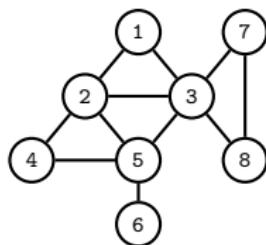
$$Q = \{1\}$$

$$Q = \emptyset$$

$$Q = \{2\}$$

BFS( $G, s$ )

1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



$$Q = \{1\}$$

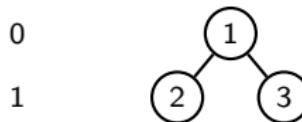
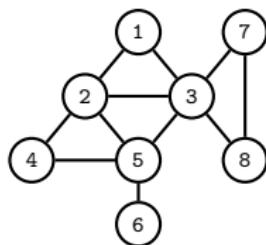
$$Q = \emptyset$$

$$Q = \{2\}$$

$$Q = \{2, 3\}$$

BFS( $G, s$ )

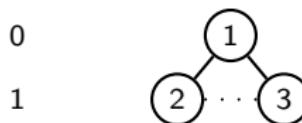
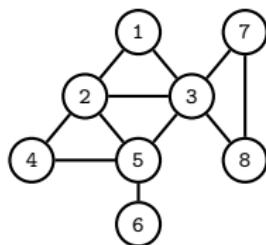
1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



$$Q = \{1\} \quad Q = \emptyset \quad Q = \{2\} \quad Q = \{2, 3\} \quad Q = \{3\}$$

BFS( $G, s$ )

1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



$$Q = \{1\}$$

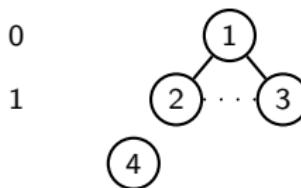
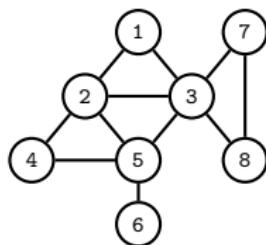
$$Q = \emptyset$$

$$Q = \{2\}$$

$$Q = \{2, 3\} \quad Q = \{3\}$$

BFS( $G, s$ )

1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



$$Q = \{1\}$$

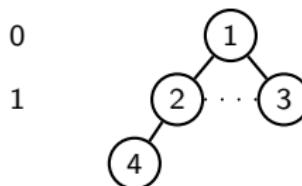
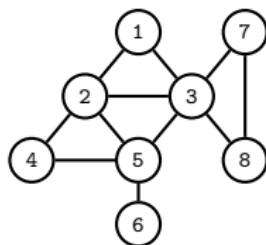
$$Q = \emptyset$$

$$Q = \{2\}$$

$$Q = \{2, 3\} \quad Q = \{3\}$$

BFS( $G, s$ )

1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



$$Q = \{1\}$$

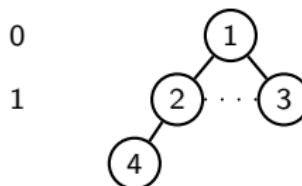
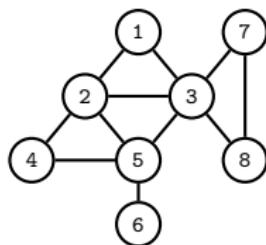
$$Q = \emptyset$$

$$Q = \{2\}$$

$$Q = \{2, 3\} \quad Q = \{3\}$$

BFS( $G, s$ )

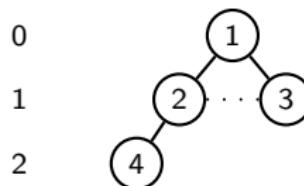
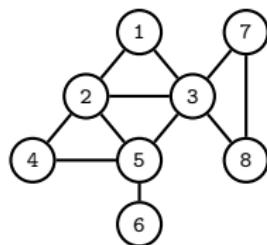
1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



$$\begin{array}{lllll} Q = \{1\} & Q = \emptyset & Q = \{2\} & Q = \{2, 3\} & Q = \{3\} \\ Q = \{3, 4\} & & & & \end{array}$$

BFS( $G, s$ )

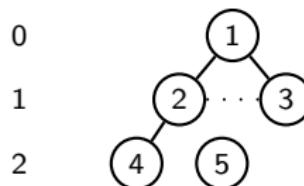
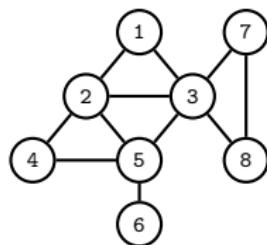
1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v]=\text{false}$  then
8.     Poni  $\text{Scoperto}[v]=\text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



$$\begin{array}{lllll} Q = \{1\} & Q = \emptyset & Q = \{2\} & Q = \{2, 3\} & Q = \{3\} \\ Q = \{3, 4\} & & & & \end{array}$$

BFS( $G, s$ )

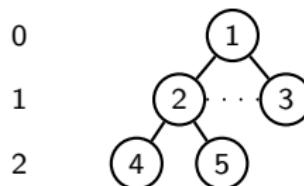
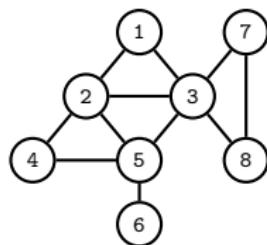
1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



$$\begin{array}{lllll} Q = \{1\} & Q = \emptyset & Q = \{2\} & Q = \{2, 3\} & Q = \{3\} \\ Q = \{3, 4\} & & & & \end{array}$$

BFS( $G, s$ )

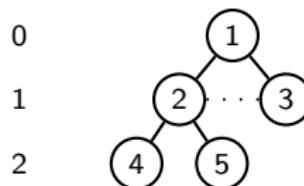
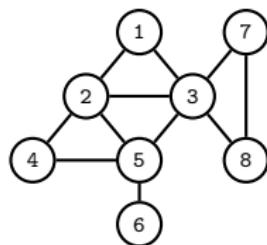
1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



$$\begin{array}{lllll} Q = \{1\} & Q = \emptyset & Q = \{2\} & Q = \{2, 3\} & Q = \{3\} \\ Q = \{3, 4\} & & & & \end{array}$$

BFS( $G, s$ )

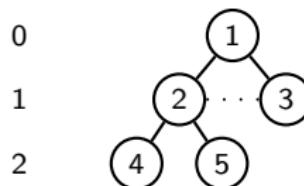
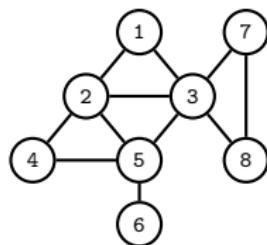
1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



$$\begin{array}{lllll} Q = \{1\} & Q = \emptyset & Q = \{2\} & Q = \{2, 3\} & Q = \{3\} \\ Q = \{3, 4\} & Q = \{3, 4, 5\} & & & \end{array}$$

BFS( $G, s$ )

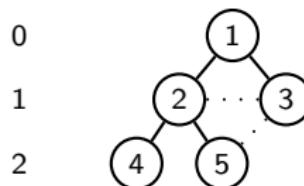
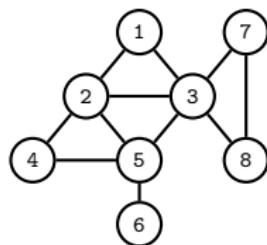
1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



$$\begin{array}{lllll} Q = \{1\} & Q = \emptyset & Q = \{2\} & Q = \{2, 3\} & Q = \{3\} \\ Q = \{3, 4\} & Q = \{3, 4, 5\} & Q = \{4, 5\} & & \end{array}$$

BFS( $G, s$ )

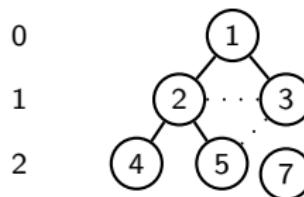
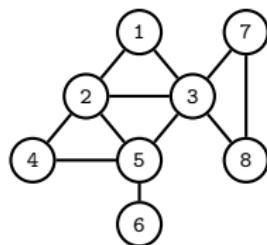
1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



$$\begin{array}{lllll} Q = \{1\} & Q = \emptyset & Q = \{2\} & Q = \{2, 3\} & Q = \{3\} \\ Q = \{3, 4\} & Q = \{3, 4, 5\} & Q = \{4, 5\} & & \end{array}$$

BFS( $G, s$ )

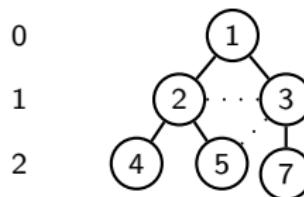
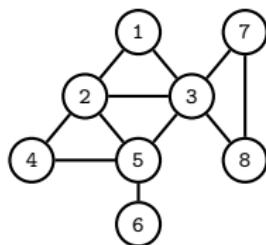
1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



$$\begin{array}{lllll} Q = \{1\} & Q = \emptyset & Q = \{2\} & Q = \{2, 3\} & Q = \{3\} \\ Q = \{3, 4\} & Q = \{3, 4, 5\} & Q = \{4, 5\} & & \end{array}$$

BFS( $G, s$ )

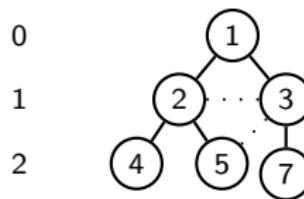
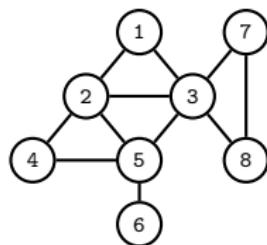
1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



$$\begin{array}{lllll} Q = \{1\} & Q = \emptyset & Q = \{2\} & Q = \{2, 3\} & Q = \{3\} \\ Q = \{3, 4\} & Q = \{3, 4, 5\} & Q = \{4, 5\} & & \end{array}$$

BFS( $G, s$ )

1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$

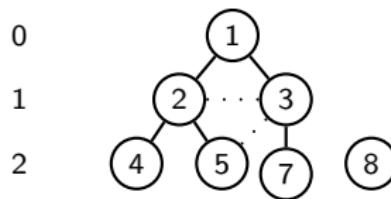
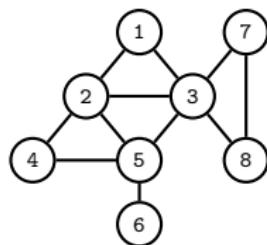


$$Q = \{1\} \quad Q = \emptyset \quad Q = \{2\} \quad Q = \{2, 3\} \quad Q = \{3\}$$

$$Q = \{3, 4\} \quad Q = \{3, 4, 5\} \quad Q = \{4, 5\} \quad Q = \{4, 5, 7\}$$

BFS( $G, s$ )

1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$

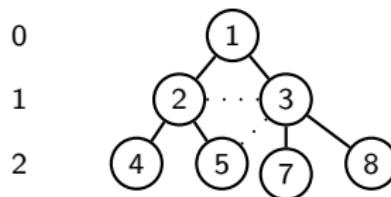
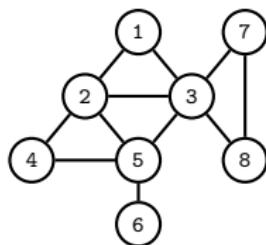


$$Q = \{1\} \quad Q = \emptyset \quad Q = \{2\} \quad Q = \{2, 3\} \quad Q = \{3\}$$

$$Q = \{3, 4\} \quad Q = \{3, 4, 5\} \quad Q = \{4, 5\} \quad Q = \{4, 5, 7\}$$

BFS( $G, s$ )

1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$

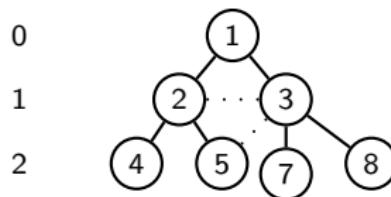
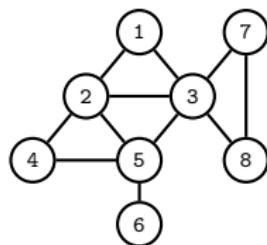


$$Q = \{1\} \quad Q = \emptyset \quad Q = \{2\} \quad Q = \{2, 3\} \quad Q = \{3\}$$

$$Q = \{3, 4\} \quad Q = \{3, 4, 5\} \quad Q = \{4, 5\} \quad Q = \{4, 5, 7\}$$

BFS( $G, s$ )

1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



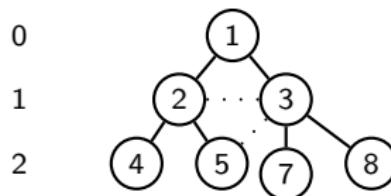
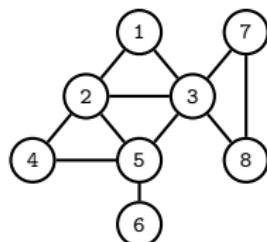
$$Q = \{1\} \quad Q = \emptyset \quad Q = \{2\} \quad Q = \{2, 3\} \quad Q = \{3\}$$

$$Q = \{3, 4\} \quad Q = \{3, 4, 5\} \quad Q = \{4, 5\} \quad Q = \{4, 5, 7\}$$

$$Q = \{4, 5, 7, 8\}$$

BFS( $G, s$ )

1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



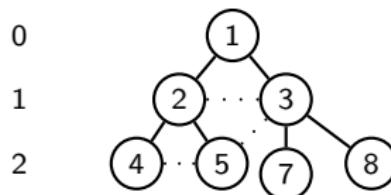
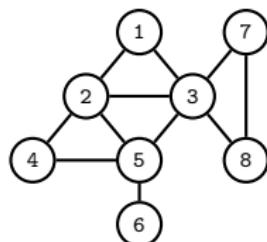
$$Q = \{1\} \quad Q = \emptyset \quad Q = \{2\} \quad Q = \{2, 3\} \quad Q = \{3\}$$

$$Q = \{3, 4\} \quad Q = \{3, 4, 5\} \quad Q = \{4, 5\} \quad Q = \{4, 5, 7\}$$

$$Q = \{4, 5, 7, 8\} \quad Q = \{5, 7, 8\}$$

BFS( $G, s$ )

1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



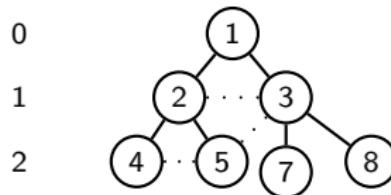
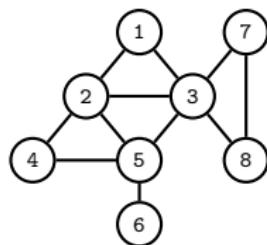
$$Q = \{1\} \quad Q = \emptyset \quad Q = \{2\} \quad Q = \{2, 3\} \quad Q = \{3\}$$

$$Q = \{3, 4\} \quad Q = \{3, 4, 5\} \quad Q = \{4, 5\} \quad Q = \{4, 5, 7\}$$

$$Q = \{4, 5, 7, 8\} \quad Q = \{5, 7, 8\}$$

BFS( $G, s$ )

1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



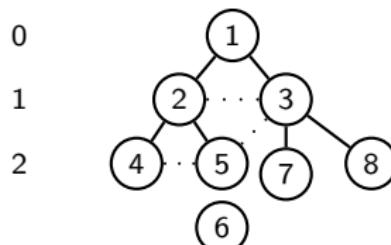
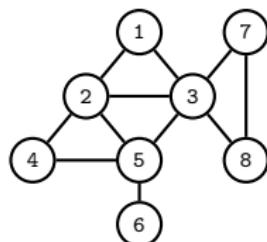
$$Q = \{1\} \quad Q = \emptyset \quad Q = \{2\} \quad Q = \{2, 3\} \quad Q = \{3\}$$

$$Q = \{3, 4\} \quad Q = \{3, 4, 5\} \quad Q = \{4, 5\} \quad Q = \{4, 5, 7\}$$

$$Q = \{4, 5, 7, 8\} \quad Q = \{5, 7, 8\} \quad Q = \{7, 8\}$$

BFS( $G, s$ )

1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



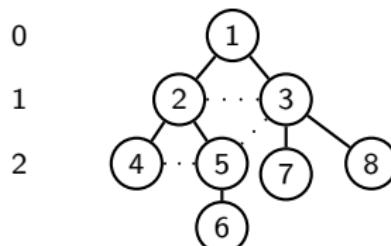
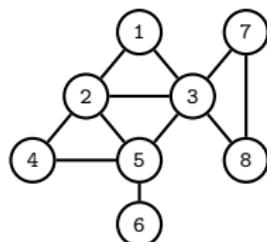
$$Q = \{1\} \quad Q = \emptyset \quad Q = \{2\} \quad Q = \{2, 3\} \quad Q = \{3\}$$

$$Q = \{3, 4\} \quad Q = \{3, 4, 5\} \quad Q = \{4, 5\} \quad Q = \{4, 5, 7\}$$

$$Q = \{4, 5, 7, 8\} \quad Q = \{5, 7, 8\} \quad Q = \{7, 8\}$$

BFS( $G, s$ )

1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



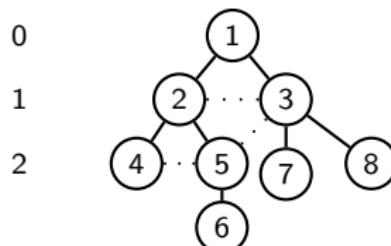
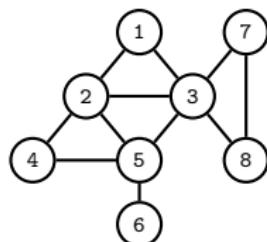
$$Q = \{1\} \quad Q = \emptyset \quad Q = \{2\} \quad Q = \{2, 3\} \quad Q = \{3\}$$

$$Q = \{3, 4\} \quad Q = \{3, 4, 5\} \quad Q = \{4, 5\} \quad Q = \{4, 5, 7\}$$

$$Q = \{4, 5, 7, 8\} \quad Q = \{5, 7, 8\} \quad Q = \{7, 8\}$$

BFS( $G, s$ )

1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



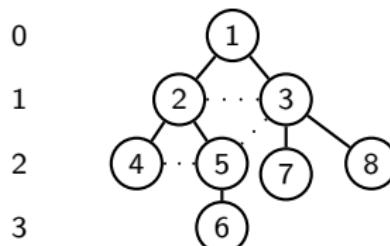
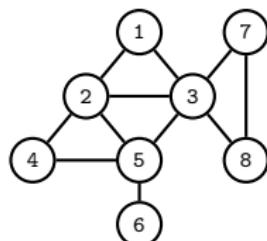
$$Q = \{1\} \quad Q = \emptyset \quad Q = \{2\} \quad Q = \{2, 3\} \quad Q = \{3\}$$

$$Q = \{3, 4\} \quad Q = \{3, 4, 5\} \quad Q = \{4, 5\} \quad Q = \{4, 5, 7\}$$

$$Q = \{4, 5, 7, 8\} \quad Q = \{5, 7, 8\} \quad Q = \{7, 8\} \quad Q = \{7, 8, 6\}$$

BFS( $G, s$ )

1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



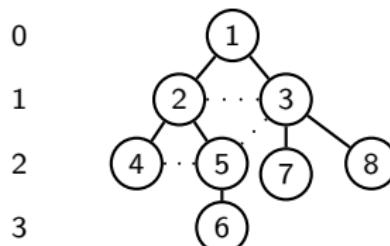
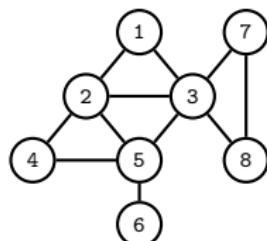
$$Q = \{1\} \quad Q = \emptyset \quad Q = \{2\} \quad Q = \{2, 3\} \quad Q = \{3\}$$

$$Q = \{3, 4\} \quad Q = \{3, 4, 5\} \quad Q = \{4, 5\} \quad Q = \{4, 5, 7\}$$

$$Q = \{4, 5, 7, 8\} \quad Q = \{5, 7, 8\} \quad Q = \{7, 8\} \quad Q = \{7, 8, 6\}$$

BFS( $G, s$ )

1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



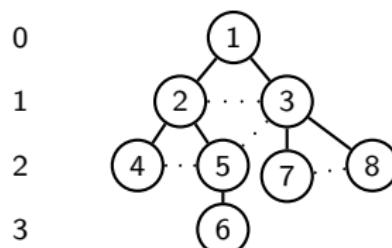
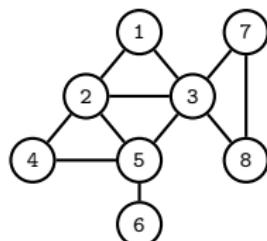
$$Q = \{1\} \quad Q = \emptyset \quad Q = \{2\} \quad Q = \{2, 3\} \quad Q = \{3\}$$

$$Q = \{3, 4\} \quad Q = \{3, 4, 5\} \quad Q = \{4, 5\} \quad Q = \{4, 5, 7\}$$

$$Q = \{4, 5, 7, 8\} \quad Q = \{5, 7, 8\} \quad Q = \{7, 8\} \quad Q = \{7, 8, 6\} \quad Q = \{8, 6\}$$

BFS( $G, s$ )

1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



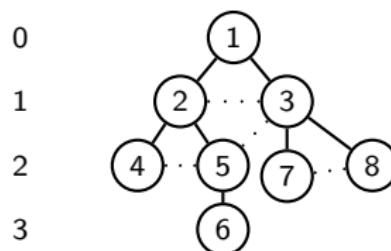
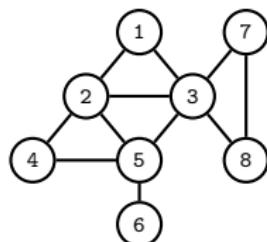
$$Q = \{1\} \quad Q = \emptyset \quad Q = \{2\} \quad Q = \{2, 3\} \quad Q = \{3\}$$

$$Q = \{3, 4\} \quad Q = \{3, 4, 5\} \quad Q = \{4, 5\} \quad Q = \{4, 5, 7\}$$

$$Q = \{4, 5, 7, 8\} \quad Q = \{5, 7, 8\} \quad Q = \{7, 8\} \quad Q = \{7, 8, 6\} \quad Q = \{8, 6\}$$

BFS( $G, s$ )

1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



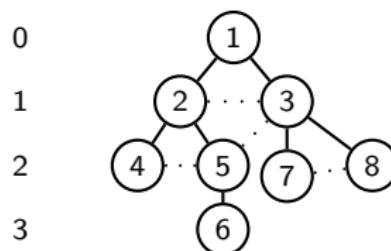
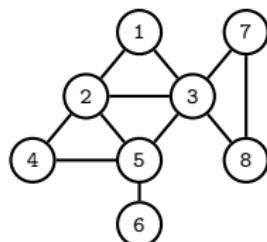
$$Q = \{1\} \quad Q = \emptyset \quad Q = \{2\} \quad Q = \{2, 3\} \quad Q = \{3\}$$

$$Q = \{3, 4\} \quad Q = \{3, 4, 5\} \quad Q = \{4, 5\} \quad Q = \{4, 5, 7\}$$

$$Q = \{4, 5, 7, 8\} \quad Q = \{5, 7, 8\} \quad Q = \{7, 8\} \quad Q = \{7, 8, 6\} \quad Q = \{8, 6\} \quad Q = \{6\}$$

BFS( $G, s$ )

1. Poni Scoperto[ $s$ ] = true e Scoperto[ $v$ ] = false  $\forall v \neq s$
2. Inizializza  $Q$  in modo che contenga  $s$ ;  $d[s] \leftarrow 0$
3. Inizializza l'albero BFS  $T$  a  $\emptyset$
4. While  $Q$  non è vuota
5. Estrai il nodo  $u$  dalla testa della lista  $Q$
6. Considera ciascun arco  $(u, v)$  incidente su  $u$
7. If  $\text{Scoperto}[v] = \text{false}$  then
8.     Poni  $\text{Scoperto}[v] = \text{true}$
9.     Aggiungi l'arco  $(u, v)$  all'albero  $T$
10.    Aggiungi  $v$  alla fine della coda  $Q$ ;  $d[v] \leftarrow d[u] + 1$



$$Q = \{1\} \quad Q = \emptyset \quad Q = \{2\} \quad Q = \{2, 3\} \quad Q = \{3\}$$

$$Q = \{3, 4\} \quad Q = \{3, 4, 5\} \quad Q = \{4, 5\} \quad Q = \{4, 5, 7\}$$

$$Q = \{4, 5, 7, 8\} \quad Q = \{5, 7, 8\} \quad Q = \{7, 8\} \quad Q = \{7, 8, 6\} \quad Q = \{8, 6\} \quad Q = \{6\} \quad Q = \emptyset$$