

Luiss
Libera Università Internazionale
degli Studi Sociali Guido Carli

Corso di preparazione per la selezione territoriale delle Olimpiadi di Informatica

Lezione 6. Grafi

Blerina Sinaimeri, Alessio Martino

LUISS 



Chi vuol essere PetaChad?

- Finora: Chad, GigaChad, TeraChad, PetaChad
- Oggi è il ExaChad day: potete essere 1000 volte PetaChad!

yotta	[Y]	10^{24}	= 1 000 000 000 000 000 000 000 000
zetta	[Z]	10^{21}	= 1 000 000 000 000 000 000 000 000
exa	[E]	10^{18}	= 1 000 000 000 000 000 000 000 000
peta	[P]	10^{15}	= 1 000 000 000 000 000 000 000 000
tera	[T]	10^{12}	= 1 000 000 000 000 000 000 000 000
giga	[G]	10^9	= 1 000 000 000 000 000 000 000 000
mega	[M]	10^6	= 1 000 000 000 000 000 000 000 000
kilo	[k]	10^3	= 1 000 000 000 000 000 000 000 000
hecto	[h]	10^2	= 100 000 000 000 000 000 000 000 000
deca	[da]	10^1	= 10 000 000 000 000 000 000 000 000 000

Regole delle FantaOlimpiadi

Punti bonus:

- Rispondere a domande sulla chat: +5
- Chad/Stacy per un giorno: +10
- Chad/Stacy per un giorno ringrazia professori Luiss: +10
- Superamento territoriali: +30
- Fase finale: Oro +100 , Argento +70, Bronzo +50

Punti malus:

- Stare nella chat sbagliata: -10
- Dire "Fantaolimpiadi" su chat/video: -10
- Scrivere “buon pomeriggio” in chat appena inizia la lezione: -5
- Chiedere se si può usare Python alle territoriali: -10

Programma di oggi (ambiziosissimo!)

1. Soluzione dei problemi della volta scorsa (almeno due...)
2. Introduzione ai grafi ed ordinamento topologico
3. Visite sui grafi e cammini minimi
4. Esercizi per casa

Correzione compiti a casa



Esercizi per casa

- Sui tetti di Pisa (Pre-EGOI 2022)

<https://training.olinfo.it/-/task/pre-egoi-parkour/statement>

- Truffa al sushi (Practice round OII 2020)

https://training.olinfo.it/#/task/preoii_sushi/statement

- La coda per il buffet (OII 2022)

https://training.olinfo.it/#/task/oii_coda/statement

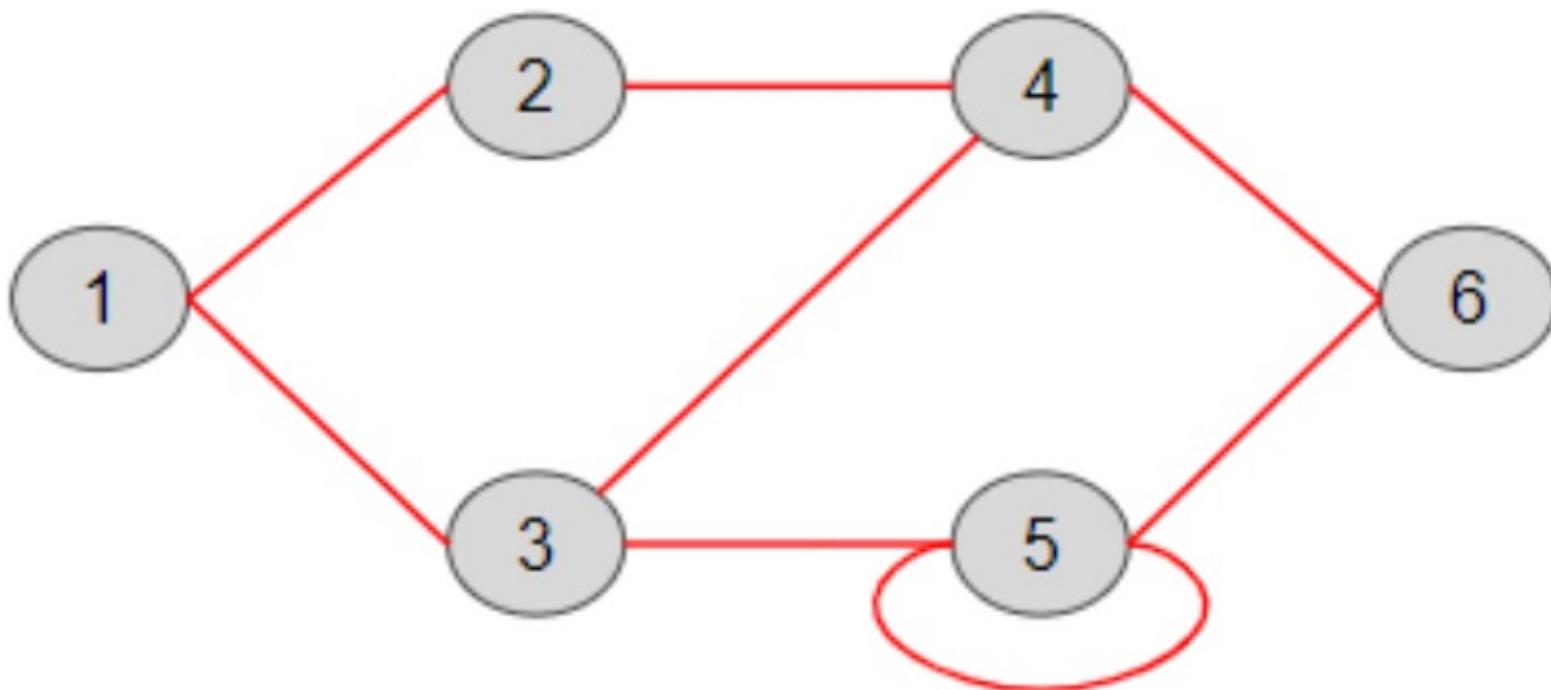
- Pub Encounter (OIS 2022)

https://training.olinfo.it/#/task/ois_pickup/statement

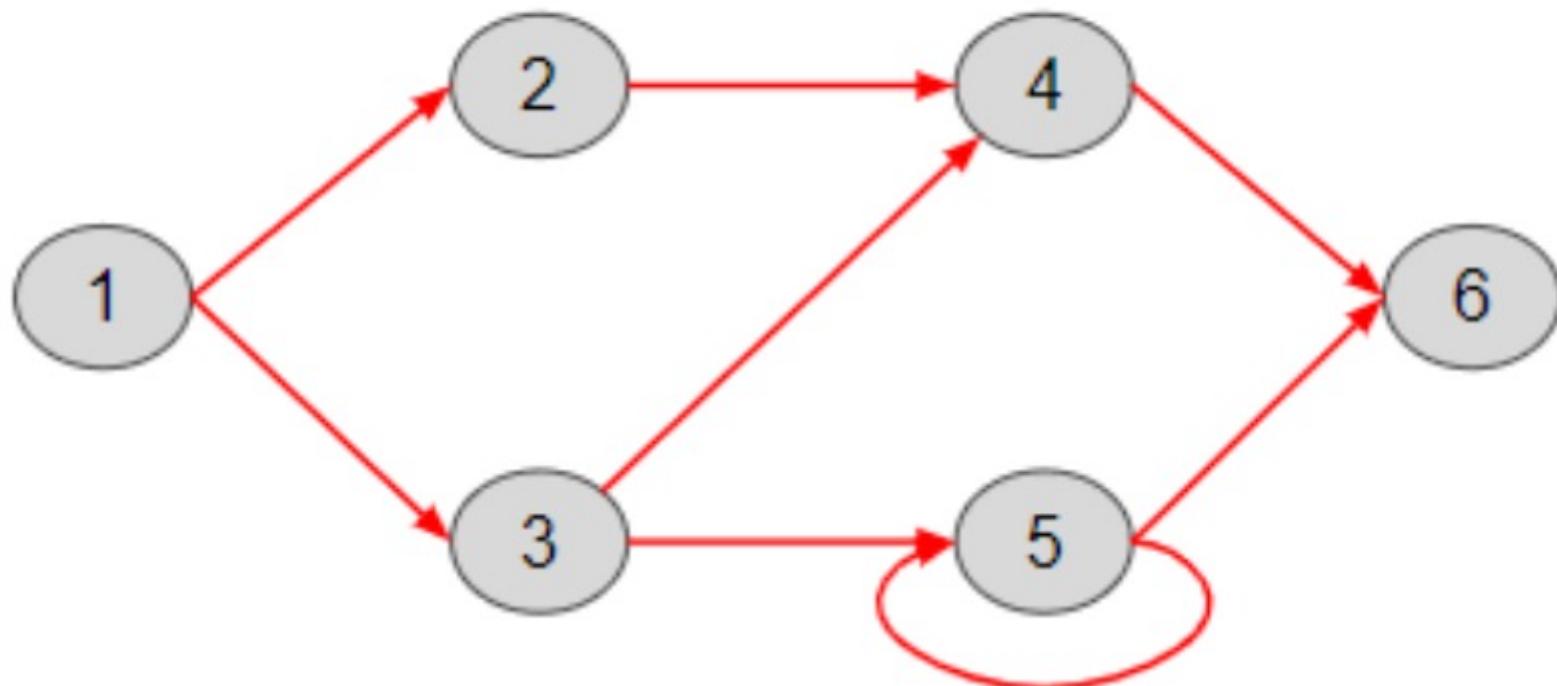
Grafi



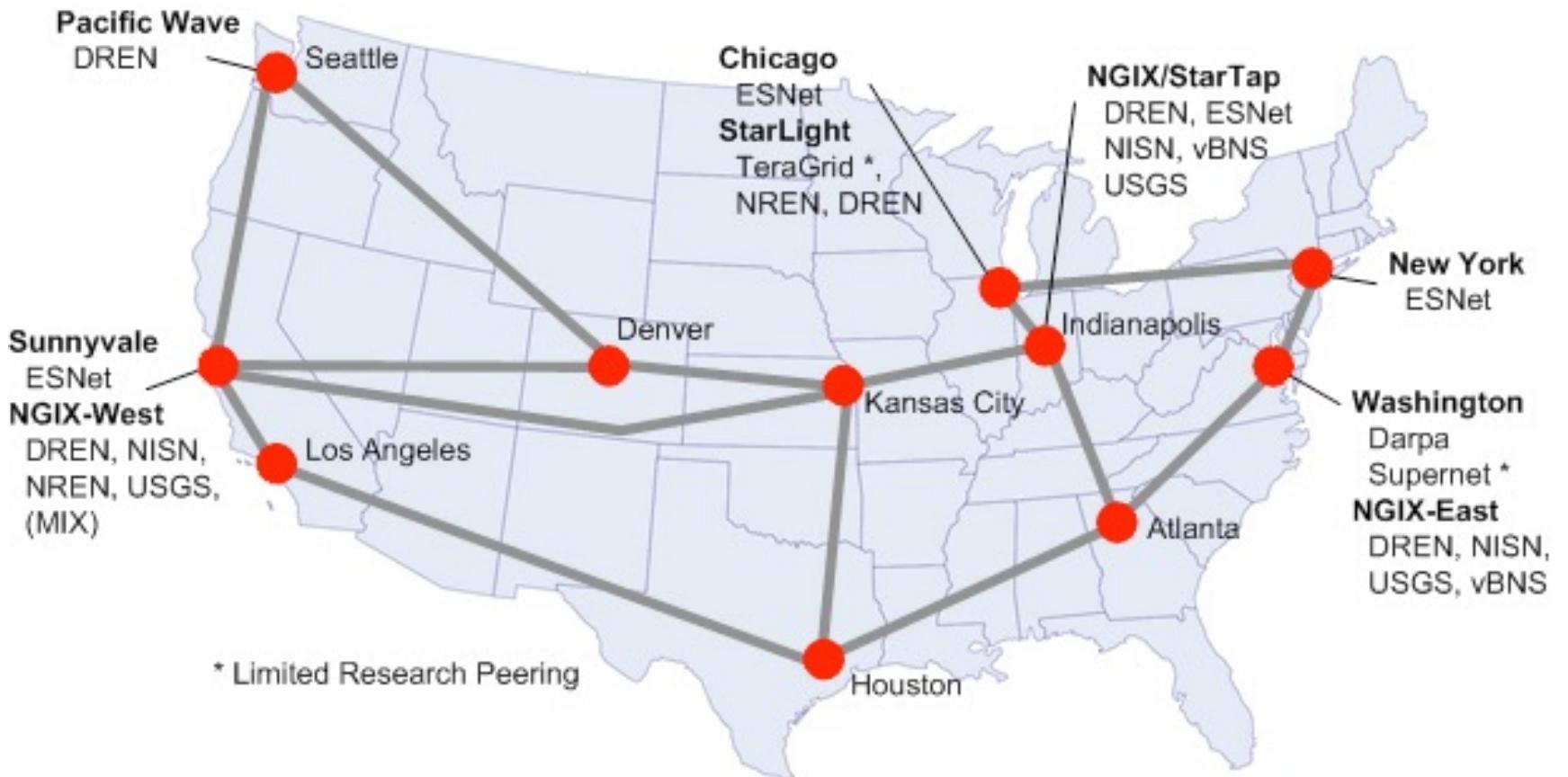
GRAFO NON ORIENTATO

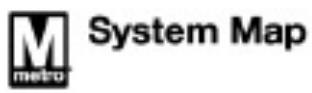


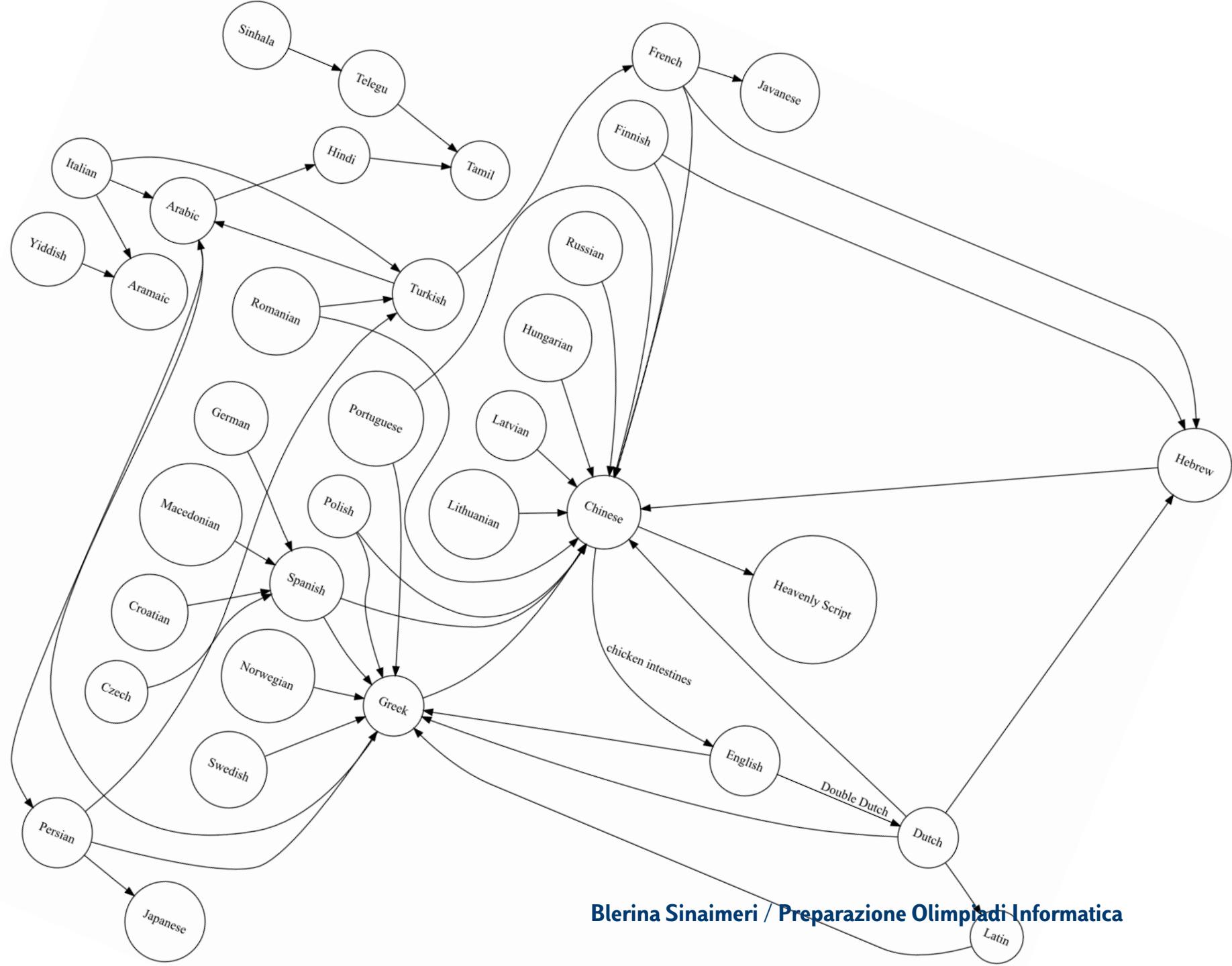
GRAFO ORIENTATO



Abilene Federal/Research Network Peers









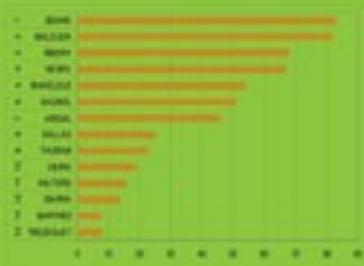
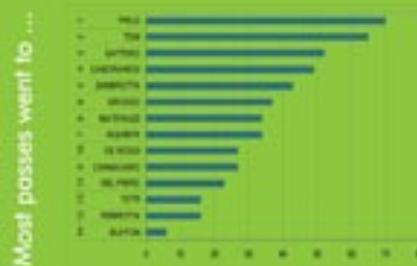
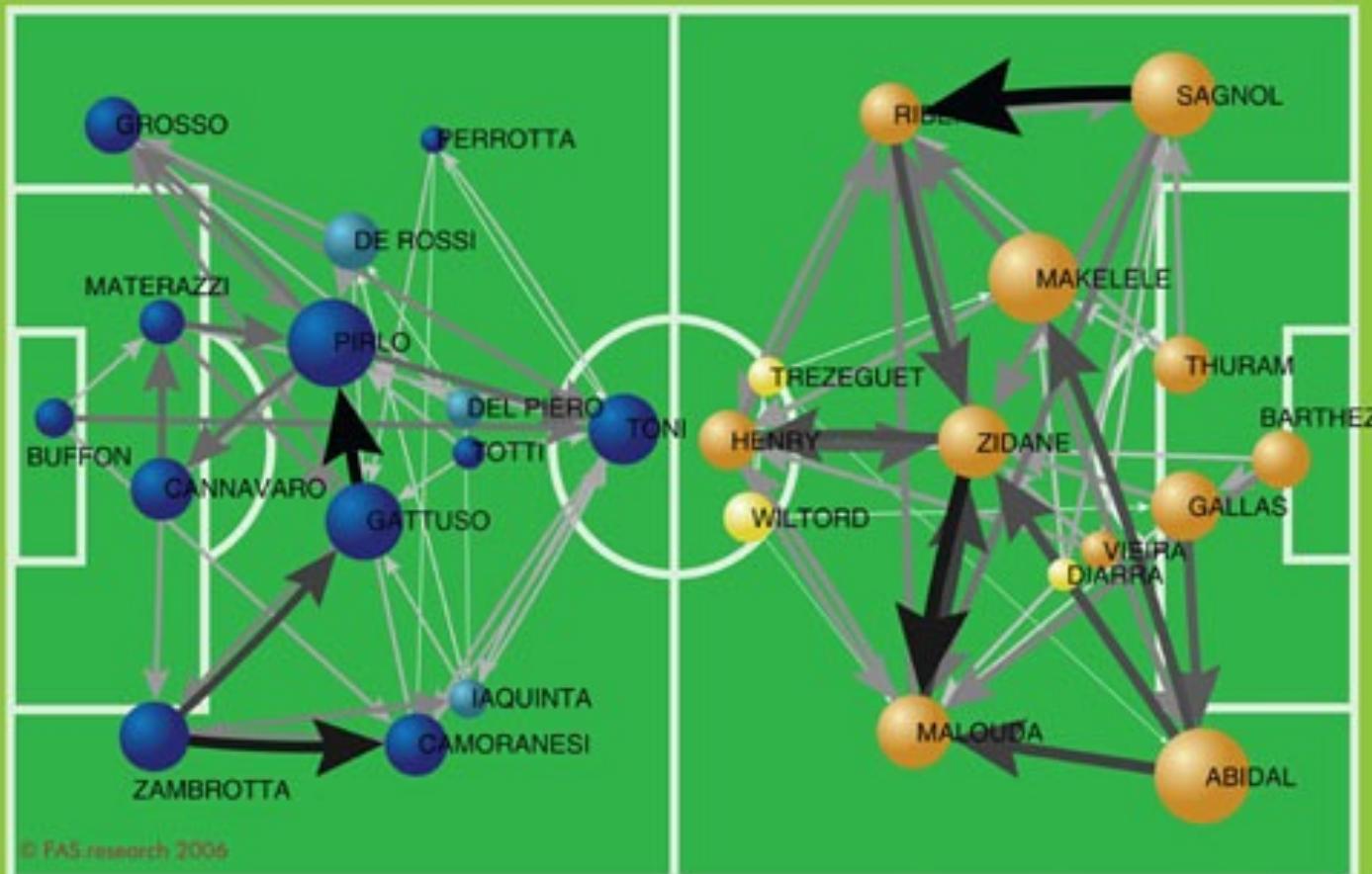
facebook

Blerina Sinaimeri / Preparazione Olimpiadi Informatica

December 13, 2010

WORLDCUP 2006 Final

ITALY - FRANCE



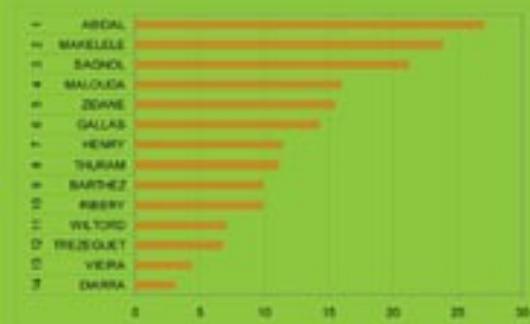
This network shows the passes from every player to those three team-mates, he passes to most frequently.

Strength of arcs displays the number of passes.
Size of nodes displays the influence
(flowbetweenness) of a player.

The match was coded and analyzed
by Harald Katzmaier and Helmut
Neundlinger, Vienna, July 9th, 2006.



The most frequent (>15) passes between players

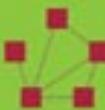


Keyplayer (hub in x % of activities)



LUI

FAS.research



Network Analysis for Science and Business

Definizione

Un grafo $G = (V, E)$ consiste in:

- un insieme V di vertici (o nodi)
- un insieme E di coppie di vertici, detti archi: ogni arco connette due vertici

Definizione

Un grafo $G = (V, E)$ consiste in:

- un insieme V di vertici (o nodi)
- un insieme E di coppie di vertici, detti archi: ogni arco connette due vertici

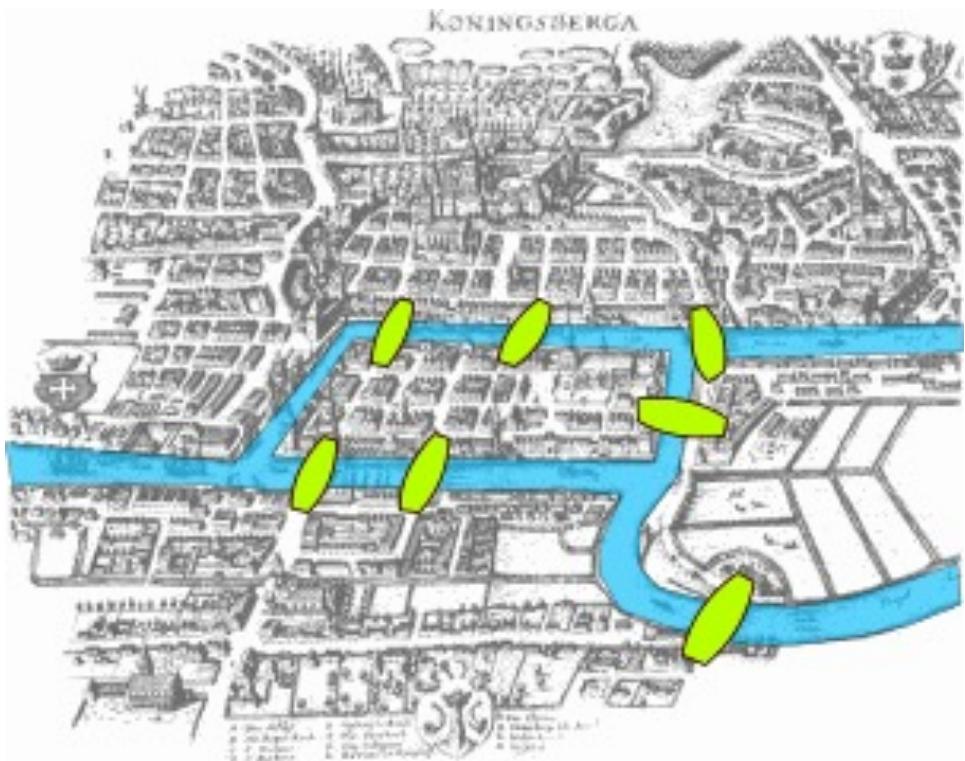
Esempio 1 Grafo di Facebook: $V = \{\text{utenti Facebook}\}$,
 $E = \{\text{amicizie su Facebook}\}$



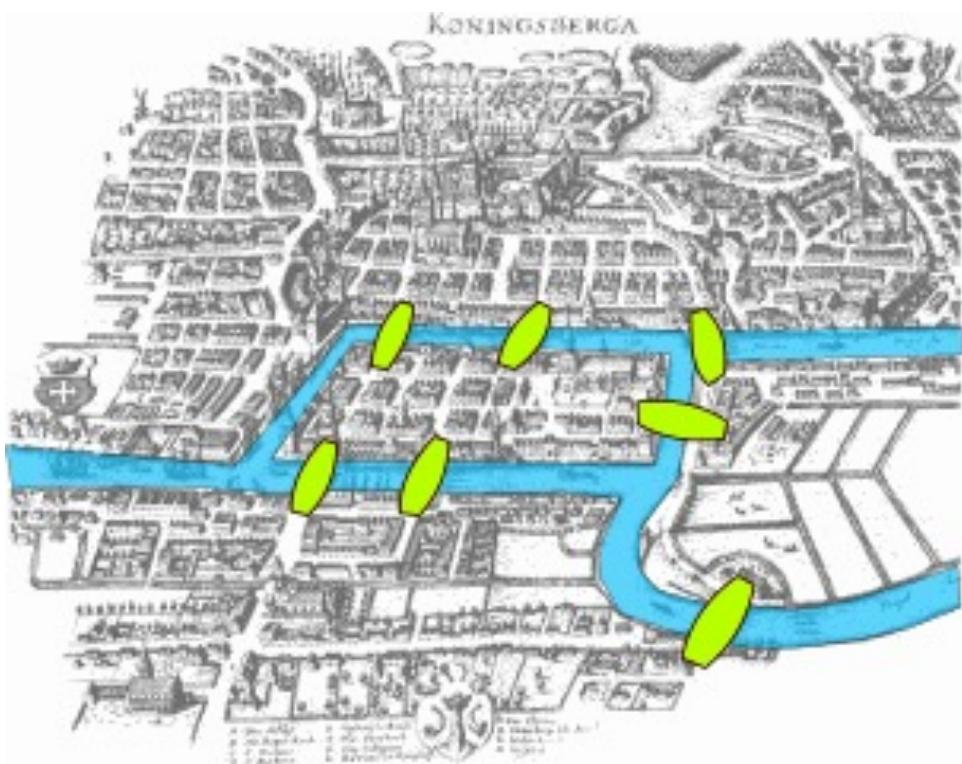
Esempio 2 Grafo di Instagram: $V = \{\text{utenti Instagram}\}$,
 $E = \{(x,y) \text{ tale che } x \text{ segue } y \text{ su Instagram}\}$



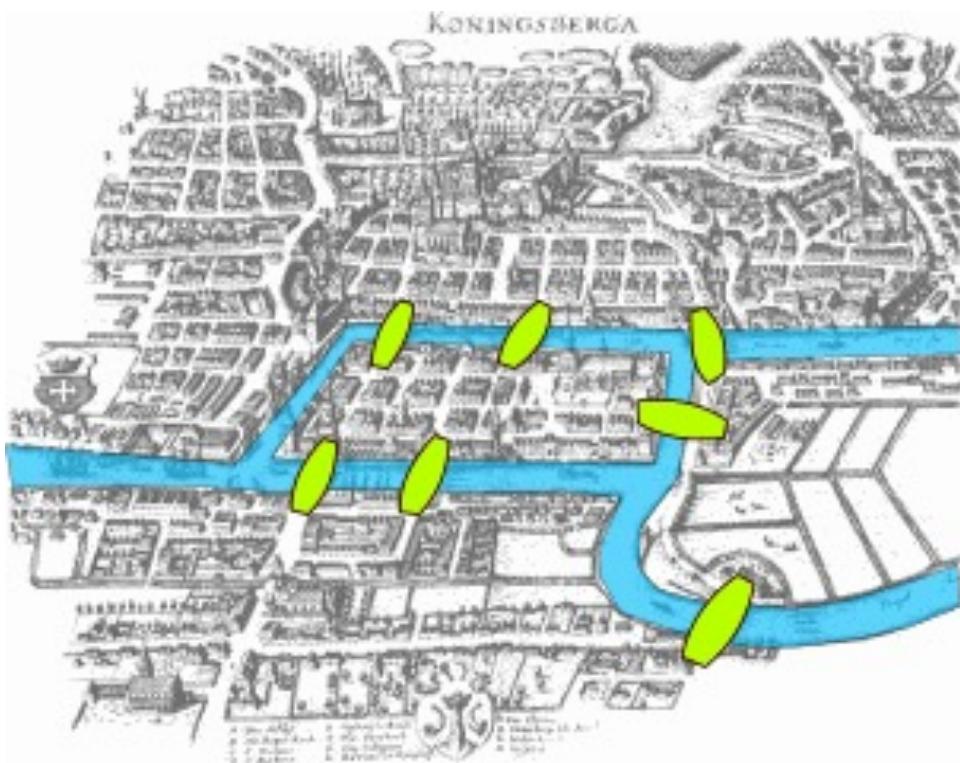
I sette ponti di Königsberg



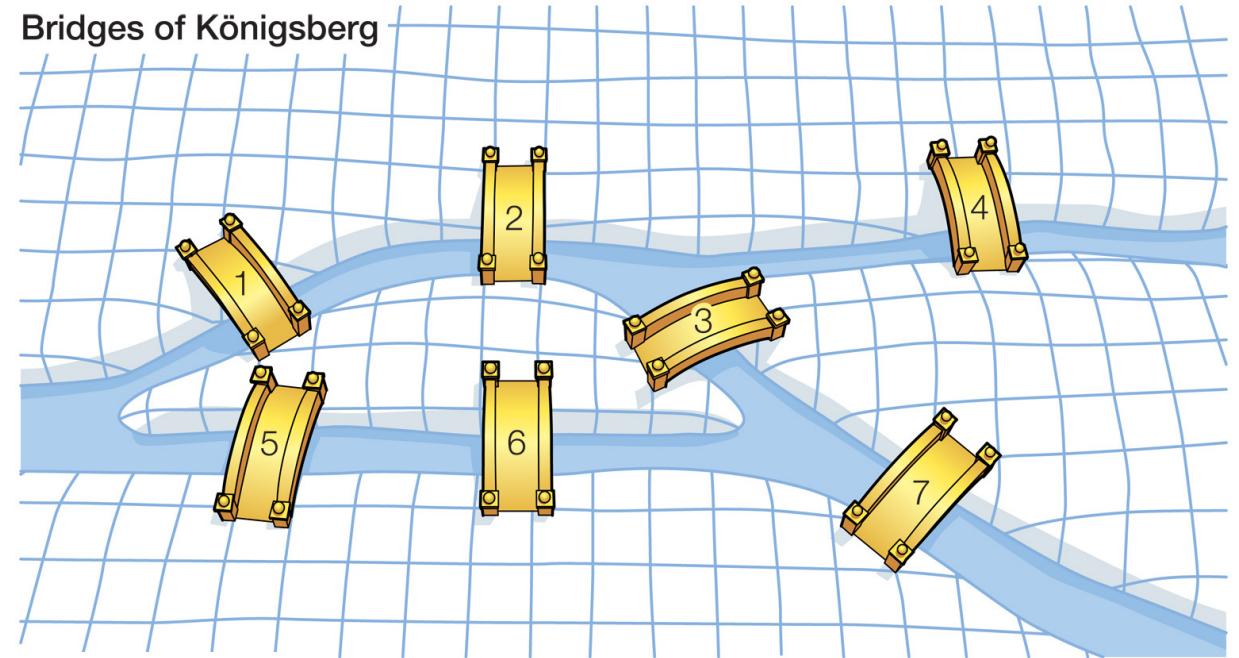
I sette ponti di Königsberg



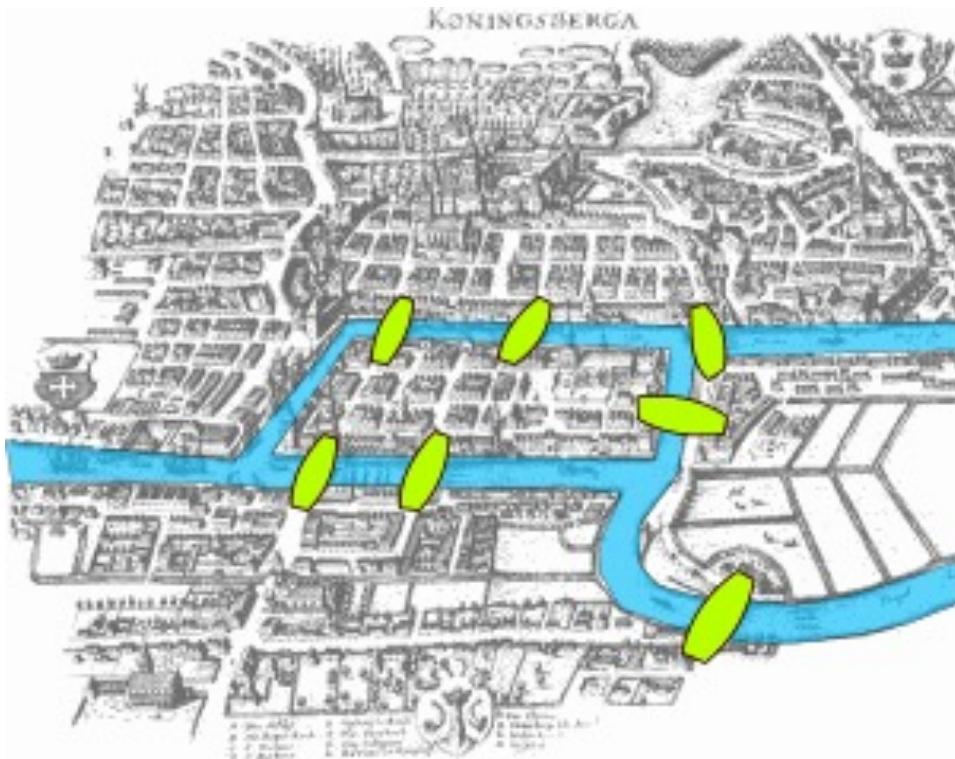
I sette ponti di Königsberg



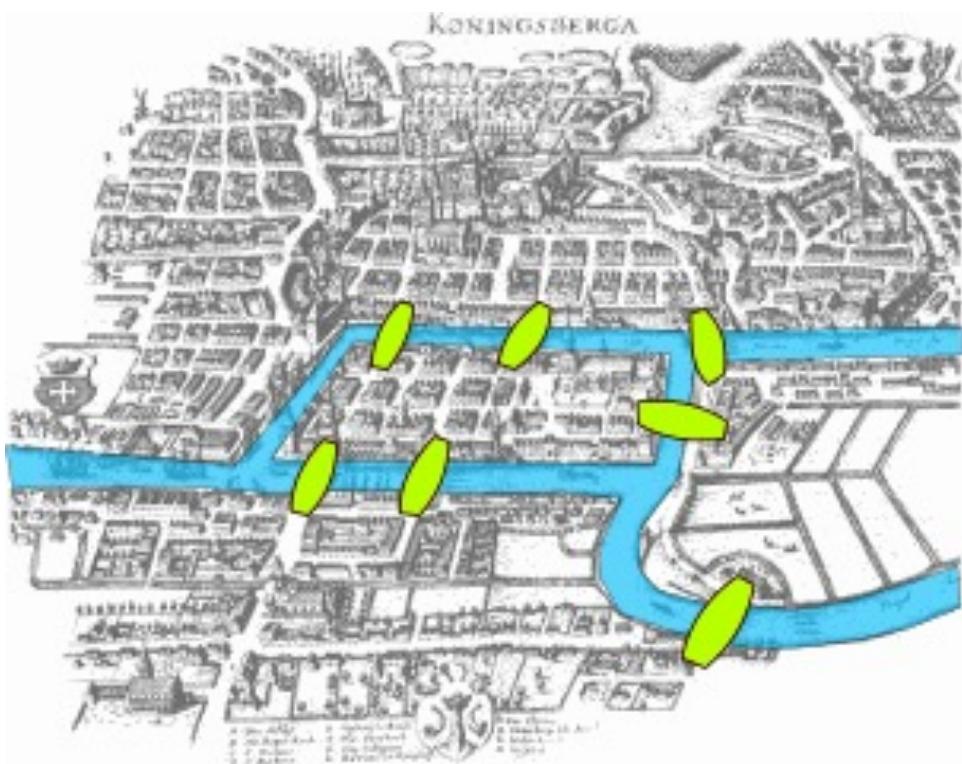
Bridges of Königsberg



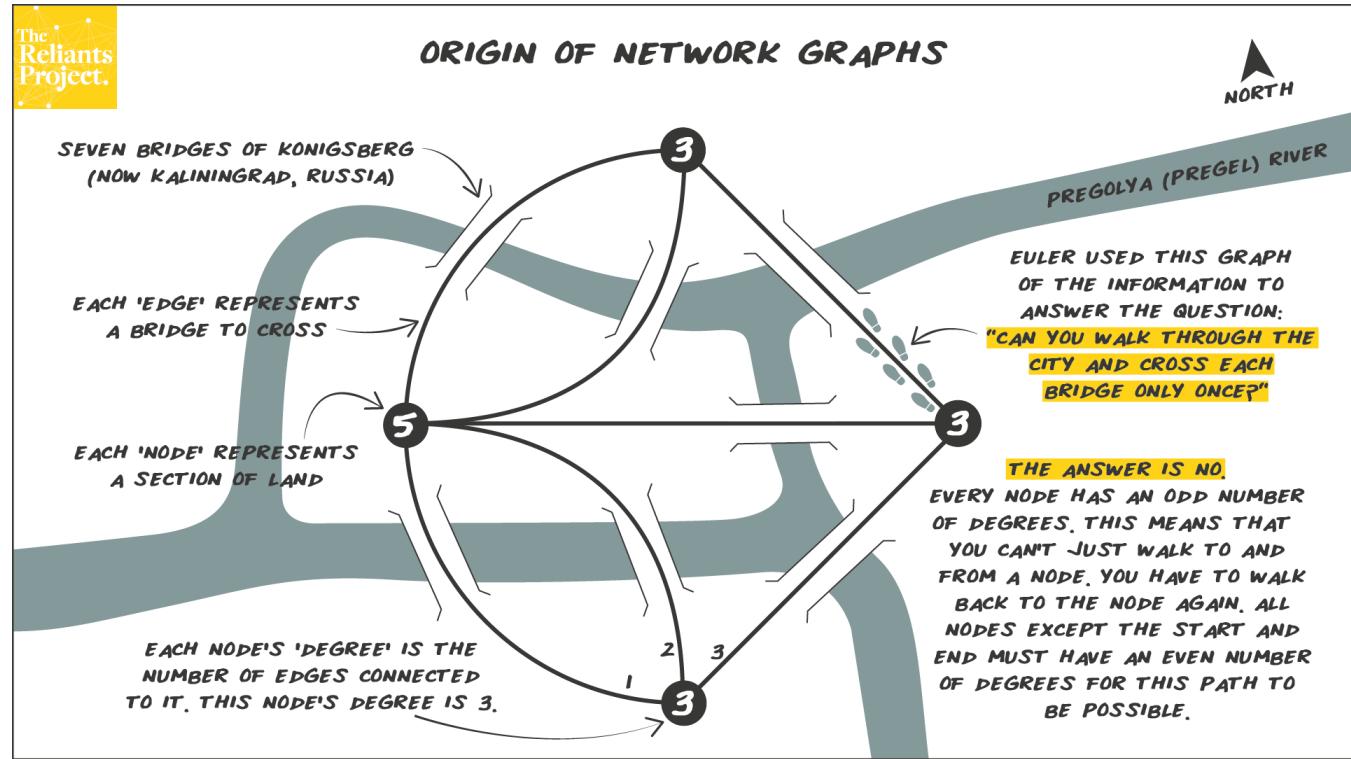
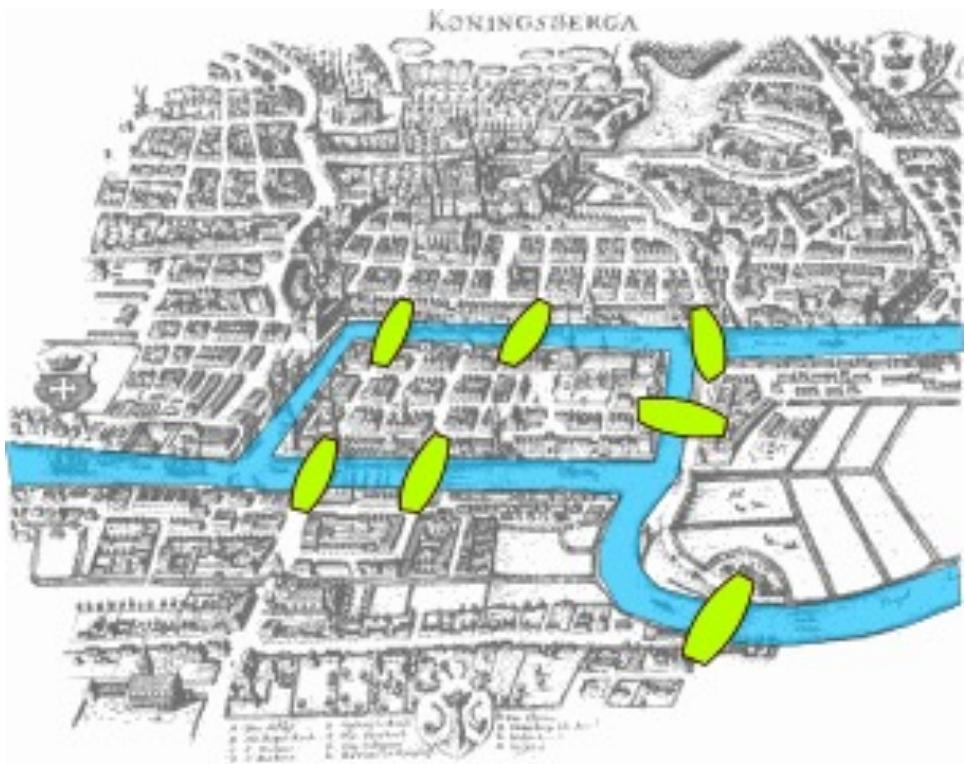
I sette ponti di Königsberg



I sette ponti di Königsberg



I sette ponti di Königsberg

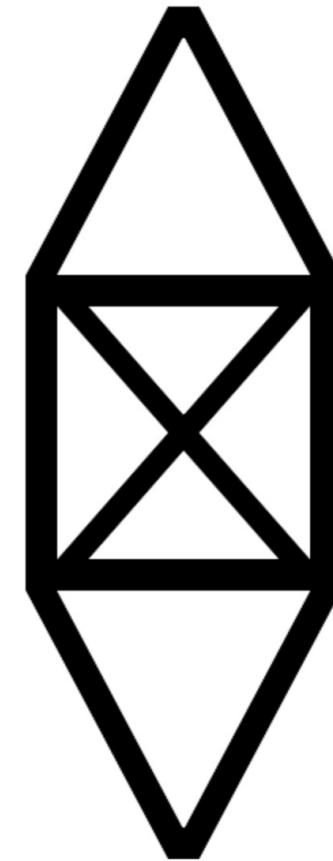


Teorema (Eulero). Un grafo è Euleriano se e solo se:

- Tutti i nodi hanno grado pari, oppure
- Soltanto due nodi hanno grado dispari.

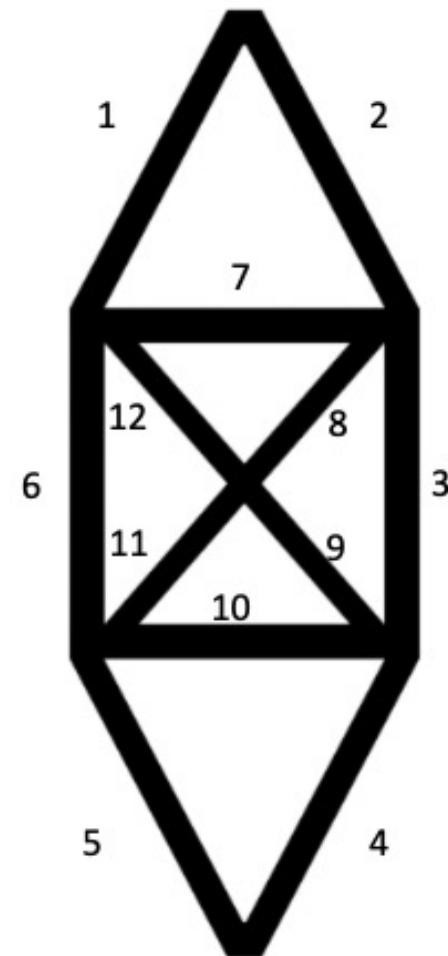
Un gioco

- Potete disegnare la figura senza mai staccare la penna dal foglio e senza ripassare su una linea già tracciata?



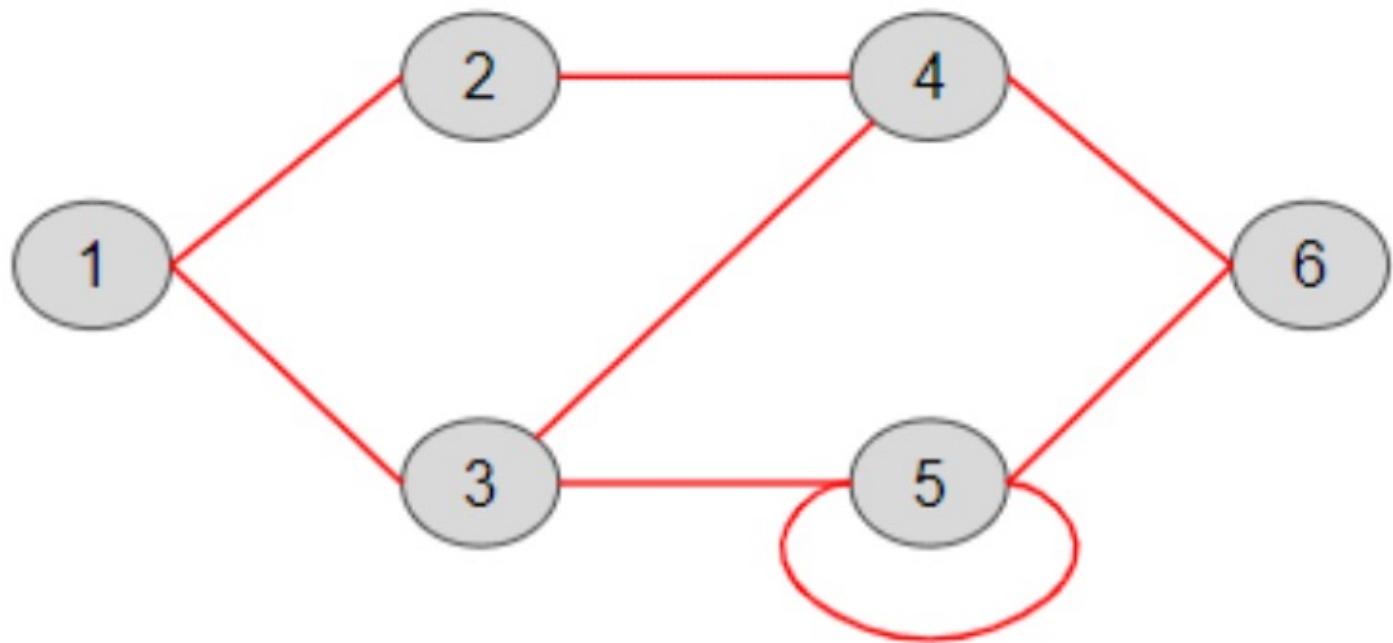
Un gioco

- Potete disegnare la figura senza mai staccare la penna dal foglio e senza ripassare su una linea già tracciata?



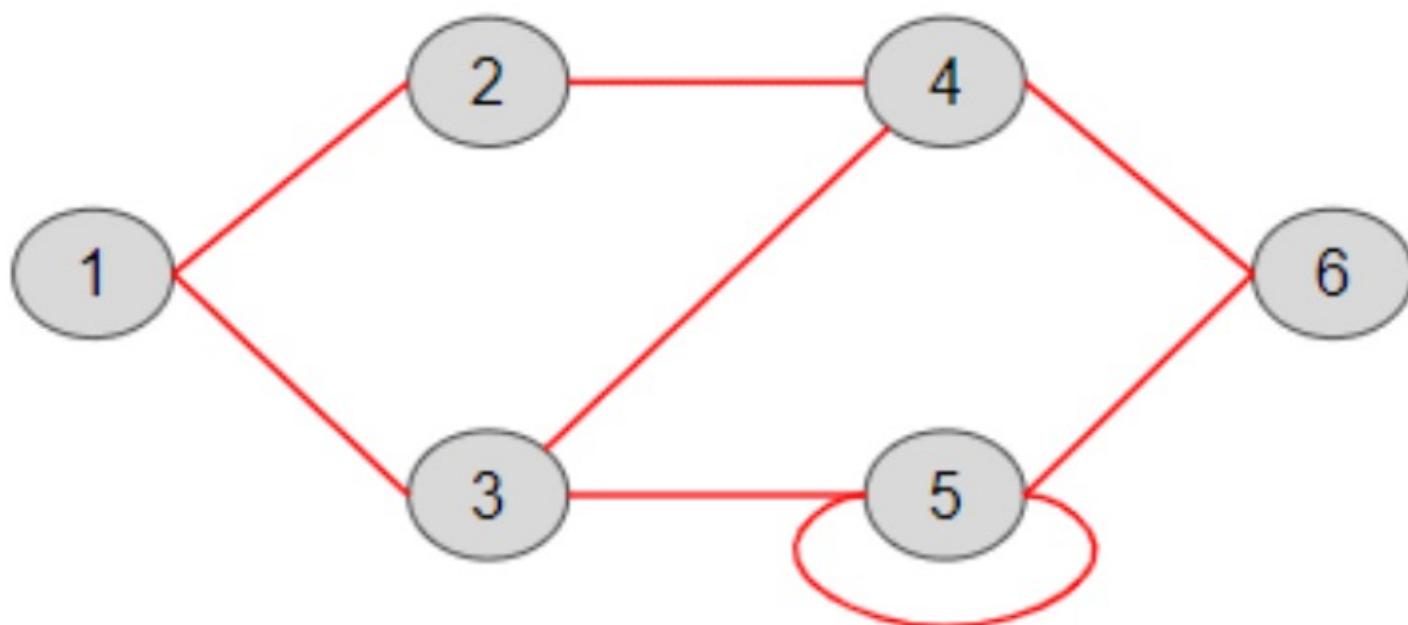
Strutture dati per rappresentare grafi

GRAFO NON ORIENTATO



NODI	LISTE DI ADIACENZA
1	2 , 3
2	1 , 4
3	1 , 4 , 5
4	2 , 3 , 6
5	3 , 5 , 6
6	4 , 5

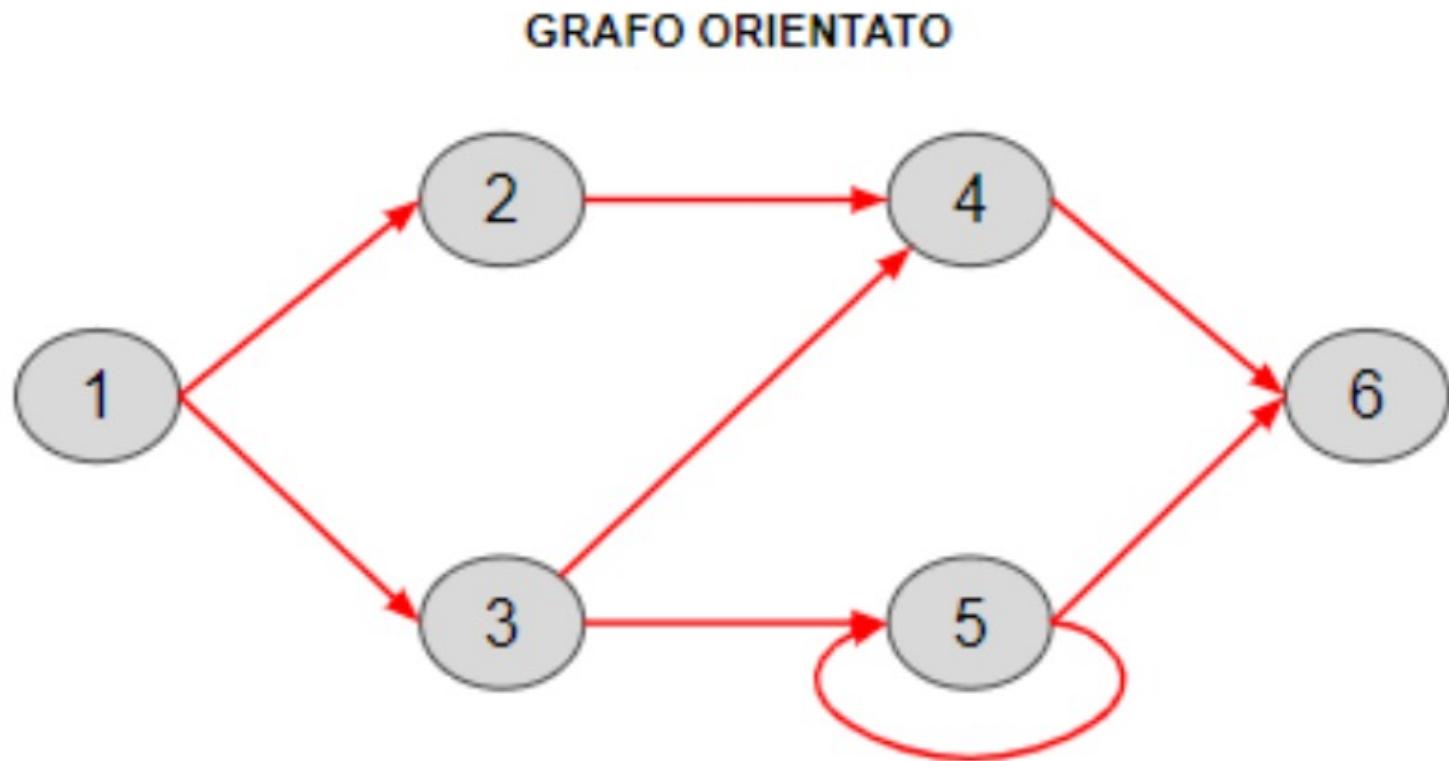
GRAFO NON ORIENTATO



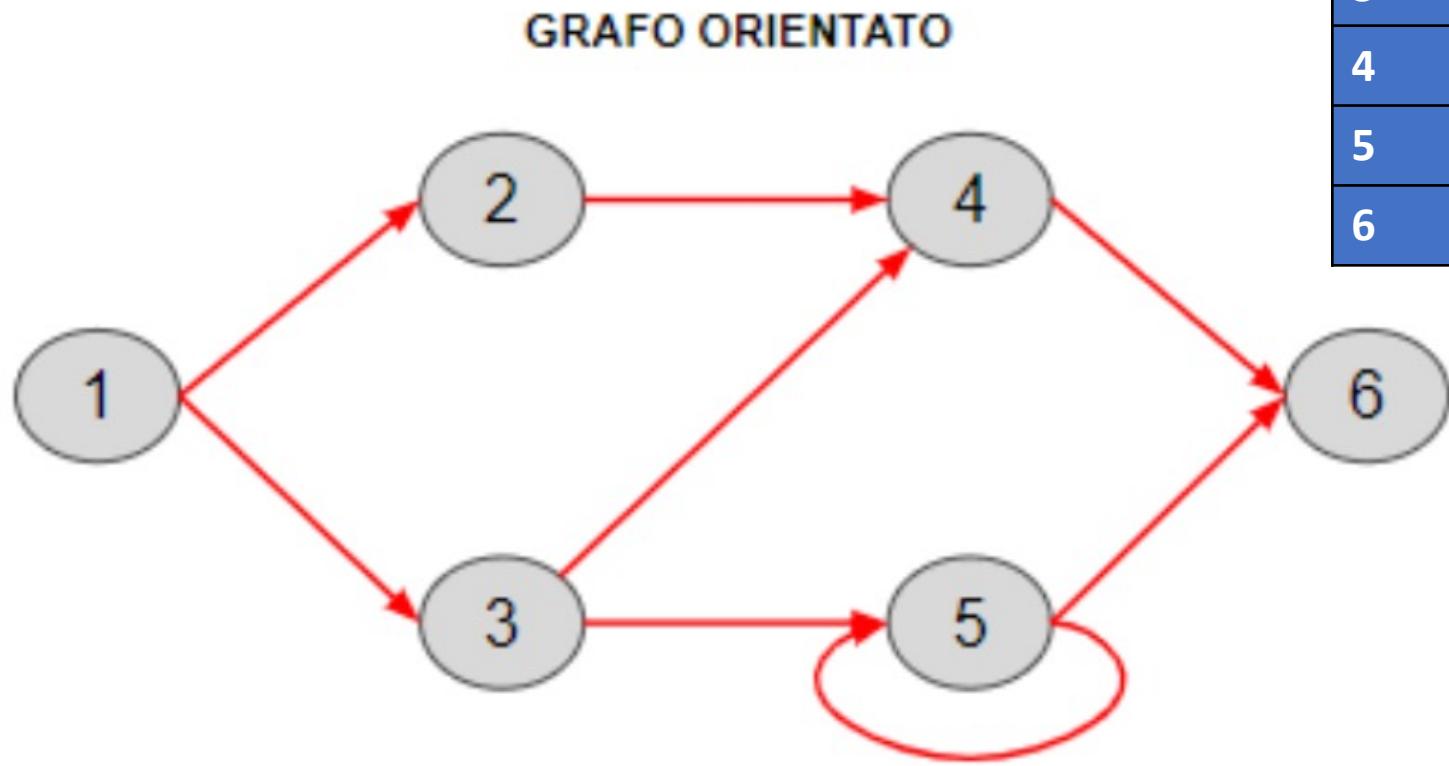
nodi destinazione

	1	2	3	4	5	6
1	0	1	1	0	0	0
2	1	0	0	1	0	0
3	1	0	0	1	1	0
4	0	1	1	0	0	1
5	0	0	1	0	1	1
6	0	0	0	1	1	0

nodi di partenza



Nodi	Liste di adiacenza
1	2, 3
2	4
3	4, 5
4	6
5	5, 6
6	-

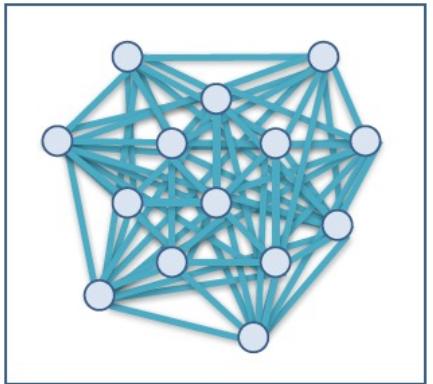


	1	2	3	4	5	6
1	0	1	3	0	0	0
2	0	0	0	1	0	0
3	0	0	0	1	1	0
4	0	0	0	0	0	1
5	0	0	0	0	1	1
6	0	0	0	0	0	0

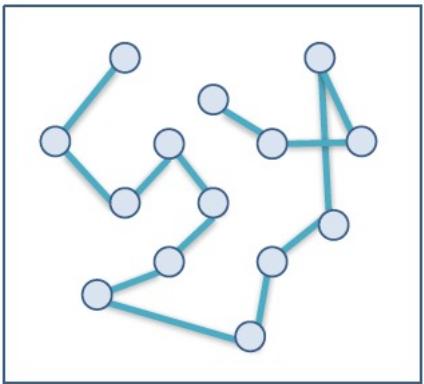
Grafi particolari



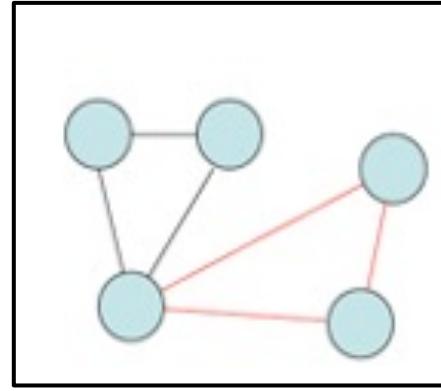
Grafi particolari



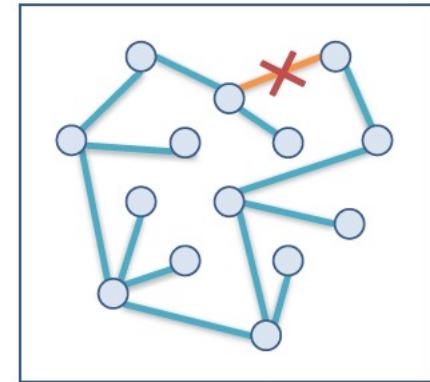
Un grafo completo (**clique**) contiene tutti gli archi possibili



Un **cammino** è una sequenza di nodi connessi con archi.



Un **ciclo**.



Un **albero** è un grafo connesso che NON contiene cicli.

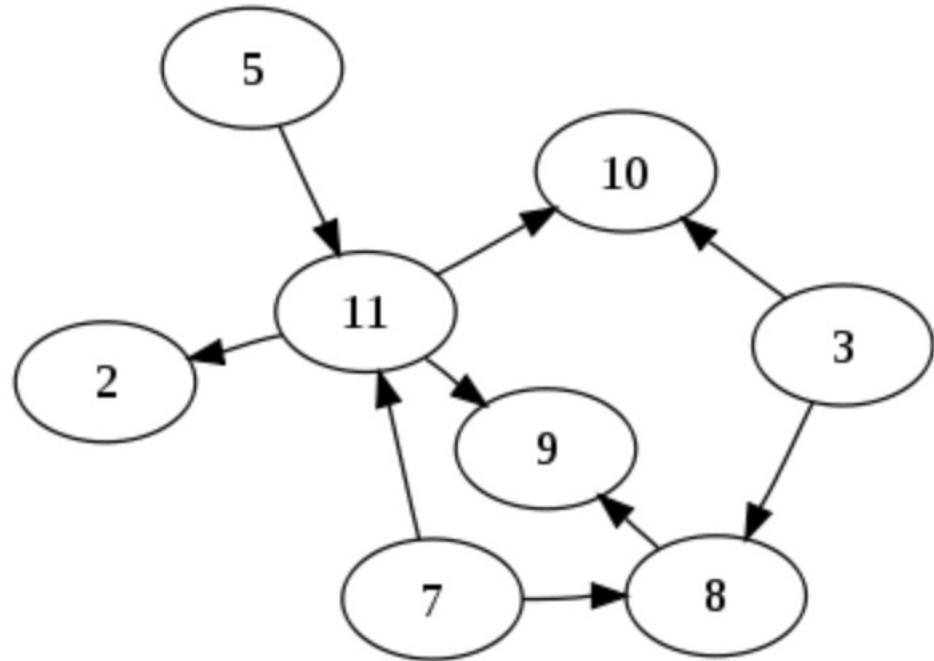
Principali algoritmi per grafi



Principali algoritmi per grafi

1. Ordinamento topologico (grafi orientati senza cicli)
2. Visita in profondità (DFS) e in ampiezza (BFS)
3. Cammini minimi (Dijkstra)

Grafo diretto aciclico (DAG: Directed Acyclic Graph)

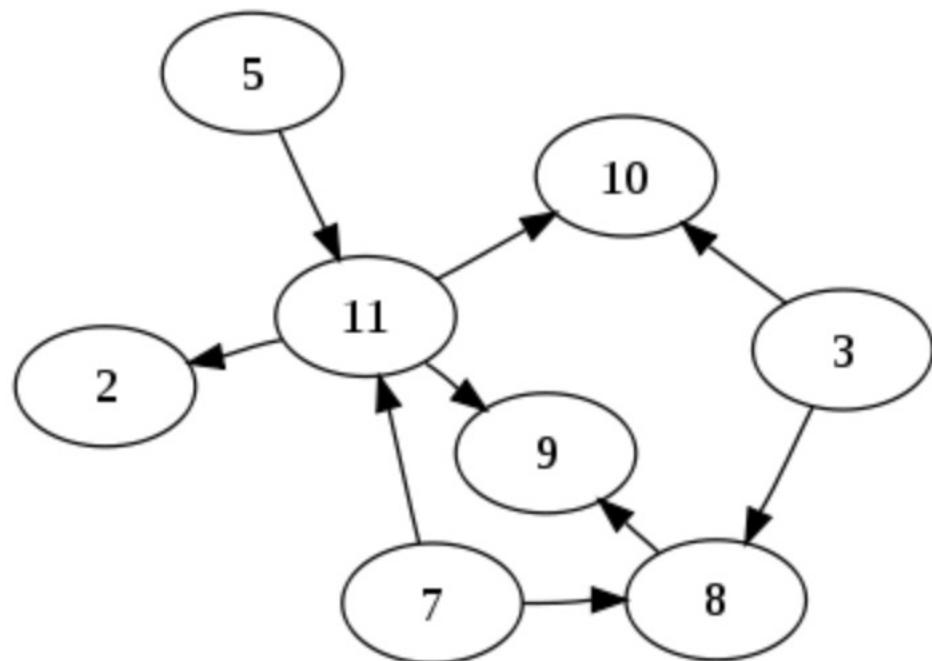


Grafo orientato che non contiene cicli:
comunque scegliamo un nodo del grafo,
non possiamo mai ritornare allo stesso
nodo (percorrendo gli archi del grafo)

Un DAG ammette un ordinamento
parziale tra i nodi

Ordinamento topologico

- Se esiste un arco da A a B , allora il nodo A comparirà nell'ordinamento prima di B
- L'ordinamento topologico di un DAG esiste sempre, e non è necessariamente unico



5 3 7 8 11 9 10 2

7 5 11 2 3 10 8 9

7 3 5 11 10 8 2 9

...

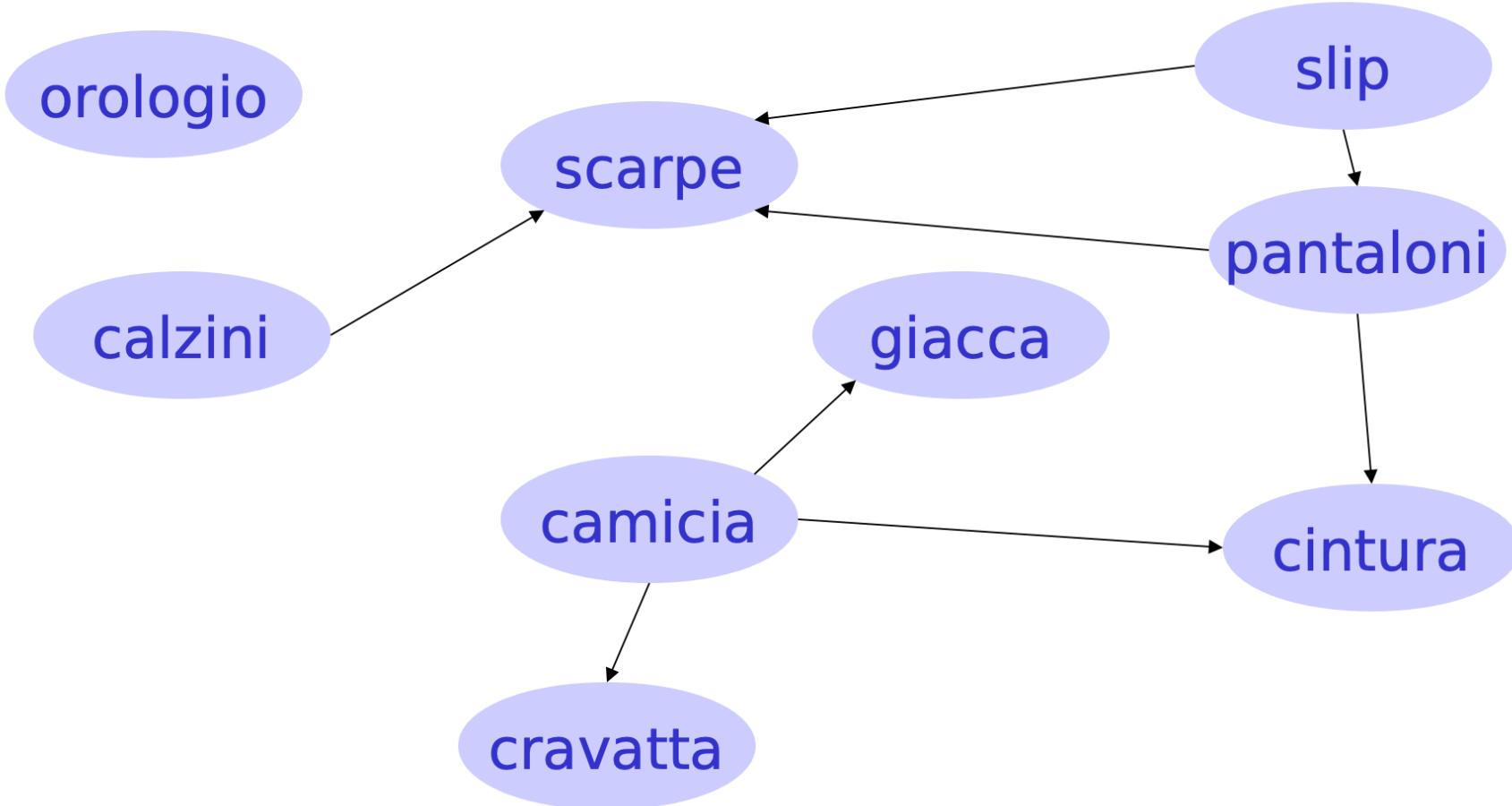
Torero Escamillo (torero)

Difficoltà D = 2

Descrizione del problema

Il celebre torero Escamillo deve indossare il proprio costume prima di entrare nell'arena. Egli è costretto a rispettare un dato numero di precedenze, indossando certi indumenti prima di altri, mentre alcuni indumenti possono essere liberamente indossati in un ordine qualsiasi. Per esempio, le "medias" (calze) vanno indossate prima delle "zapatillas" (scarpe), ma non vi è alcun vincolo sull'ordine con cui indossare la "chaquetilla" (giacca) e la "montera" (cappello). Il costume di Escamillo è particolarmente raffinato ed elaborato e si compone di N indumenti. Sfortunatamente, Carmen non ha ancora consegnato uno degli N indumenti necessari alla vestizione di Escamillo. Aiutalo a vestirsi il più possibile, calcolando il massimo numero di indumenti che può indossare in attesa che Carmen gli consegna l'indumento mancante.

Grafo diretto aciclico (DAG: Directed Acyclic Graph)



Dati di input

Il file `input.txt` contiene nella prima riga una tripla di interi, separati da uno spazio: l'intero positivo N che indica il numero di indumenti per la vestizione di Escamillo, dove gli indumenti sono numerati da 1 a N ; l'intero positivo M che indica il numero di precedenze tra coppie di indumenti da rispettare durante la vestizione; l'intero Q , compreso tra 1 e N , che indica l'indumento non ancora consegnato da Carmen. Ognuna delle successive M righe contiene una coppia di interi, compresi tra 1 e N , separati da uno spazio. Tale coppia di interi I e J rappresenta la precedenza in cui l'indumento numero I deve essere indossato prima dell'indumento numero J .

Dati di output

Il file `output.txt` è composto da una riga contenente un solo intero, che rappresenta il massimo numero di indumenti che Escamillo riesce a indossare in attesa dell'indumento Q che Carmen deve ancora consegnargli.

Assunzioni

- $1 < N < 100000$.
- $1 < M < 100000$.
- $1 \leq Q \leq N$.

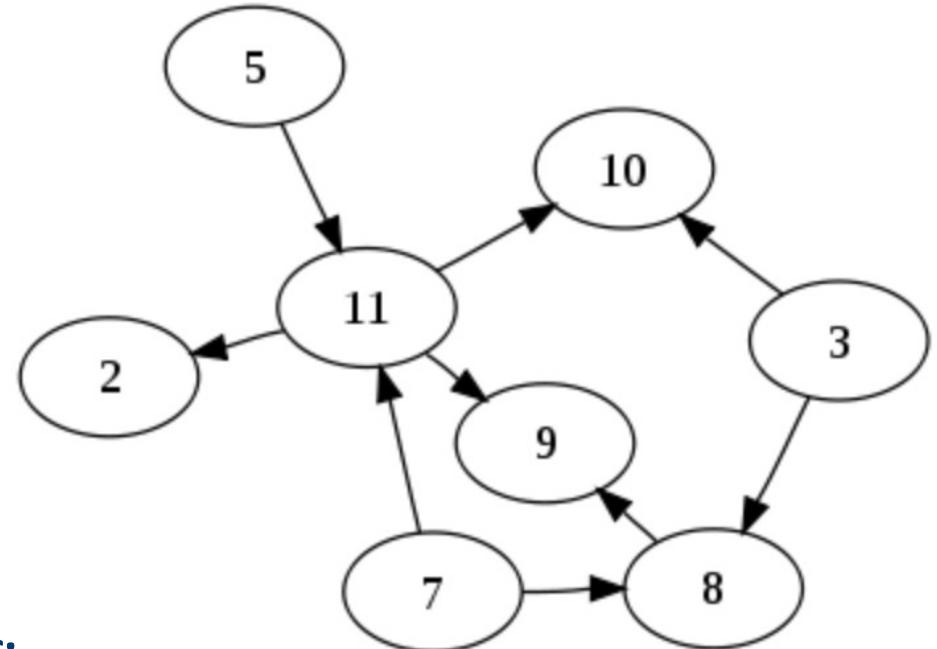
Esempi di input/output

File input.txt	File output.txt
4 5 3 1 3 1 4 3 2 3 4 4 2	1

Algoritmo di ordinamento topologico

1. Trova un nodo che non ha archi entranti, ovvero un nodo sorgente (esiste sempre: perché?)
2. Aggiungilo all'ordinamento topologico
3. Rimuovi dal grafo il nodo e tutti i suoi archi (uscenti)
4. Ripeti... (trova un altro nodo sorgente, ecc)

Se vuoi l'ordinamento, al passo 2 inserisci alla fine di una lista i nodi sorgente man mano che li incontri



Algoritmo di ordinamento topologico

```
L ← lista vuota che conterrà gli elementi ordinati
S ← insieme di nodi senza archi entranti
while S non è vuoto do
    rimuovi un vertice u da S
    inserisci u in L
    for each vertice v con un arco e da u a v do
        rimuovi arco e dal grafo
        if v non ha altri archi entranti then
            inserisci v in S
    if il grafo ha ancora archi then
        ritorna un errore (il grafo ha almeno un ciclo)
else
    ritorna L (l'ordinamento topologico)
```

Complessità lineare:

$O(n+m)$

dove:

- n è il numero di nodi
- m è il numero di archi

**Un problema già visto
(dalle Territoriali 2015):
Rispetta i versi**

<https://training.olinfo.it/#/task/disuguaglianze/statement>



Rispetta i versi (disuguaglianze)

Limite di tempo: 1.0 secondi

Limite di memoria: 256 MiB

Difficoltà: 2

Gabriele ha un nuovo rompicapo preferito, chiamato “Rispetta i versi”. Si tratta di un solitario giocato su una griglia formata da N caselle separate da un simbolo di disuguaglianza; in figura è mostrato un esempio con $N = 6$.



L’obiettivo del gioco è quello di riempire le celle vuote con tutti i numeri da 1 a N (ogni numero deve comparire esattamente una volta), in modo da rispettare le disuguaglianze tra caselle adiacenti. Per la griglia della figura, una delle possibili soluzioni al rompicapo è la seguente:



Dati di input

Il file `input.txt` contiene due righe di testo. Sulla prima è presente l'intero N , il numero di caselle del gioco. Sulla seconda è presente una stringa di $N - 1$ caratteri, ognuno dei quali può essere solo `<` o `>`, che descrive i vincoli tra le caselle, da sinistra a destra.

Dati di output

Il file `output.txt` contiene su una sola riga una qualunque permutazione dei numeri da 1 a N - separati tra loro da uno spazio - che risolve il rompicapo. I numeri corrispondono ai valori scritti nelle caselle, leggendo da sinistra verso destra.

Assunzioni

- $2 \leq N \leq 100\,000$.
- Nel 30% dei casi, il valore di N non supera 10.
- Nel 60% dei casi, il valore di N non supera 20.
- Si garantisce l'esistenza di almeno una soluzione per ciascuno dei casi di test utilizzati nella verifica del funzionamento del programma.

Esempi di input/output

input.txt	output.txt
6 <><>	2 5 1 3 6 4
5 >><<	5 3 1 2 4
8 >><>><>	6 5 4 7 3 2 8 1

Grafi vs. Alberi

Gli alberi sono una categoria particolare di grafi

- Entrambi sono composti da **nodi** e **archi**
- In entrambi, ogni nodo può avere un qualunque numero di connessioni
 - Se l'albero è binario, c'è un vincolo (max 2 figli per ogni nodo)
- Nei grafi, **esistono cicli**
 - Negli alberi non possono esistere cicli (non si torna mai dai figli al padre)
- Negli alberi, esiste un **nodo “radice” univoco**
 - Nei grafi ogni nodo può essere una “radice”

Breadth First Search



Breadth First Search

Descrizione ad alto livello:

1. si parte dalla radice r
2. si visitano tutti i figli di r
3. a turno, ciascun figlio diventa “radice”
4. ripetere da step #2
fino a quando non si raggiungono le foglie

... o in altre parole:

un nodo nel livello i può essere visitato se e soltanto se
tutti i nodi al livello $i-1$ sono stati visitati

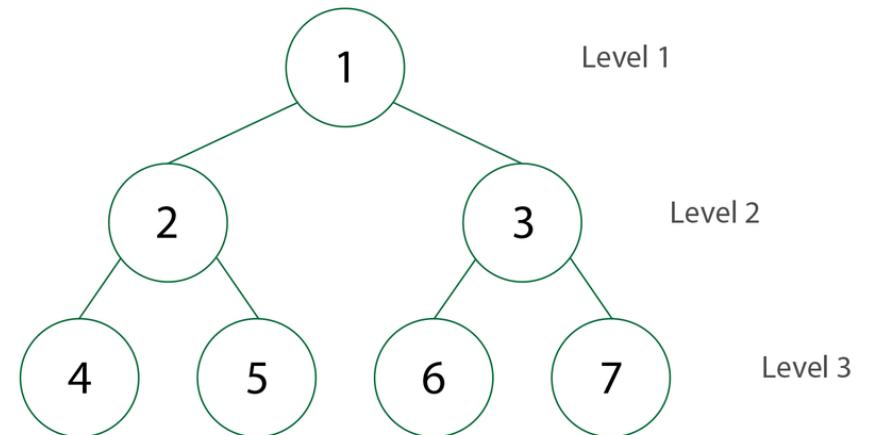


Figura 3: Ricerca in ampiezza (Breadth First Search)

Breadth First Search

Versione iterativa (generalizzata):

```
function iterative_BFS(graph G, node n):
    C = empty queue
    visited = empty list
    enqueue n in C
    add n to visited
    while C is not empty:
        dequeue n from C
        for each node v adjacent to n:
            if v not in visited:
                add v to visited
                enqueue v in C
```

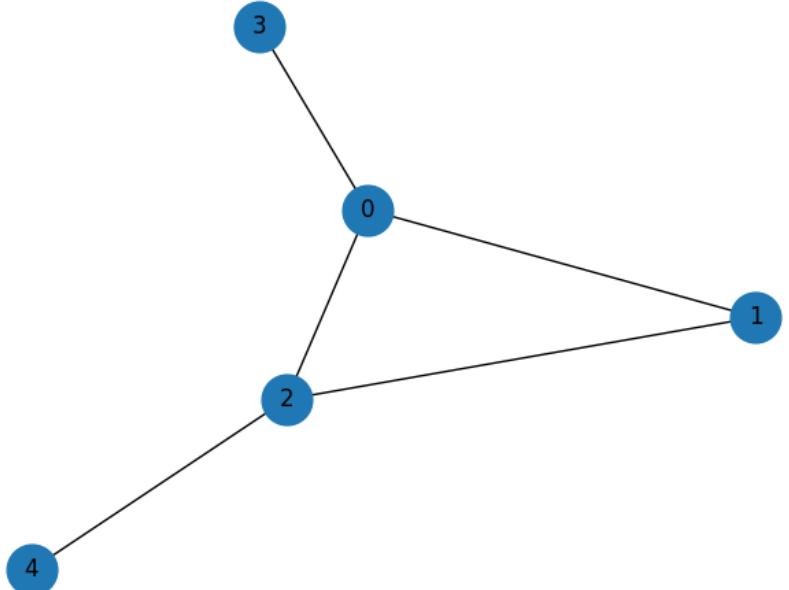


Figura 5: Esempio BFS

Breadth First Search

Inizializzazione

```
queue = [ 0 ]  
visited = [ 0 ]
```

Iterazione #1

```
queue = [ 1, 2, 3 ]  
visited = [ 0 ]
```

Iterazione #2

```
queue = [ 2, 3 ]  
visited = [ 0, 1 ]
```

Iterazione #3

```
queue = [ 3, 4 ]  
visited = [ 0, 1, 2 ]
```

Iterazione #4

```
queue = [ 4 ]  
visited = [ 0, 1, 2, 3 ]
```

Iterazione #5

```
queue = [ ]  
visited = [ 0, 1, 2, 3, 4 ]
```

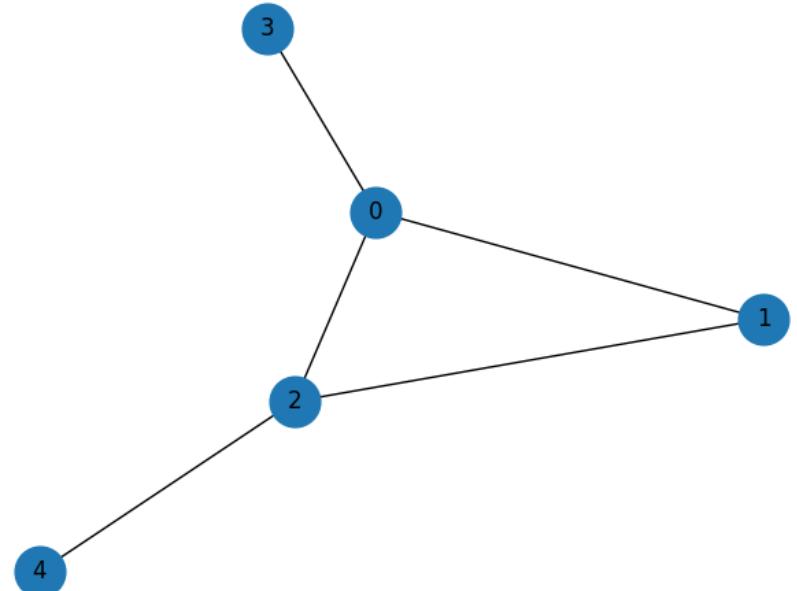


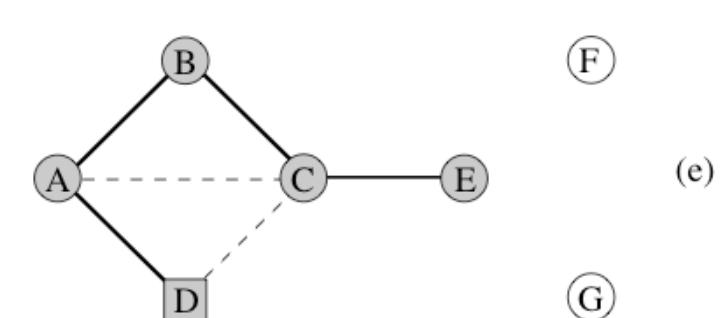
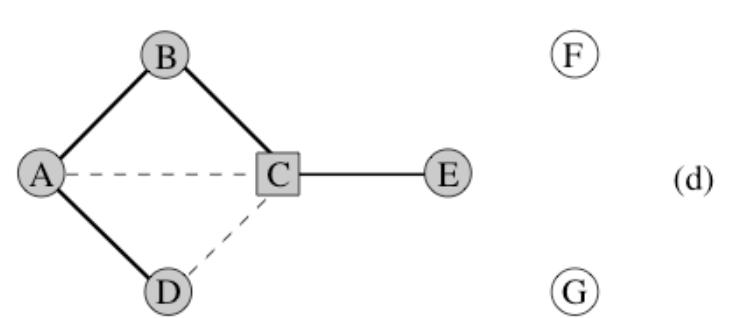
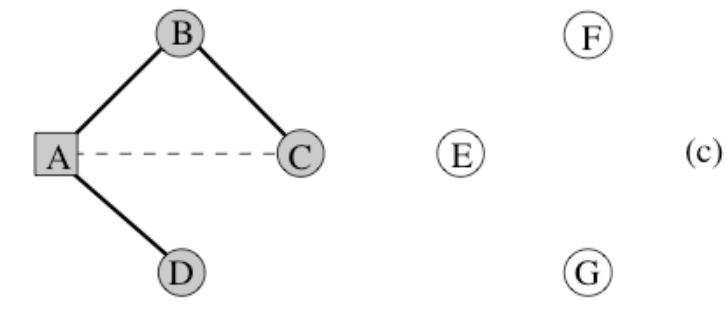
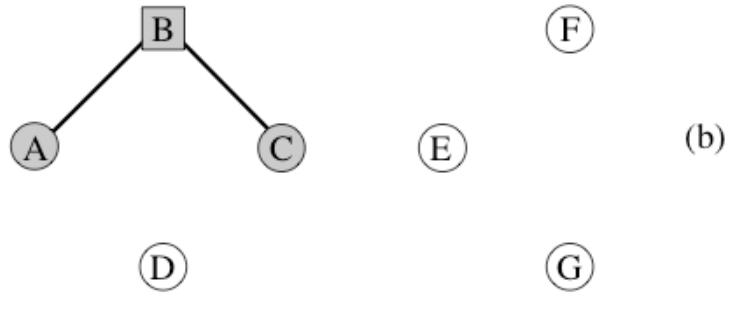
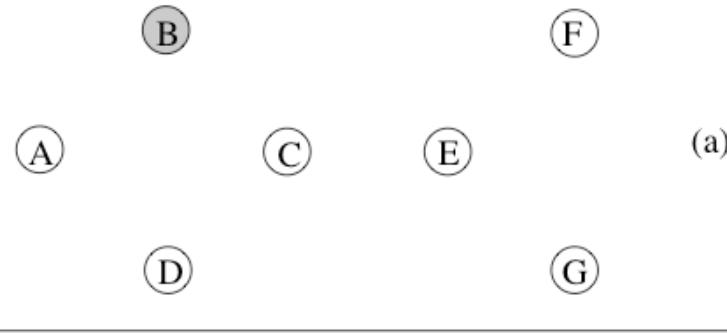
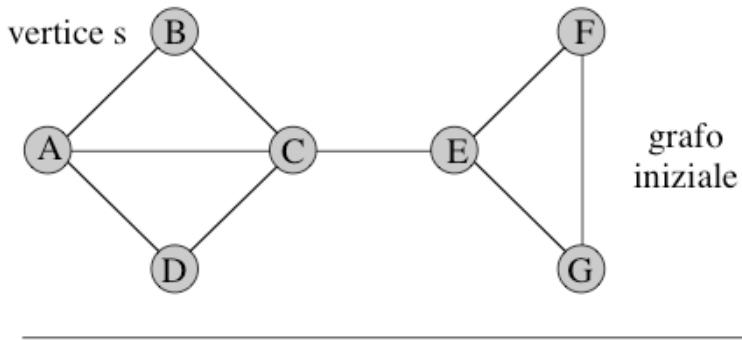
Figura 5: Esempio BFS

Breadth First Search

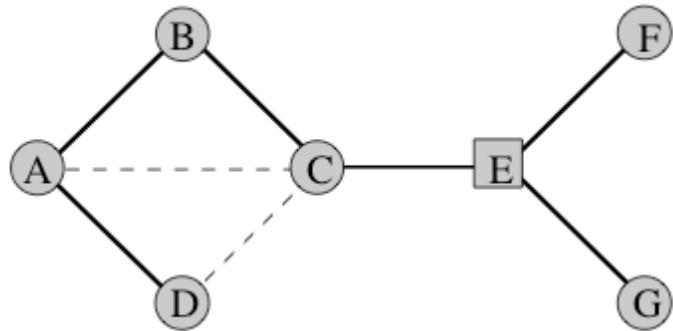
Versione iterativa (con albero di copertura):

```
function iterative_BFS(graph G, node n):
    C = empty queue
    visited = empty list
    T = empty tree
    enqueue n in C
    add n to visited
    while C is not empty:
        dequeue n from C
        for each edge (n, v) in G:
            if v not in visited:
                add v to visited
                enqueue v in C
                n is father of v in T
```

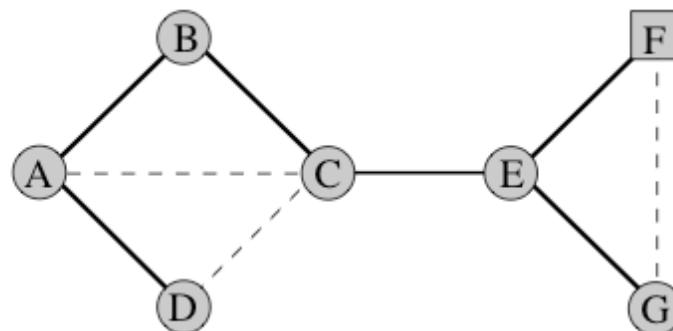
Breadth First Search



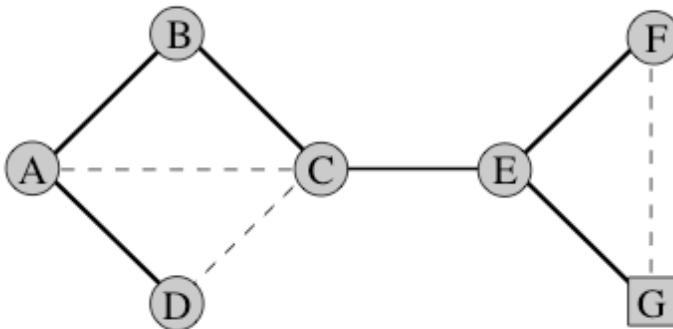
Breadth First Search



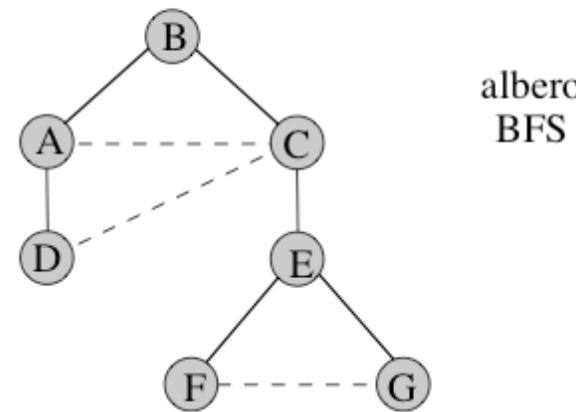
(f)



(g)



(h)



albero
BFS

Breadth First Search

Proprietà molto utile dell'albero BFS:

- Per ogni nodo v del grafo, il livello di v nell'albero di copertura BFS è pari alla distanza tra v e la sorgente s
- La [distanza](#) tra s e v è il numero di archi coinvolti nel cammino più breve da s e v
- Vedremo poi come cambia la "distanza" tra nodi nel caso di grafi [pesati](#) ...

Depth First Search



Depth First Search

Descrizione ad alto livello:

1. si parte dalla radice r
2. si visitano i nodi di figlio in figlio
fino a che non si raggiunge una foglia
3. si torna indietro all'antenato che ha figli inesplorati
 - se non esiste, l'algoritmo termina
4. si ripete da step #3 per ogni figlio

Due possibili implementazioni:

- ricorsiva
- iterativa

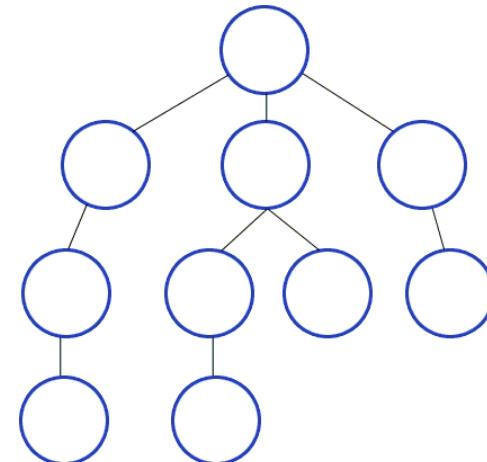


Figura 6: Ricerca in profondità (Depth First Search)

I LOVE RECURSION,
SAY IT AGAIN



Depth First Search

Versione ricorsiva per alberi binari:

```
function recursive_DFS(node n):
    if n is NULL then:
        return;
    mark n as visited;
    recursive_DFS(left child of n);
    recursive_DFS(right child of n);
```

Recursive DFS (A)

A è visitato

Recursive DFS (L)

L è visitato

Recursive DFS (E)

E è visitato

Recursive BFS (R)

R è visitato

Recursive DFS (B)

B è visitato

non ho figli a sinistra

Recursive DFS (O)

O è visitato

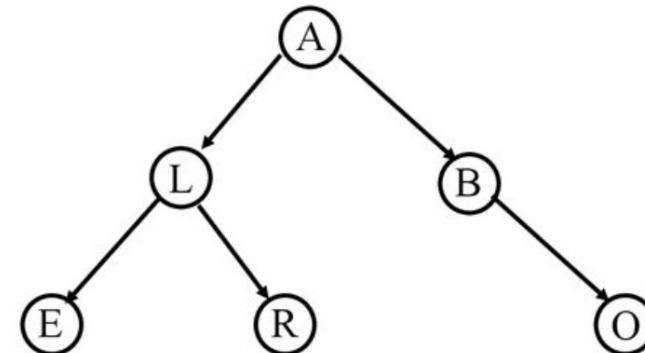


Figura 7: Esempio DFS

Depth First Search

Versione ricorsiva (generalizzata):

```
function recursive_DFS(node n):
    mark n as visited;
    for each node v adjacent to n:
        if v is not visited:
            recursive_DFS(node v);
```

Recursive DFS (0)

0 è visitato
1, 2 e 3 sono adiacenti a 0

Recursive DFS (1)

1 è visitato
2 e 0 sono adiacenti a 1, ma 0 è visitato

Recursive DFS (2)

2 è visitato
1, 0 e 4 sono adiacenti a 2, ma 1 e 0 sono visitati

Recursive BFS (4)

4 è visitato
2 è adiacente a 4, ma 2 è visitato

Recursive DFS (3)

3 è visitato

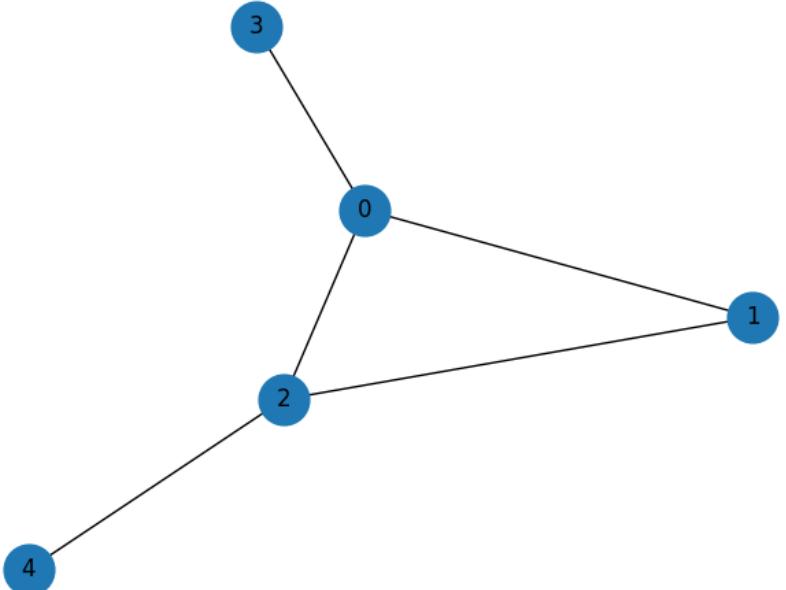


Figura 8: Esempio DFS

Depth First Search

Versione iterativa:

```
function iterative_DFS(graph G, node n):
    S = empty stack
    visited = empty list
    push n to S
    while S is not empty:
        pop n from S
        add n to visited
        for each node u adjacent to n:
            push u to S
```

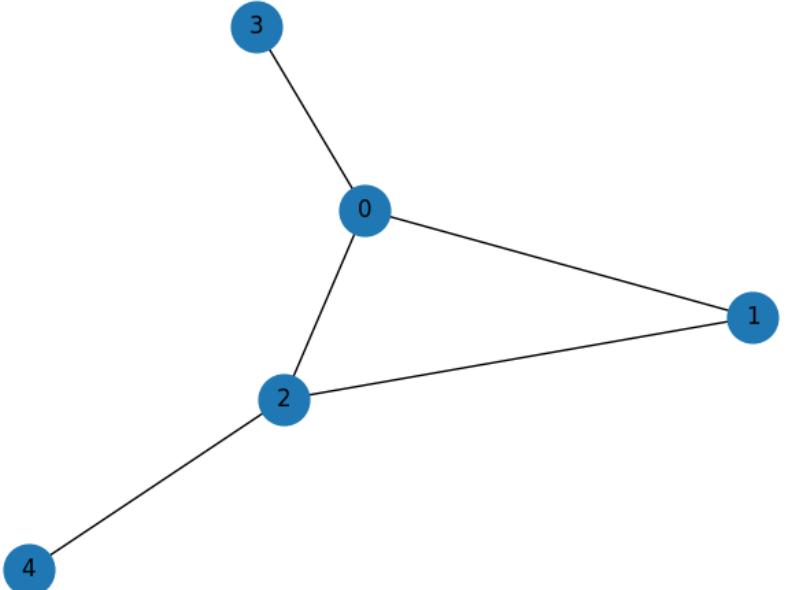


Figura 8: Esempio DFS

Depth First Search

Inizializzazione

```
stack = [ 0 ]  
visited = [ ]
```

Iterazione #1

```
stack = [ 1, 2, 3 ]  
visited = [ 0 ]
```

Iterazione #2

```
stack = [ 1, 2 ]  
visited = [ 0, 3 ]
```

Iterazione #3

```
stack = [ 1, 4 ]  
visited = [ 0, 3, 2 ]
```

Iterazione #4

```
stack = [ 1 ]  
visited = [ 0, 3, 2, 4 ]
```

Iterazione #5

```
stack = [ ]  
visited = [ 0, 3, 2, 4, 1 ]
```

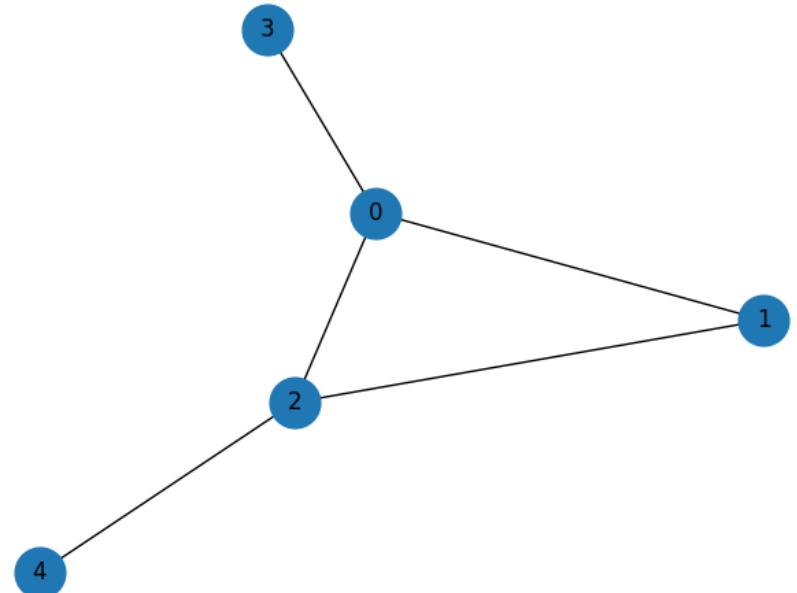


Figura 8: Esempio DFS

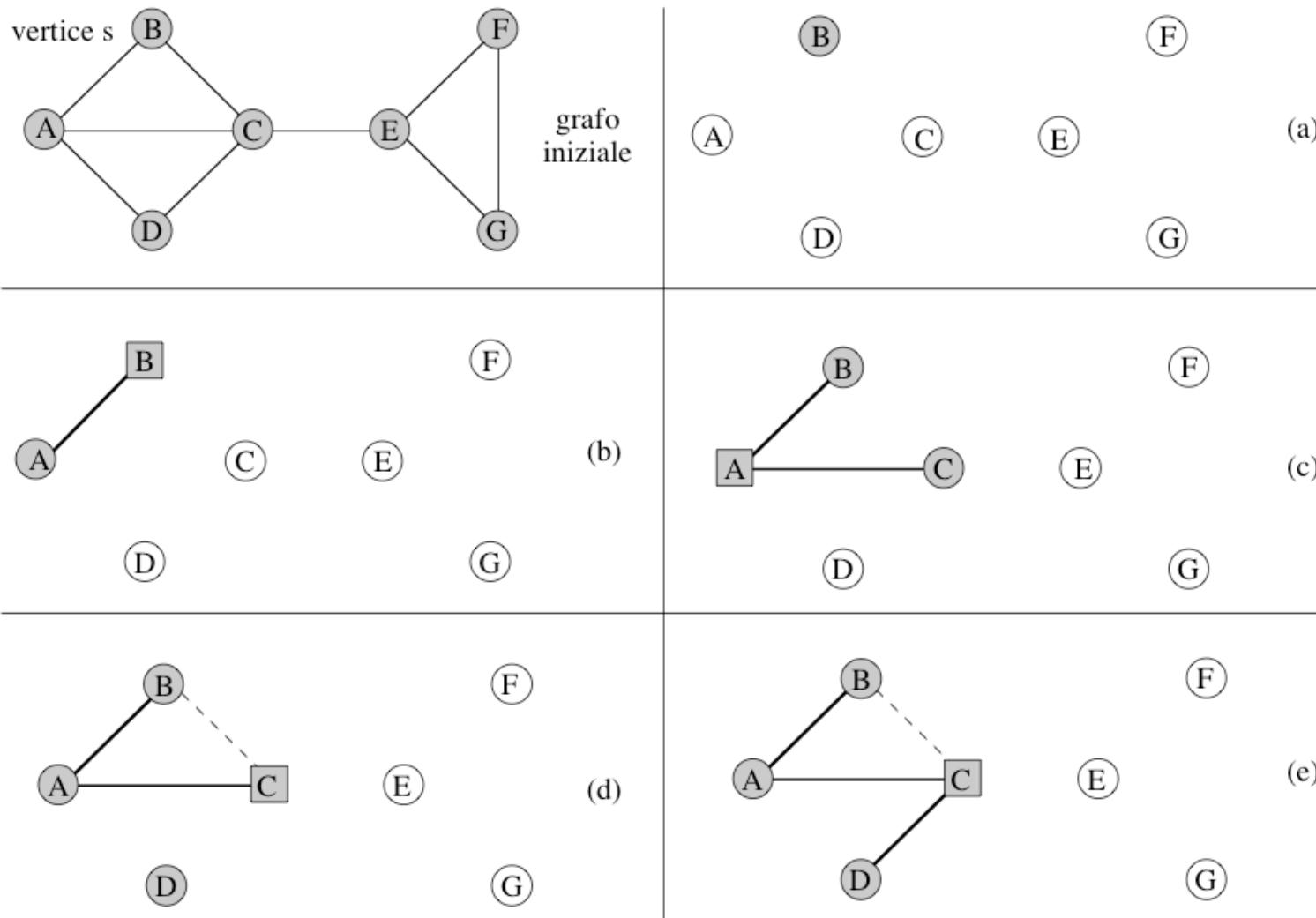
Depth First Search

Versione ricorsiva (con albero di copertura):

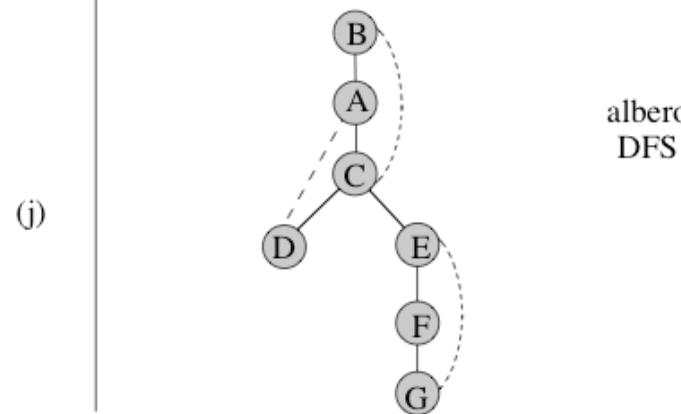
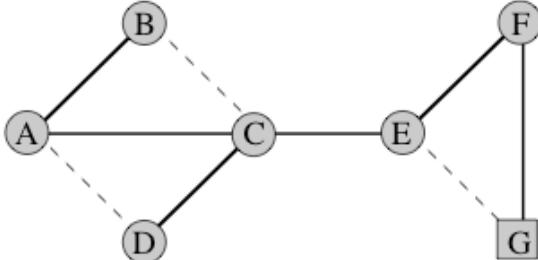
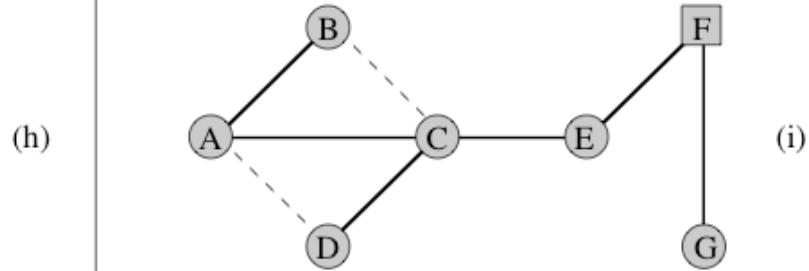
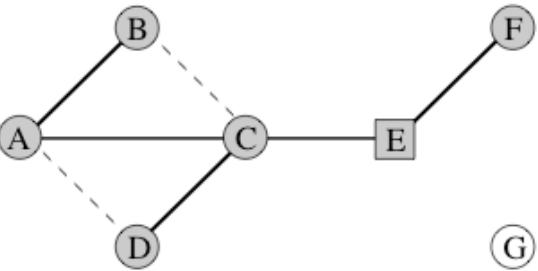
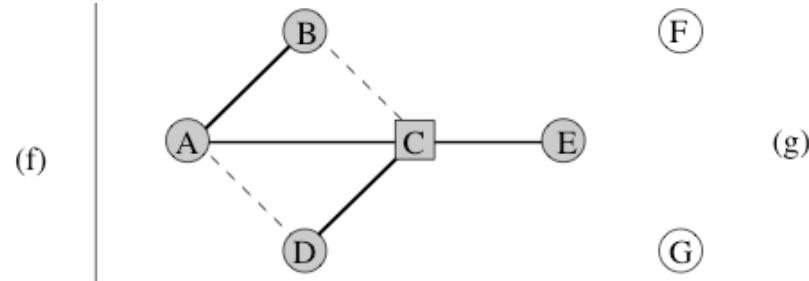
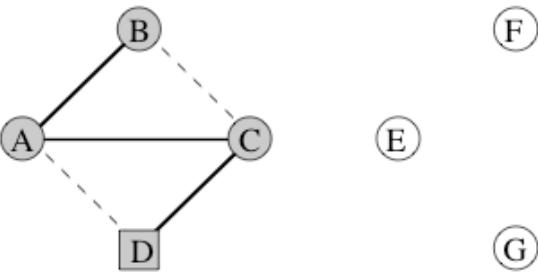
```
function recursive_DFS(node n, tree T, graph G):
    mark n as visited;
    for each edge (n, v) in G:
        if v is not visited:
            add (n, v) in T
            recursive_DFS(node v, tree T, grafo G)

driver DFS(node n, grafo G):
    T = empty tree
    recursive_DFS(node n, tree T, grafo G)
    return T
```

Depth First Search

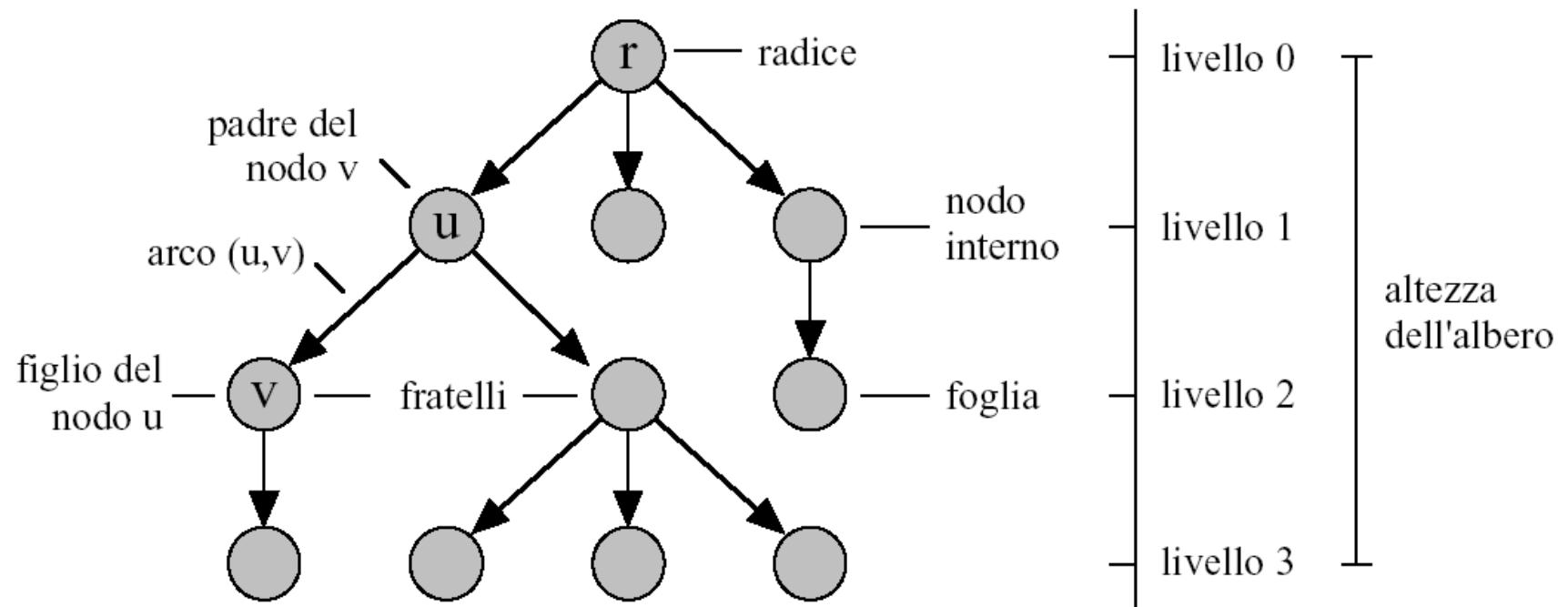


Depth First Search



Alessio Martino / Preparazione Olimpiadi Informatica

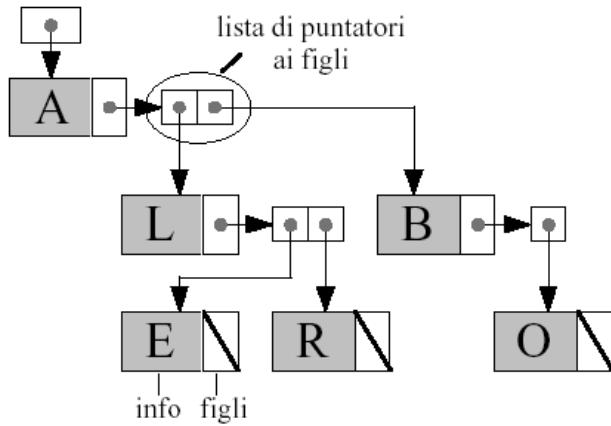
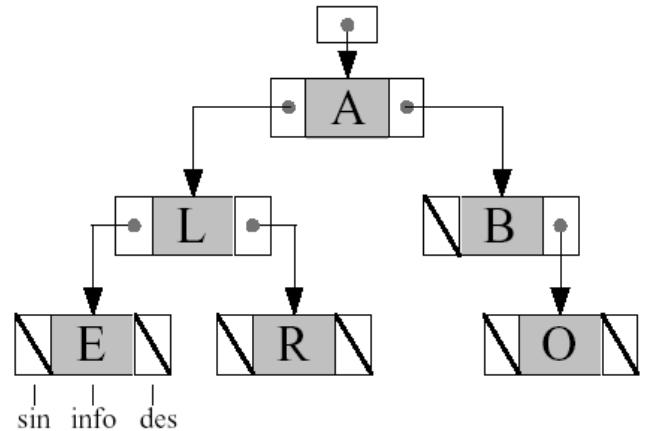
Alberi radicati: grafi aciclici con radice



- organizzazione gerarchica dei dati
 - dati contenuti nei nodi
 - relazioni gerarchiche definite dagli archi che li collegano

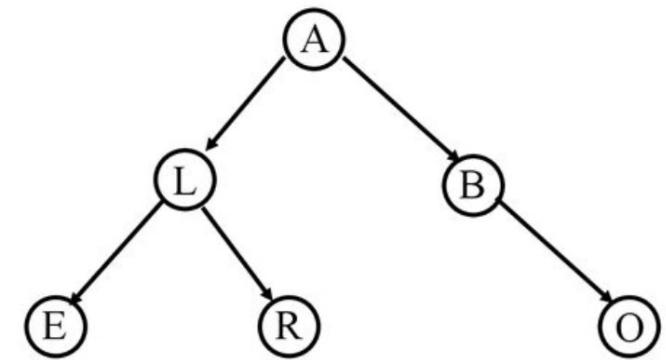
Rappresentazioni di alberi

Rappresentazione con puntatori ai figli (nodi con numero limitato di figli)



Rappresentazione con liste di puntatori ai figli (nodi con numero arbitrario di figli)

Vettore dei padri



(L,3)	(B,3)	(A,null)	(O,2)	(E,1)	(R,1)
1	2	3	4	5	6

Bae: Come over

Dijkstra: But there are so many routes to take and
I don't know which one's the fastest

Bae: My parents aren't home

Dijkstra:

Dijkstra's algorithm

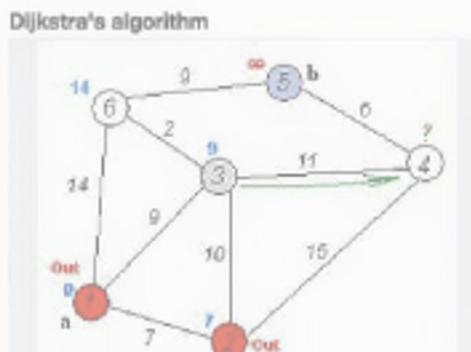
Graph search algorithm

Not to be confused with Dykstra's projection algorithm.

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later.^{[1][2]}

The algorithm exists in many variants; Dijkstra's original variant found the shortest path between two nodes,^[2] but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree.

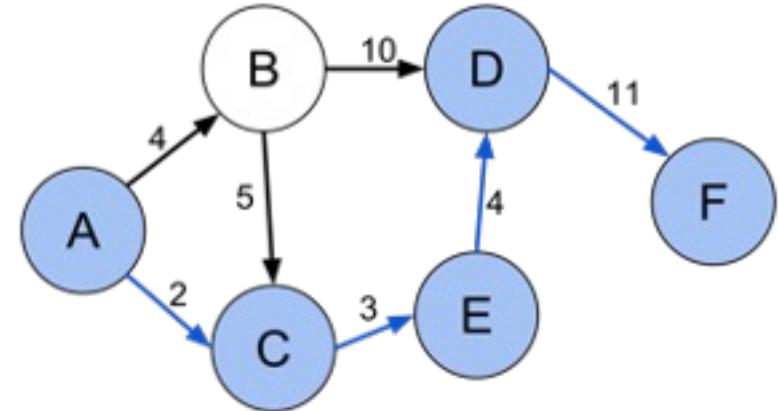
Algoritmo di Dijkstra



Cammini e costi

Sia $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ un grafo orientato

- \mathcal{V} è l'insieme di vertici
- \mathcal{E} è l'insieme degli archi
- ogni arco è formato da una tupla (s, t, w) con
 - s nodo sorgente
 - t nodo destinazione
 - w è un costo (scalare non negativo)
- Il **costo di un cammino** è la somma dei costi degli archi coinvolti nel cammino
- Il **cammino minimo** tra due nodi x ed y è il cammino di costo minore o uguale a quello di ogni altro cammino tra x ed y



Cammini e costi

- La **distanza** $d_{x,y}$ tra due vertici x ed y equivale a:
 - il costo del cammino minimo tra x ed y
 - oppure ∞ se i due nodi non sono connessi
- In matematica, la **diseguaglianza triangolare** ci dice che

$$d_{x,z} \leq d_{x,y} + d_{y,z}$$

- Da cui la **Condizione di Bellman**: per ogni arco (u, v) ed un vertice s

$$d_{s,u} + c(u, v) \geq d_{s,v}$$

dove con $c(u, v)$ indichiamo il costo dell'arco tra u e v

$$d(A, B) \leq d(A, C) + d(C, B)$$

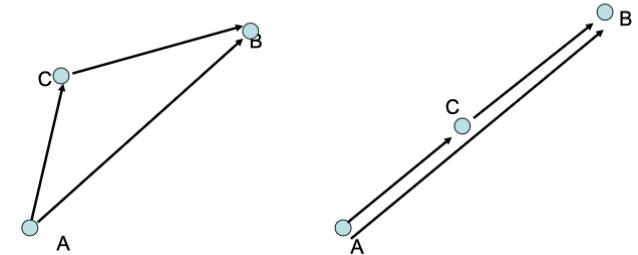


Figura 9: Diseguaglianza Triangolare

Albero dei cammini minimi

Un arco (u, v) appartiene ad un cammino minimo a partire da un vertice s se:

1. u è raggiungibile da s
2. $d_{s,u} + c(u, v) = d_{s,v}$

I cammini minimi da un vertice s a tutti gli altri vertici del grafo possono essere rappresentati tramite un albero radicato in s , detto **albero dei cammini minimi**.

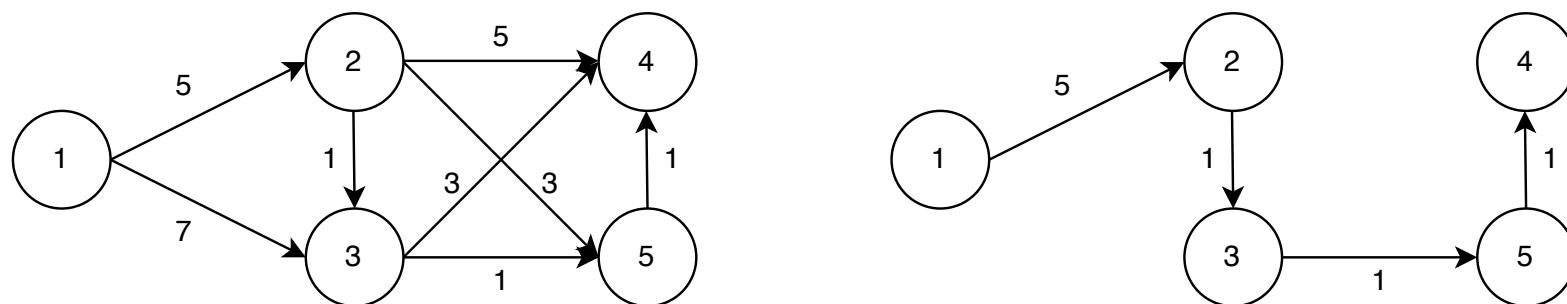


Figura 10: Albero dei cammini minimi radicato in 1

Tecnica del Rilassamento

Partendo da **stime** per eccesso delle distanze $d_{x,y}$ si aggiornano le stime, decrementandole progressivamente fino a renderle **esatte**.

Aggiornamento delle distanze sulla base della seguente regola di rilassamento:

```
if  $d_{x,v} + c(v, y) < d_{x,y}$ 
then  $d_{x,y} = d_{x,v} + c(v, y)$ 
```

Questa regola di rilassamento è alla base dell'approccio di Dijkstra:

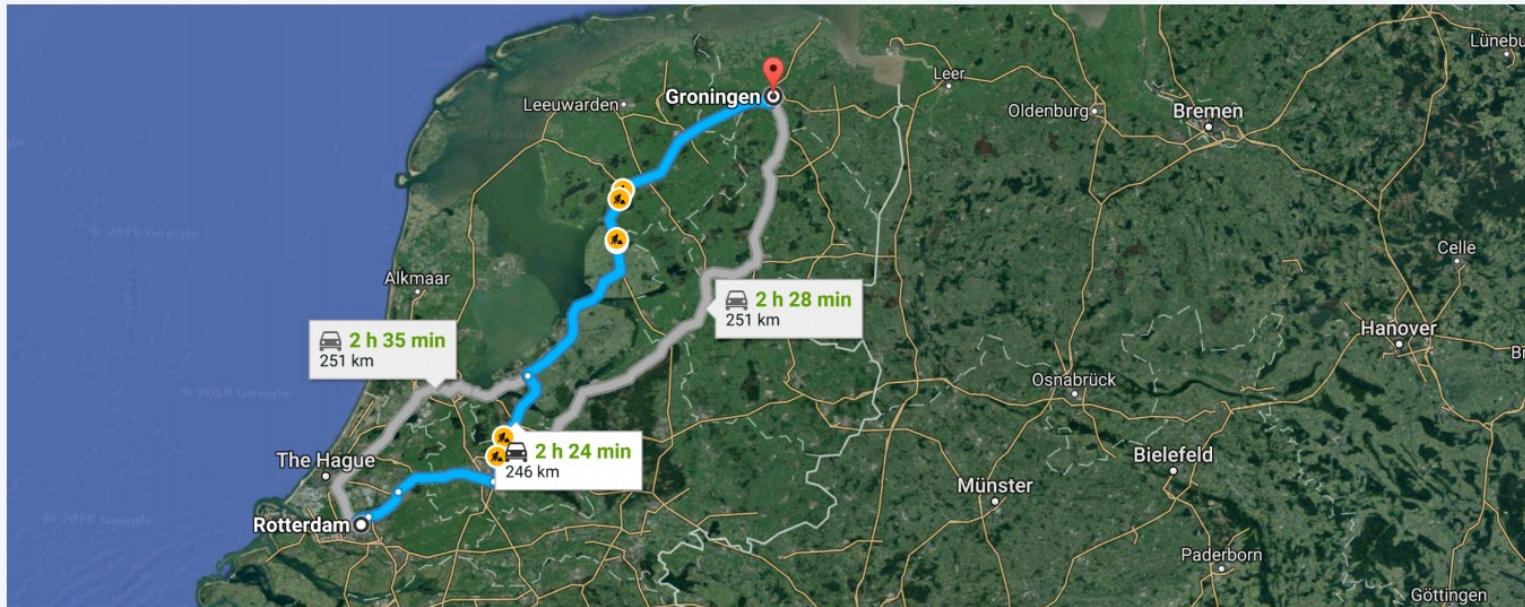
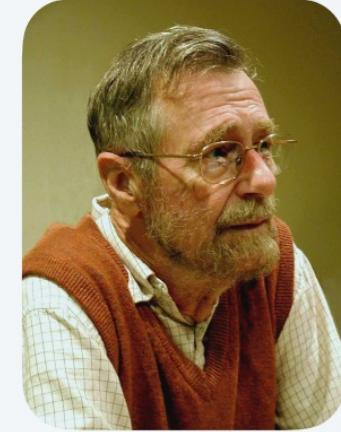
scegli un arco (u, v) con $u \in T$ e $v \notin T$ tale da minimizzare la quantità

$$d_{s,u} + c(u, v)$$

e poi effettua il passo di rilassamento, aggiungendolo a T

$$d_{s,v} = d_{s,u} + c(u, v)$$

“What’s the shortest way to travel from Rotterdam to Groningen? It is the algorithm for the shortest path, which I designed in about 20 minutes. One morning I was shopping in Amsterdam with my young fiancée, and tired, we sat down on the café terrace to drink a cup of coffee and I was just thinking about whether I could do this, and I then designed the algorithm for the shortest path.” — Edsger Dijkstra



Algoritmo di Dijkstra (v1.0)

```
function dijkstra(node  $s$ , graph  $G$ ):
```

```
     $Q$  = empty list (or set)
```

```
    for each vertex  $v$  in  $G$ :
```

```
         $dist[v] = \text{INFTY}$ 
```

```
         $prev[v] = \text{NULL}$ 
```

```
        add  $v$  to  $Q$ 
```

```
     $dist[s] = 0$ 
```

$dist[v]$ contiene la
distanza tra s e v

$prev[v]$ contiene il
nodo precedente a v

```
    while  $Q$  is not empty:
```

```
         $u$  = node from  $Q$  with minimum  $dist[u]$ 
```

```
        delete  $u$  from  $Q$ 
```

calcolo la distanza tra s ed
 v qualora passassi per u

```
        for each  $(u, v)$  with  $v$  in  $Q$ :
```

```
            temp =  $dist[u] + c(u, v)$ 
```

```
            if temp <  $dist[v]$ :
```

```
                 $dist[v] = temp$ 
```

```
                 $prev[v] = u$ 
```

ho trovato un path più
breve?

Algoritmo di Dijkstra (v1.0)

```
function dijkstra(node s, graph G):
    Q = empty list (or set)
    for each vertex v in G:
        dist[v] = INFTY
        prev[v] = NULL
        add v to Q
    dist[s] = 0

    while Q is not empty:
        u = node from Q with minimum dist[u]
        delete u from Q

        for each (u, v) with v in Q:
            temp = dist[u] + c(u, v)
            if temp < dist[v]:
                dist[v] = temp
                prev[v] = u
```

$$\mathcal{O}(n \cdot n + m) = \mathcal{O}(n^2)$$

n = numero di nodi
 m = numero di archi

itero su ciascun nodo

ricerca lineare per il minimo tra i nodi

itero su ciascun arco

Algoritmo di Dijkstra (v2.0)

```
function dijkstra(node s, graph G):
    Q = empty min-priority queue
    dist[s] = 0
    Q.add(s, dist[s])
    for each vertex v!=s in G:
        dist[v] = INFTY
        prev[v] = NULL
        Q.add(v, dist[v])

    while Q is not empty:
        u = Q.take_minimum()
        for each (u, v) in G:
            temp = dist[u] + c(u, v)
            if temp < dist[v]:
                dist[v] = temp
                prev[v] = u
                Q.change_priority(v, temp)
```

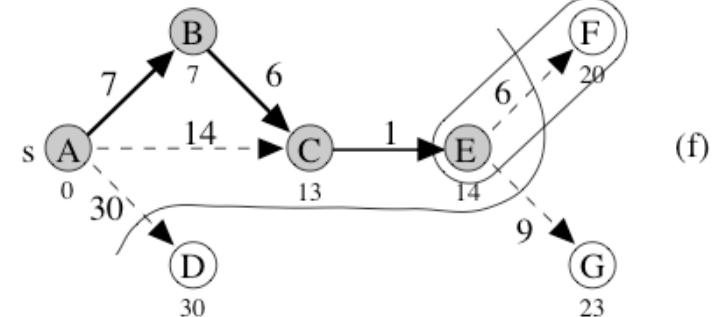
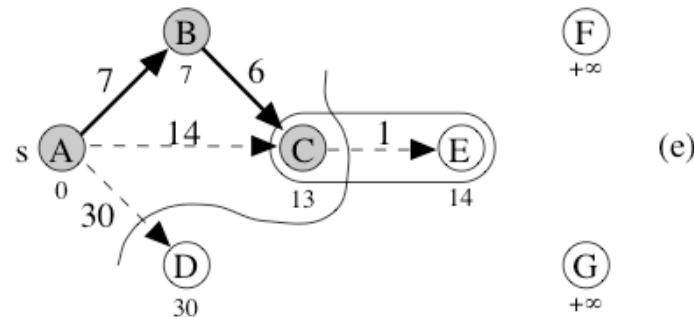
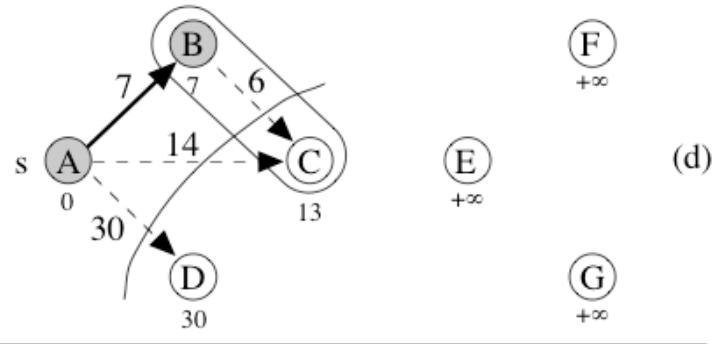
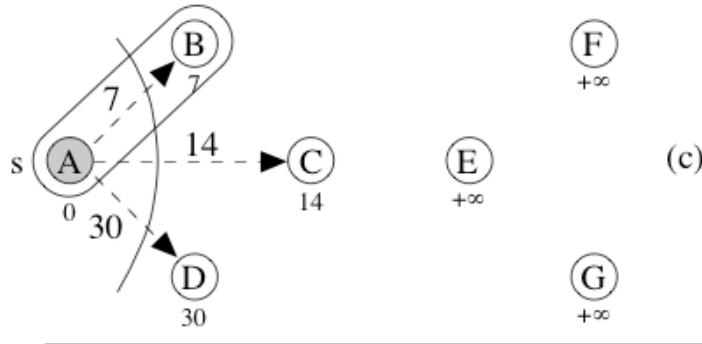
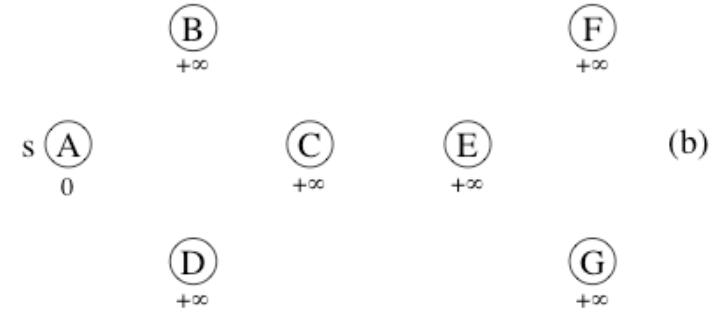
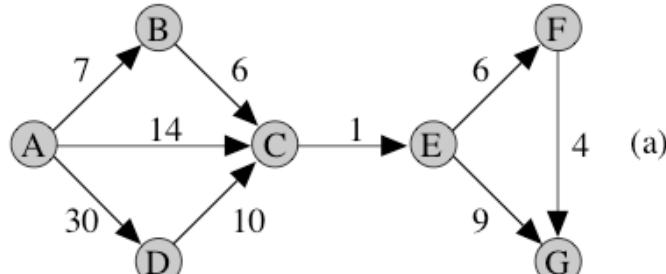
$\mathcal{O}((\log_2 n) \cdot n + m)$
 n = numero di nodi
 m = numero di archi

itero su ciascun nodo

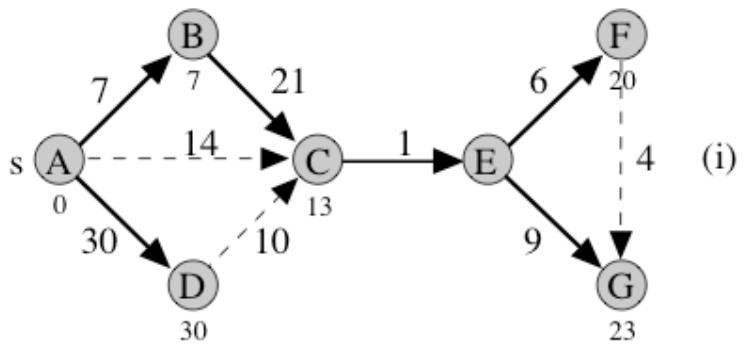
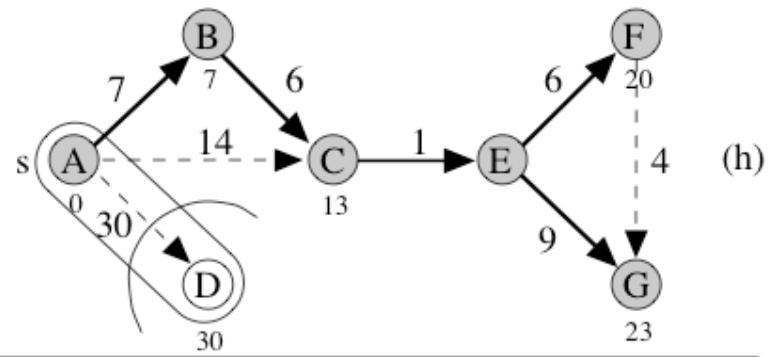
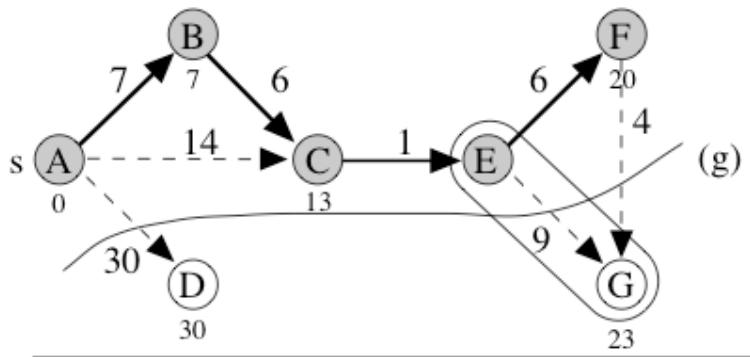
ricerca logaritmica per il minimo tra i nodi

itero su ciascun arco

Esempio



Esempio



Problema

Police Investigation 5

https://training.olinfo.it/#/task/ois_police5/statement

The police is still searching fearsome William! Given the skills of the criminal, who is apparently able to drive his car and find his way through the M roads of the city, the police has placed bombs in some of those roads to stop and eventually catch him. A button pressed at time T will trigger the simultaneous explosion of all the bombs, effectively making some roads unusable.



Figure 1: William driving: apparently is very calm in a deserted city.

Labelling the N intersections with integers, William is now at the intersection 0 and wants to reach his nest at the intersection $N - 1$. Thanks to his network of fellow criminals, he has come into possession of the plan of the police, knowing for each road from intersection A_i to B_i whether it will explode or not.

William knows how many seconds it will cost him to drive through each road, but obviously cannot drive on roads that are already exploded or will explode while driving on them. Is he going to be able to reach his nest and get away with it? If so, how many seconds is it going to take?

Input

The first line contains three integers: N , M and T . Each of the following M lines describes a road with four integers: the start A_i , the destination B_i , the cost C_i in seconds to drive through it, and whether it will explode ($E_i = 1$) or not ($E_i = 0$).

Output

You need to write a single line with an integer: the time in seconds required for William to reach his nest, or -1 if he cannot make it.

Esercizi per casa

1. Sunnydale

<https://training.olinfo.it/#/task/sunny/statement>

2. Connect the Dots

<https://training.olinfo.it/#/task/nostar/statement>

3. Gondola network

https://training.olinfo.it/#/task/ois_vapoare/statement

4. Monete a posto

https://training.olinfo.it/#/task/gator_monete/statement

5. Topologia di rete

<https://training.olinfo.it/#/task/topologia/statement>