Luiss
Libera Università Internazionale
degli Studi Sociali Guido Carli

Corso di preparazione per la selezione territoriale delle Olimpiadi di Informatica

Lezione 6. Grafi

Giuseppe F. Italiano

29 aprile 2020

LUISS

Programma di oggi

1. Correzione compiti a casa
2. Riassunto Programmazione Dinamica
3. Grafi
4. Un problema dale Territoriali (2015): Rispetta i versi (Rivisitato!)
5. Un problema dalle Territoriali (2004/2005): Sunnydale
6. Un problema dalle Gator (2016): Monete a posto
7. Esercizi Olimpiadi (compiti da fare a casa)

Correzione compiti a casa

Compiti a casa

1. Sushi variegato (pre OII 2017) https://training.olinfo.it/#/task/preoii_yutaka/statement
2. Esame di maturità (ABC 2016) https://training.olinfo.it/#/task/abc_esame/statement
3. Numeri di Figonacci (OIS 2014) https://training.olinfo.it/#/task/figonacci/statement
4. Petali di margherita (Nazionali 2007) https://training.olinfo.it/#/task/petali/statement
5. Board game (OIS 2020) https://training.olinfo.it/#/task/ois_marcel/statement
6. Canottaggio (Gator 2015) https://training.olinfo.it/#/task/canoa/statement
7. Missioni (Territoriali 2008) https://training.olinfo.it/#/task/missioni/statement
8. Fibonacci colonies (OIS 2020) https://training.olinfo.it/#/task/ois_fibonaccibug/statement

Riassunto: Il problema dello zaino (knapsack)

Molte varianti...
(versione non frazionaria)

(alcune slide rubate alla Prof. Irene Finocchi)

Le 5 Leggi della Programmazione Dinamica

1. Definire opportunamente sottoproblemi (tabella)
2. Condizioni iniziali della tabella
3. Dove sta la soluzione?
4. Riempire la tabella (da condizioni iniziali a soluzione)
5. Verificare cosa ci serve effettivamente della tabella (ridurne le dimensioni?)

Il problema dello zaino (knapsack)

- Un ladro ha uno zaino che può contenere al più W Kg di peso
- Può rubare n oggetti (senza superare il limite di peso)
- Oggetto i pesa w_i Kg e ha un valore di v_i Euro
- Cosa gli conviene rubare? Massimizzare valore complessivo della refurtiva senza superare il massimo peso W sopportabile dallo zaino
- Gli oggetti vanno presi per intero: non è possibile inserire nello zaino *porzioni di oggetti!* (versione non frazionaria)

Greedy funziona?

i	v_i	w_i
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

W = 11

- Scelta per valore (decrescente): {5,2,1} di valore 35
- Scelta per peso (crescente): {1,2,3} di valore 25
- Scelta per valore/peso (decrescente): {5,2,1} di valore 35

  Rapporti valore/peso: 1, 3, 3.6, 3.7, 4
- *Qual è la soluzione ottima?*

Programmazione dinamica

- Sottoproblemi definiti in termini di due parametri:
  - Insieme degli oggetti tra cui scegliere
  - Peso massimo ammissibile

- OPT(i, p) = valore della soluzione ottima scegliendo un sottoinsieme degli oggetti {**1, ..., i**}, per i ≤ **n,** e assumendo di avere un limite p ≤ **W** al massimo peso sopportabile dallo zaino

- La soluzione del nostro problema originario sarà OPT(n, W)

Esempio

i	v_i	w_i
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

W = 11

	0	1	2	3	4	5	6	7	8	9	10	11
{}												
{1}												
{1,2}												
{1,2,3}												
{1,2,3,4}												
{1,2,3,4,5}												

- Condizioni iniziali?
- Dov'è la soluzione?

Riempire la tabella: da (i-1) a i oggetti...

	0	1	2	3	4	5	6	7	8	9	10	11
{}	0	0	0	0	0	0	0	0	0	0	0	0
{1}	0											
{1,2}	0											
{1,2,3}	0											
{1,2,3,4}	0											
{1,2,3,4,5}	0											

Come passiamo da (i-1) ad i oggetti avendo a disposizione un certo peso p?

1) Se $w_i > p$, di sicuro l'oggetto i non può essere preso, quindi OPT(i,p) = OPT(i-1,p)

2) Se $w_i \leq p$, possiamo prendere o meno l'oggetto i:

- Se non lo prendiamo, OPT(i,p) = OPT(i-1, p)
- Se lo prendiamo, OPT(i,p) = v_i + OPT(i-1, p-w_i) (peso a disposizione diventa p-w_i)

Quale scelta è più conveniente? Prendiamo il massimo di queste due quantità!

Esempio

i	v_i	w_i
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

W = 11

	0	1	2	3	4	5	6	7	8	9	10	11
{}	0	0	0	0	0	0	0	0	0	0	0	0
{1}	0											
{1,2}	0											
{1,2,3}	0											
{1,2,3,4}	0											
{1,2,3,4,5}	0											

$$OPT(i,w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1,w) & \text{if } w_i > w \\ \max\{ OPT(i-1,w),\ \ v_i + OPT(i-1,w-w_i)\} & \text{otherwise} \end{cases}$$

Esempio

i	v_i	w_i
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

W = 11

	0	1	2	3	4	5	6	7	8	9	10	11
{ }	0	0	0	0	0	0	0	0	0	0	0	0
{ 1 }	0	1	1	1	1	1	1	1	1	1	1	1
{ 1, 2 }	0	1	6	7	7	7	7	7	7	7	7	7
{ 1, 2, 3 }	0	1	6	7	7	18	19	24	25	25	25	25
{ 1, 2, 3, 4 }	0	1	6	7	7	18	22	24	28	29	29	40
{ 1, 2, 3, 4, 5 }	0	1	6	7	7	18	22	28	29	34	35	40

Come si trovano gli oggetti scelti?

$$OPT(i,w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1,w) & \text{if } w_i > w \\ \max\{ OPT(i-1,w),\ \ v_i + OPT(i-1,w-w_i)\} & \text{otherwise} \end{cases}$$

Implementazione (bottom up iterativa)

KNAPSACK $(n, W, w_1, \ldots, w_n, v_1, \ldots, v_n)$

FOR $w = 0$ TO W

  $M[0, w] \leftarrow 0$.

FOR $i = 1$ TO n

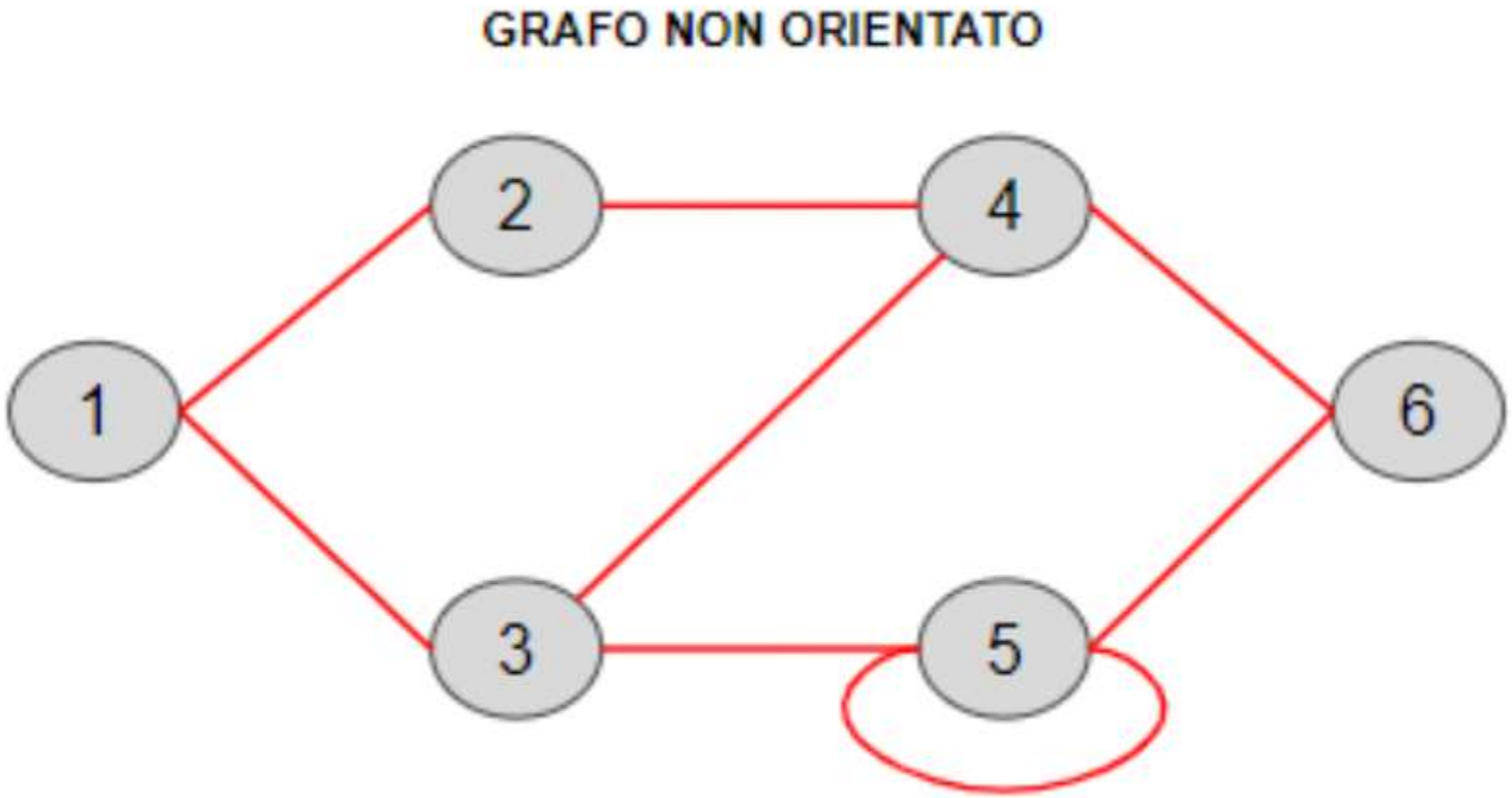
  FOR $w = 0$ TO W

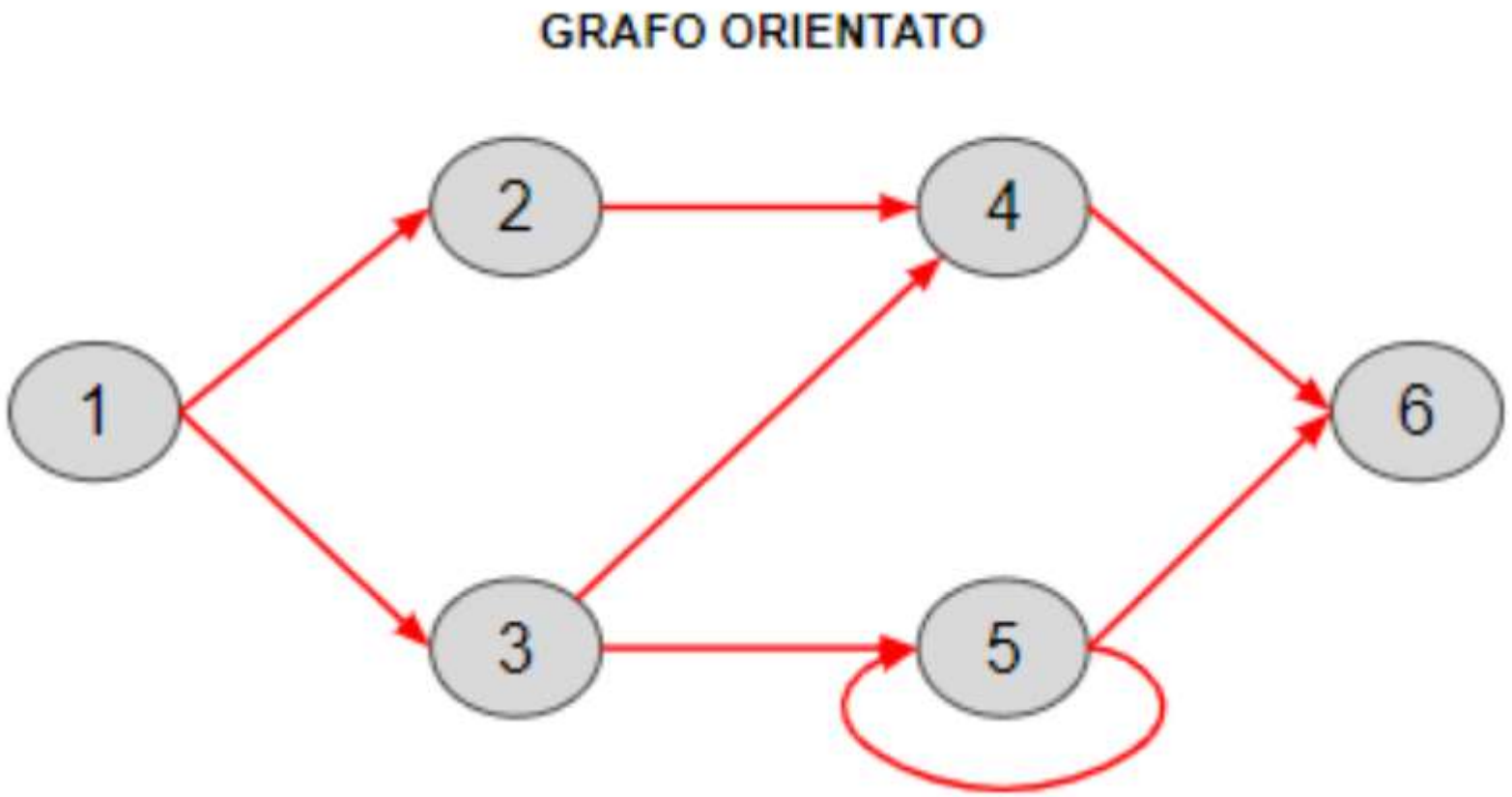
  IF $(w_i > w)$ $M[i, w] \leftarrow M[i-1, w]$.

  ELSE $M[i, w] \leftarrow \max \{ M[i-1, w],\ v_i + M[i-1, w - w_i] \}$.

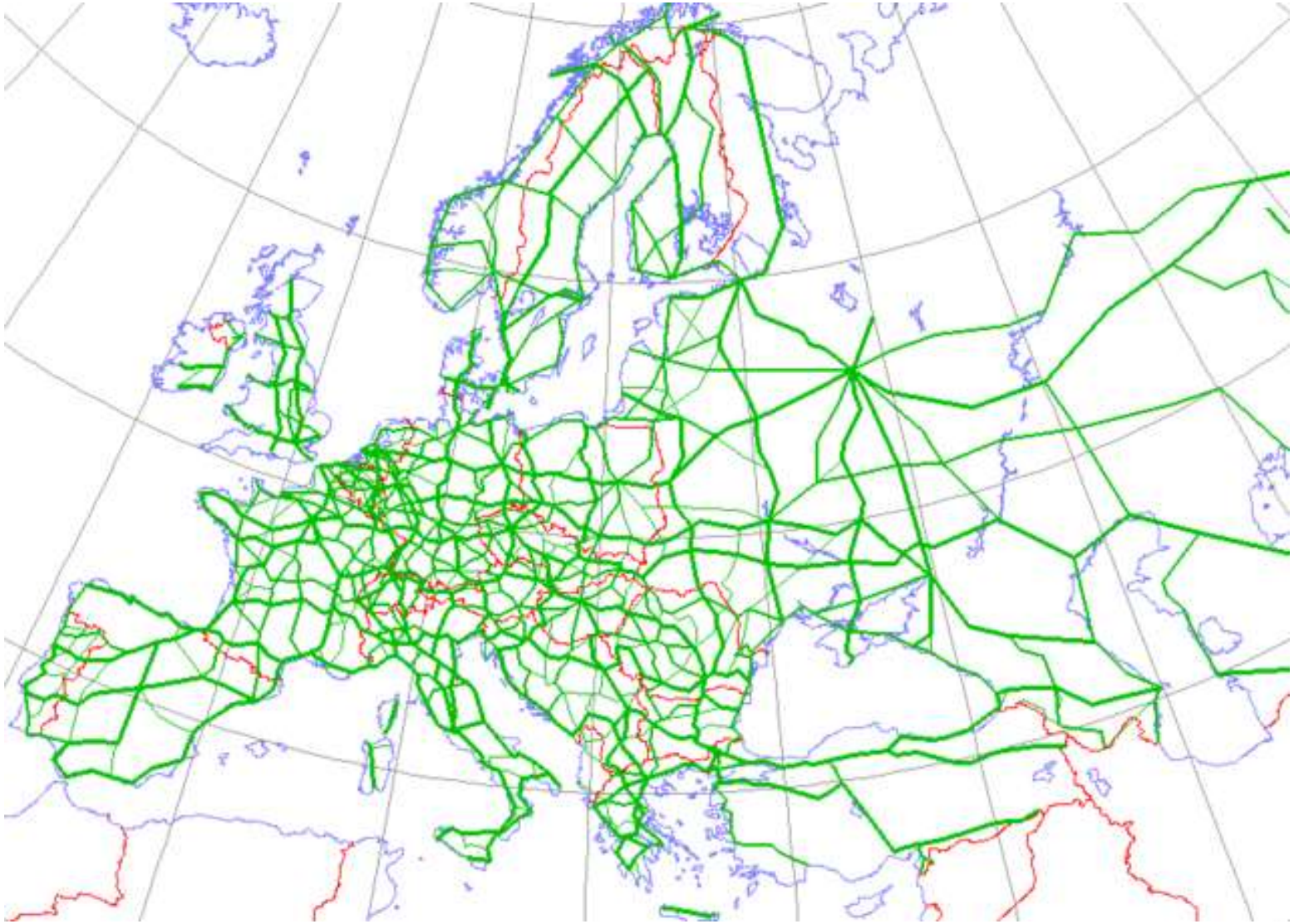
RETURN $M[n, W]$.

Grafi

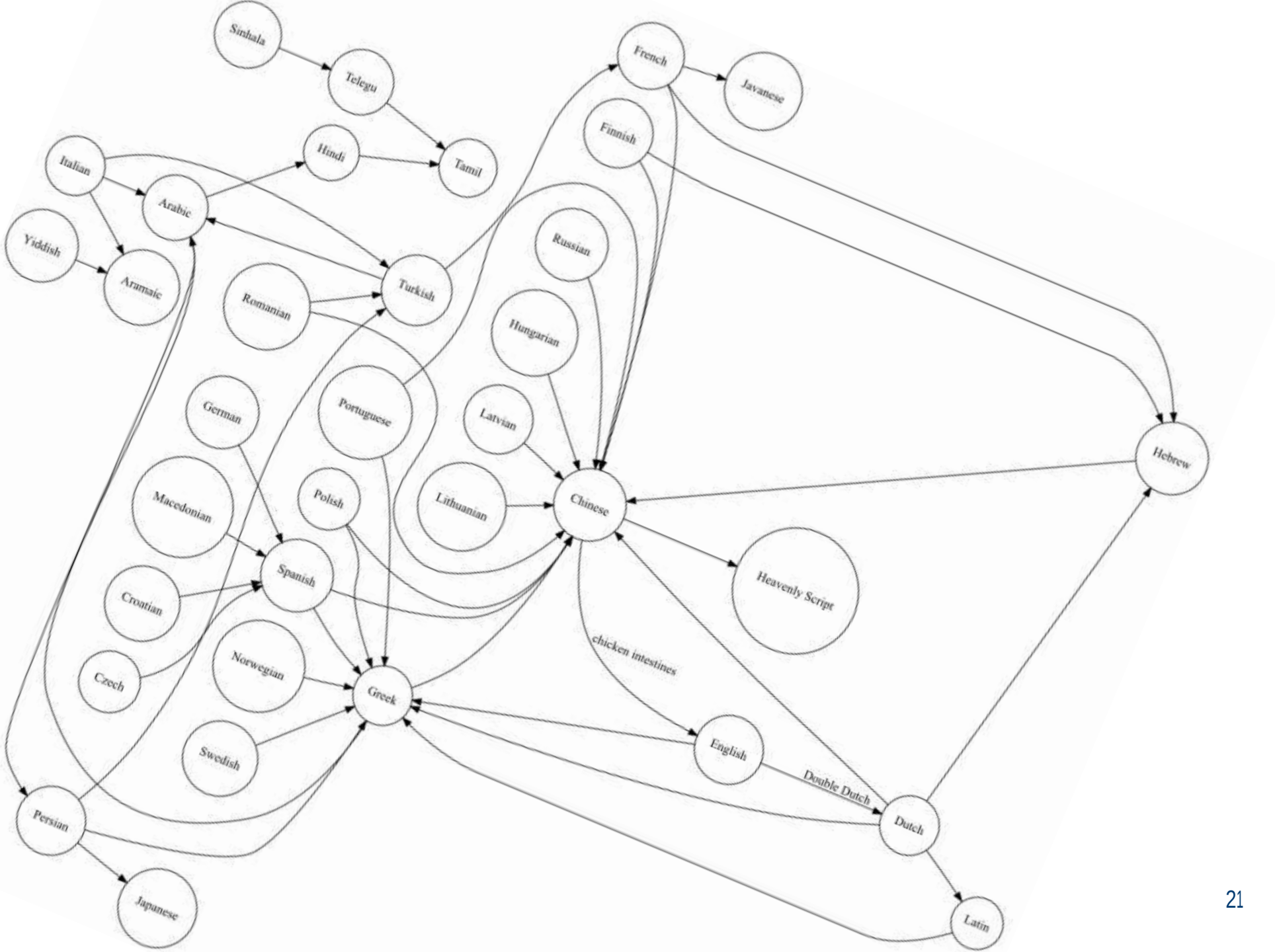

GRAFO NON ORIENTATO
1
2
3
4
5
6


GRAFO ORIENTATO
1
2
3
4
5
6

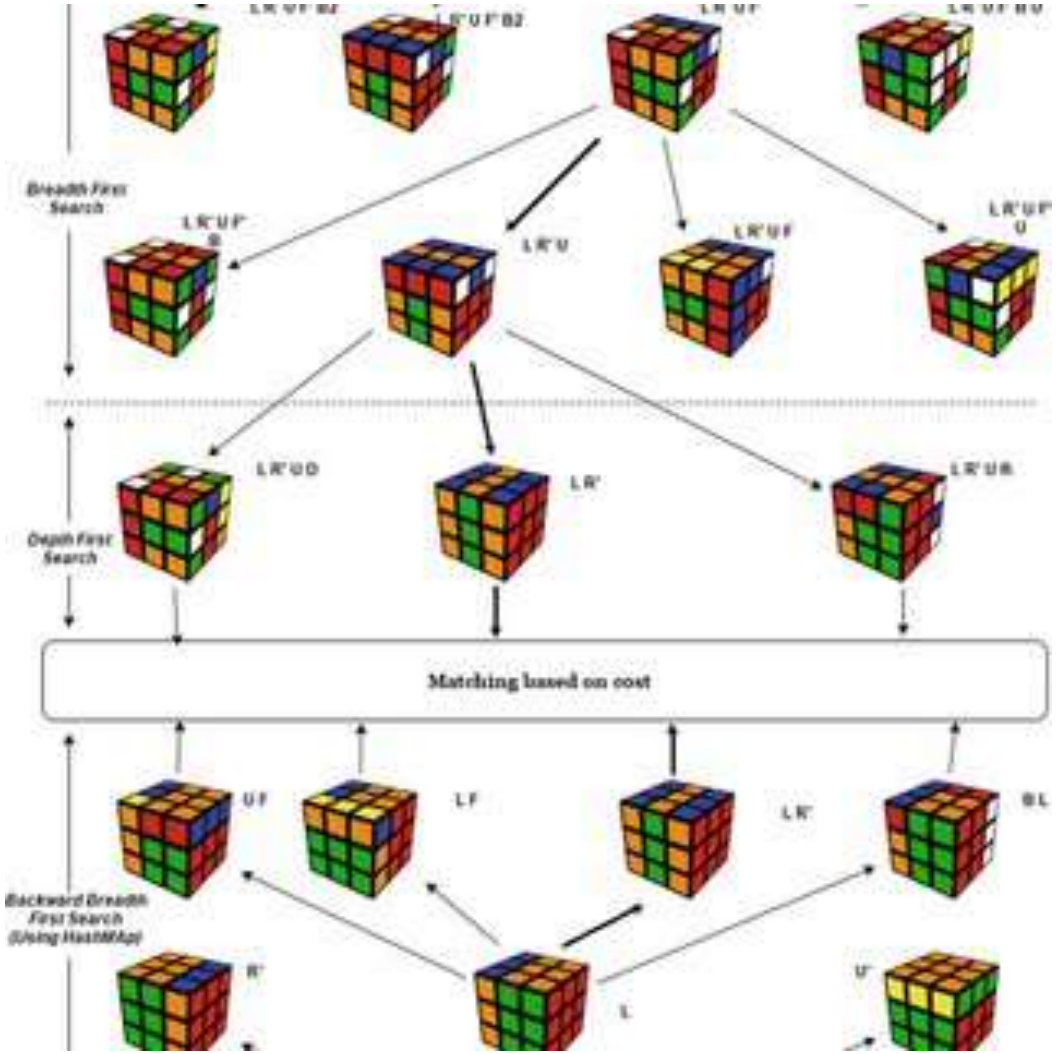
Abilene Federal/Research Network Peers

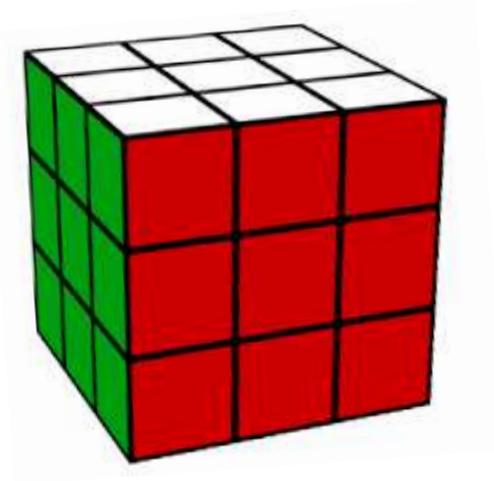



System Map
Legend
Commuter Rail
Parking
N


Sinhala
Telegu
Tamil
Hindi
Italian
Arabic
Yiddish
Aramaic
Turkish
Romanian
French
Javanese
Finnish
Russian
Hungarian
Portuguese
German
Latvian
Hebrew
Macedonian
Polish
Lithuanian
Chinese
Spanish
Heavenly Script
Croatian
chicken intestines
Norwegian
Czech
Greek
Swedish
English
Double Dutch
Persian
Dutch
Japanese
Latin


facebook
December 22 2010

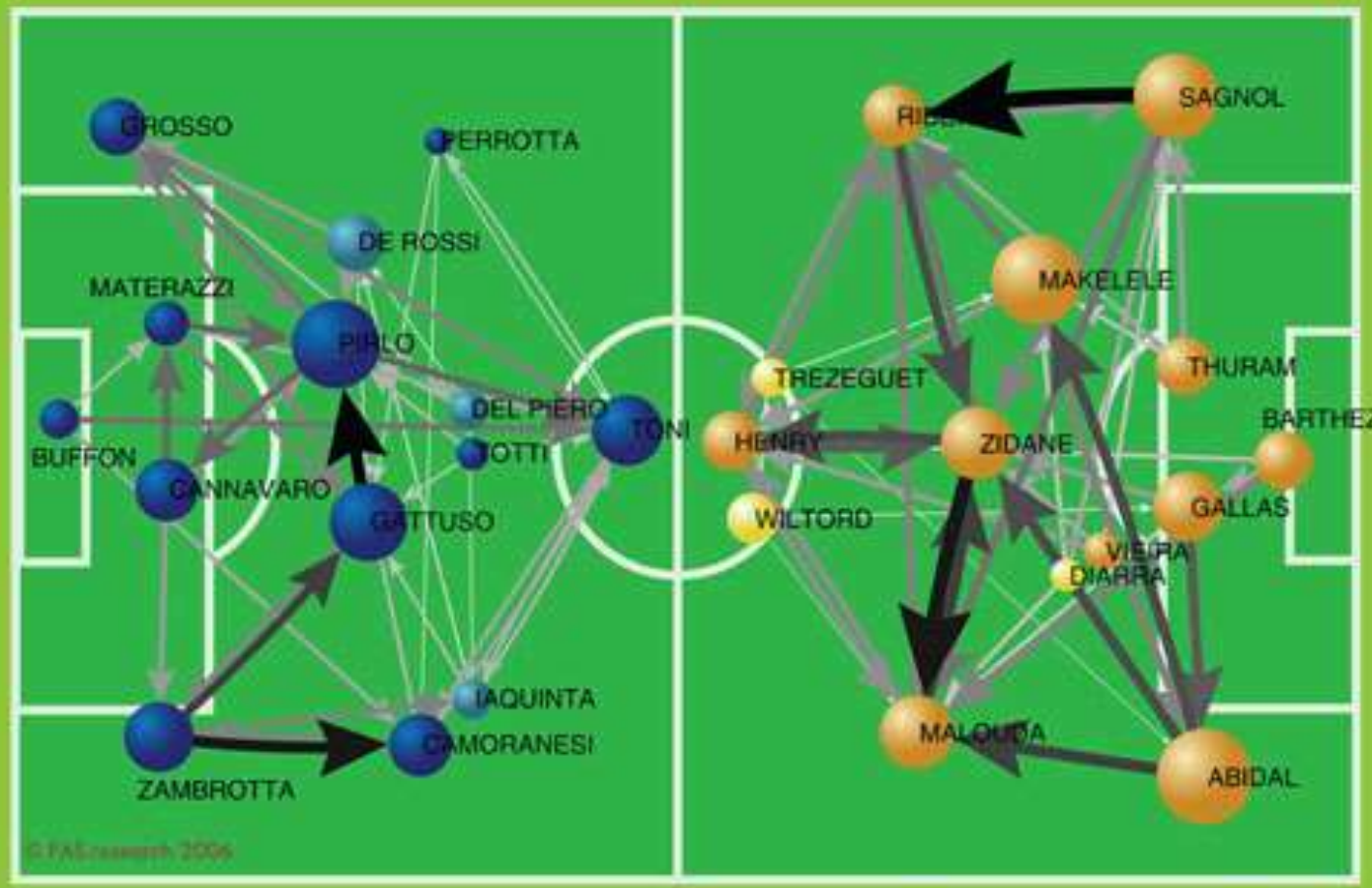

Breadth First Search
Depth First Search
Matching based on cost
Backward Breadth First Search (Using HashMap)



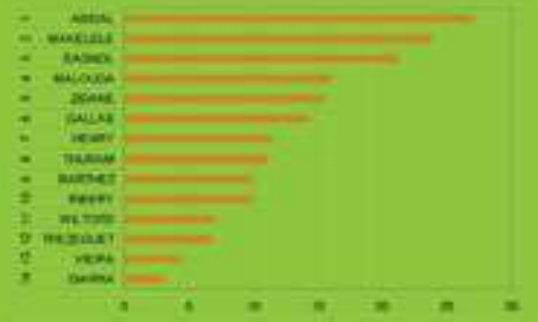
LUISS

WORLDCUP 2006 *Final*

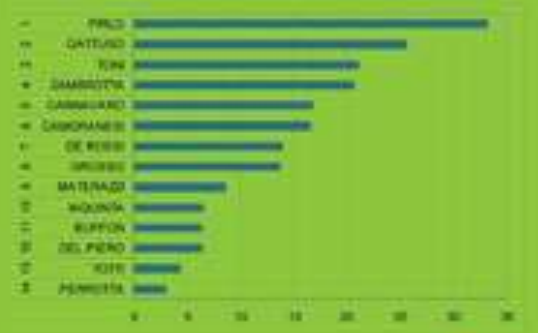
ITALY - FRANCE

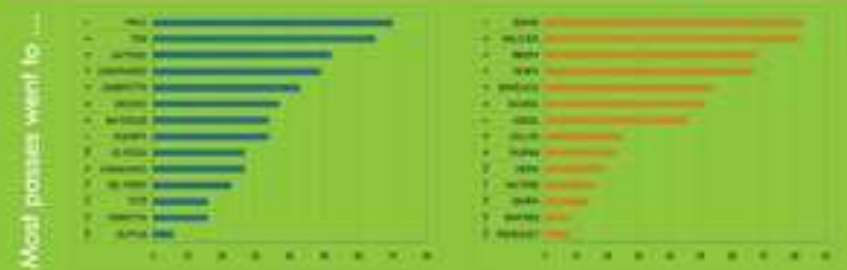


The most frequent (>15) passes between players



Keyplayer (hub in x % of activities)



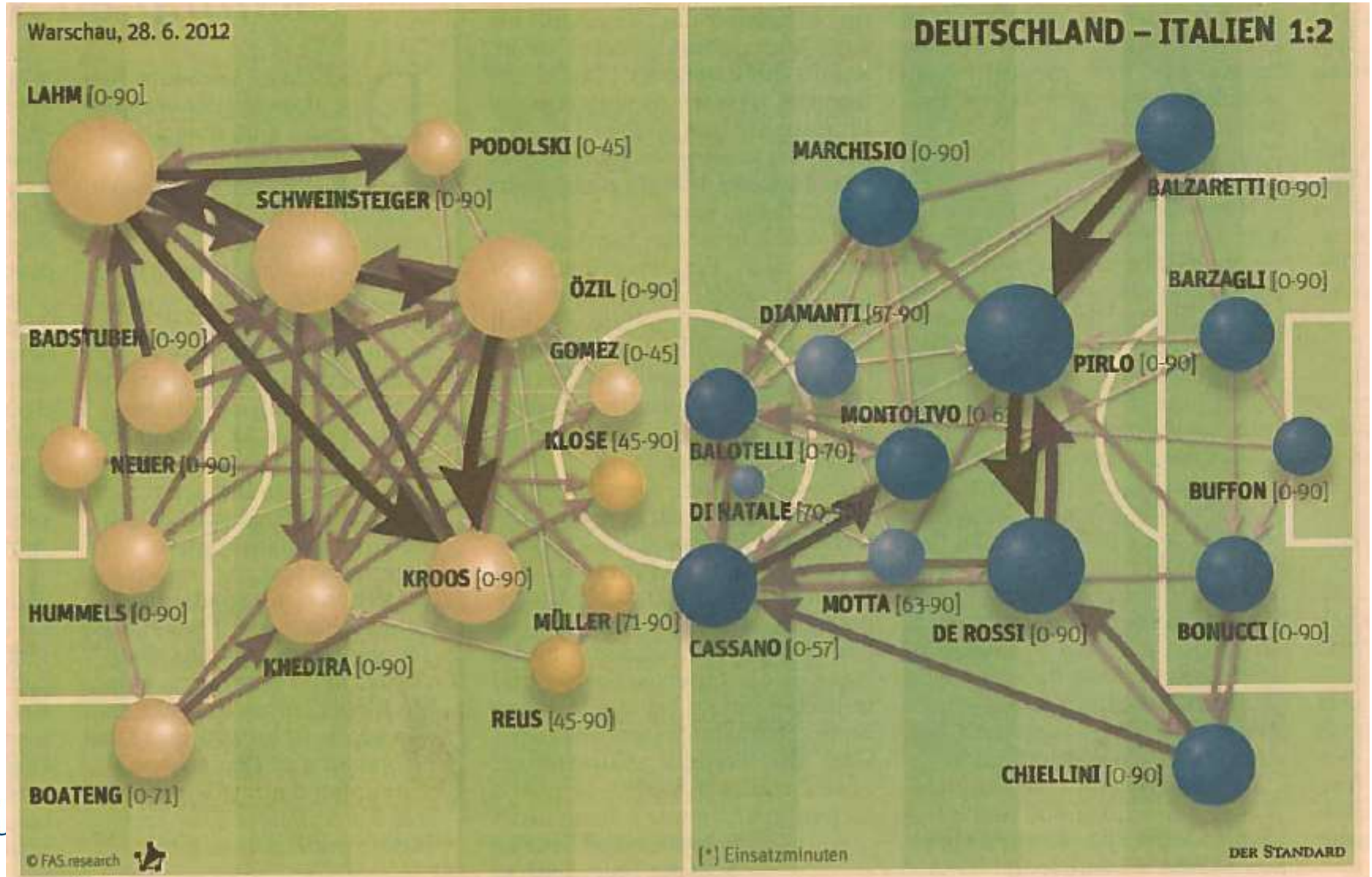


This network shows the passes from every player to those three team-mates, he passes to most frequently.
Strength of arcs displays the number of passes.
Size of nodes displays the influence (flowbetweenness) of a player.

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

FAS.research

Network Analysis for Science and Business


Warschau, 28. 6. 2012
DEUTSCHLAND – ITALIEN 1:2
LAHM [0-90]
PODOLSKI [0-45]
SCHWEINSTEIGER [0-90]
ÖZIL [0-90]
BADSTUBER [0-90]
GOMEZ [0-45]
KLOSE [45-90]
NEUER [0-90]
KROOS [0-90]
HUMMELS [0-90]
MÜLLER [71-90]
KHEDIRA [0-90]
REUS [45-90]
BOATENG [0-71]
© FAS.research
MARCHISIO [0-90]
BALZARETTI [0-90]
BARZAGLI [0-90]
DIAMANTI [57-90]
PIRLO [0-90]
MONTOLIVO [0-6
BALOTELLI [0-70]
BUFFON [0-90]
DI NATALE [70-
MOTTA [63-90]
DE ROSSI [0-90]
BONUCCI [0-90]
CASSANO [0-57]
CHIELLINI [0-90]
[*] Einsatzminuten
DER STANDARD

Giuseppe F. Italiano: Revision history

Help

View logs for this page (view filter log)

Filter revisions

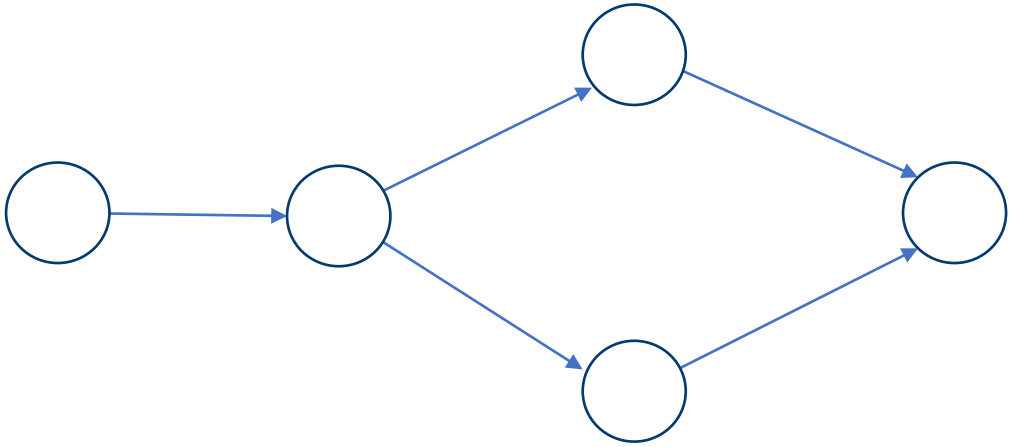
External tools: Find addition/removal (Alternate) · Find edits by user · Page statistics · Pageviews · Fix dead links

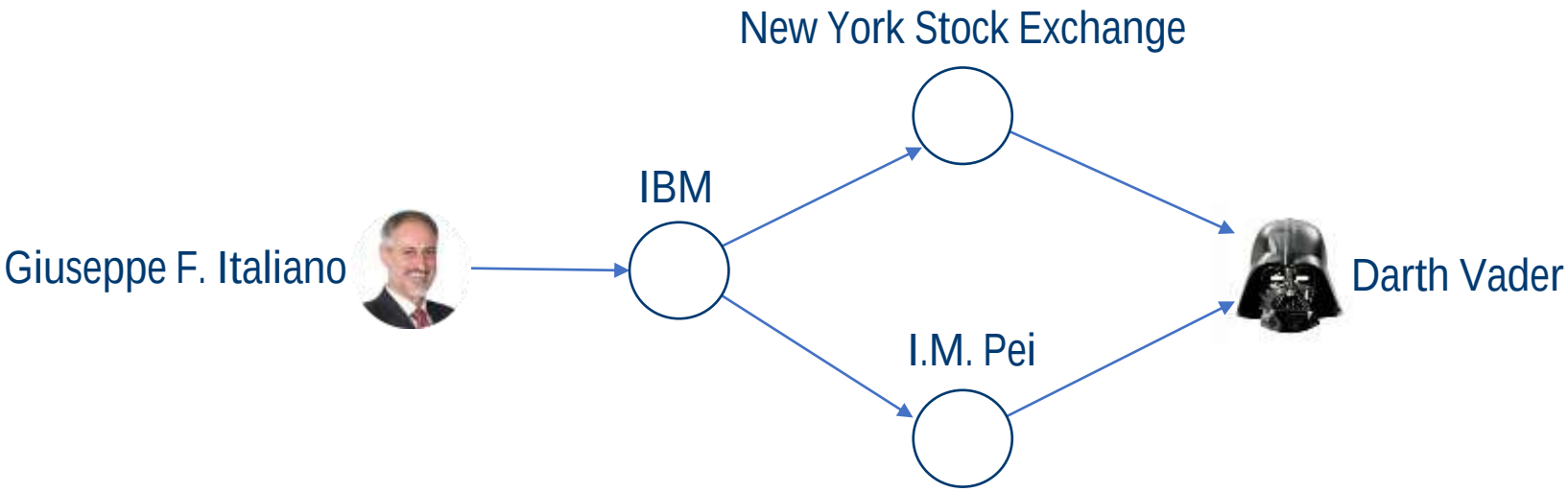
For any version listed below, click on its date to view it. For more help, see Help:Page history and Help:Edit summary. (cur) = difference from current version, (prev) = difference from preceding version, **m** = minor edit, → = section edit, ← = automatic edit summary

(newest | oldest) View (newer 50 | older 50) (20 | 50 | 100 | 250 | 500)

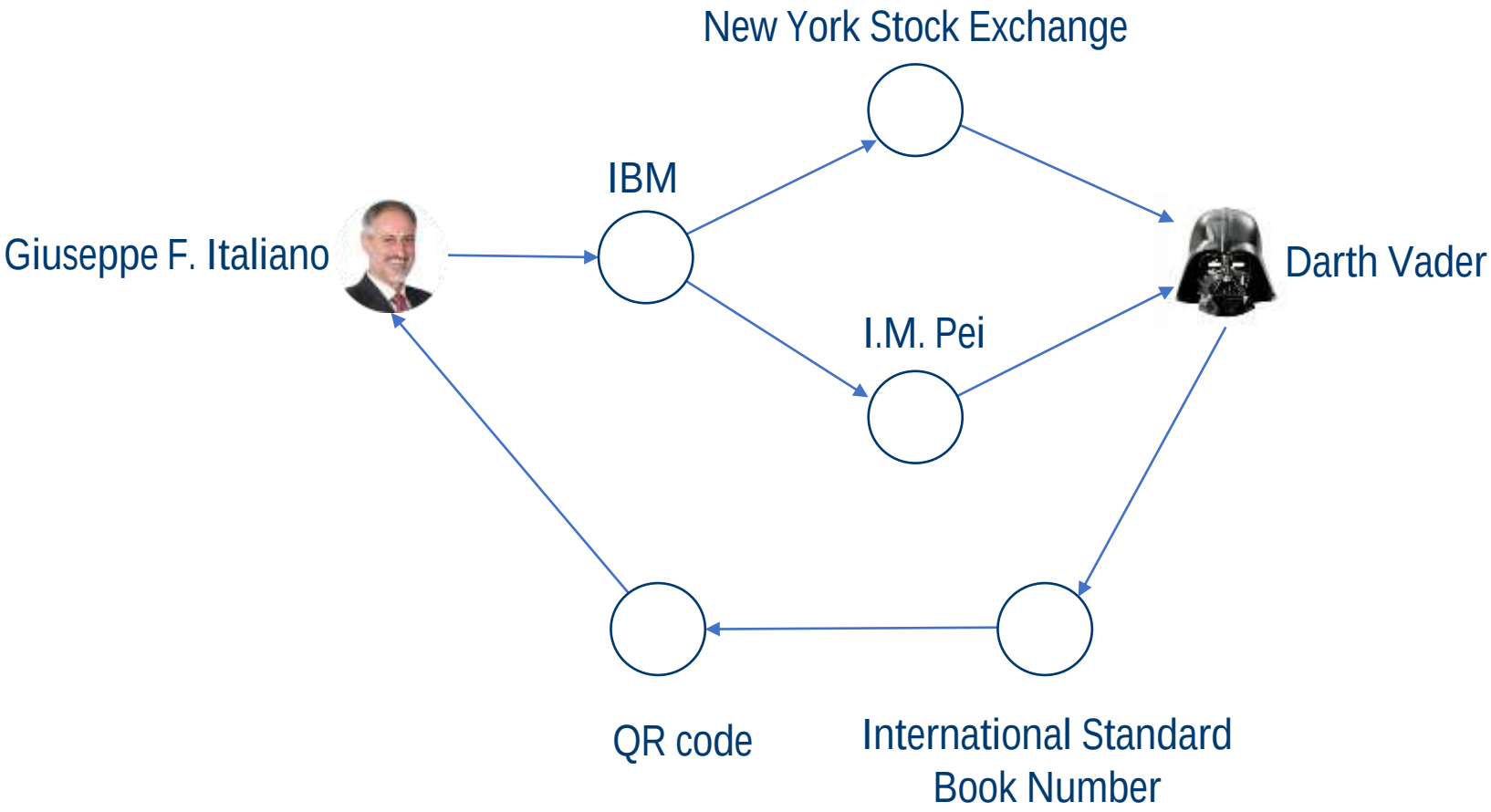
Compare selected revisions

- (cur | prev) 21:10, 23 April 2021 MusikBot II (talk | contribs) **m** . . (5,766 bytes) (+17) . . *(Adding missing protection template (more info))*
- (cur | prev) 20:59, 23 April 2021 David Eppstein (talk | contribs) **m** . . (5,749 bytes) (0) . . *(Protected "Giuseppe F. Italiano": Violations of the biographies of living persons policy ([Edit=Require autoconfirmed or confirmed access] (expires 20:59, 7 May 2021 (UTC)) [Move=Require autoconfirmed or confirmed access] (expires 20:59, 7 May 2021 (UTC))))*
- (cur | prev) 20:59, 23 April 2021 David Eppstein (talk | contribs) **m** . . (5,749 bytes) (−14) . . *(Reverted edits by 213.243.197.65 (talk) to last version by David Eppstein)* *(Tag: Rollback)*
- (cur | prev) 19:34, 23 April 2021 213.243.197.65 (talk) . . (5,763 bytes) (+14) . . *(i published the truth)* *(Tags: Visual edit, Manual revert, Reverted)*
- (cur | prev) 16:37, 22 April 2021 David Eppstein (talk | contribs) **m** . . (5,749 bytes) (−14) . . *(Reverted edits by Vittorio Smorfia (talk) to last version by Citation bot)* *(Tags: Rollback, Reverted)*
- (cur | prev) 14:18, 22 April 2021 Vittorio Smorfia (talk | contribs) **m** . . (5,763 bytes) (+14) . . *(Italiano is into memes and he is a very expert memer. In KEK community he's known as a Memelord)* *(Tag: Reverted)*
- (cur | prev) 01:50, 30 August 2020 Citation bot (talk | contribs) . . (5,749 bytes) (+96) . . *(Add: s2cid. | You can use this bot yourself. Report bugs here. |*





Da: sixdegreesofwikipedia.com/



Da: sixdegreesofwikipedia.com/

Definizione

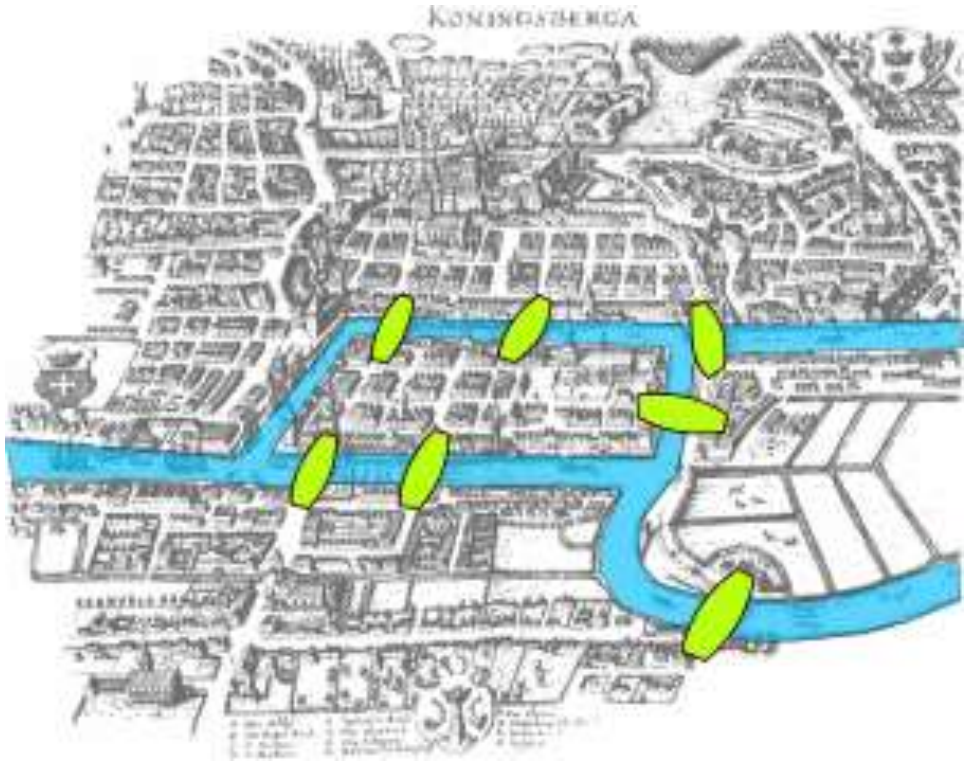
Un grafo G = (V,E) consiste in:

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

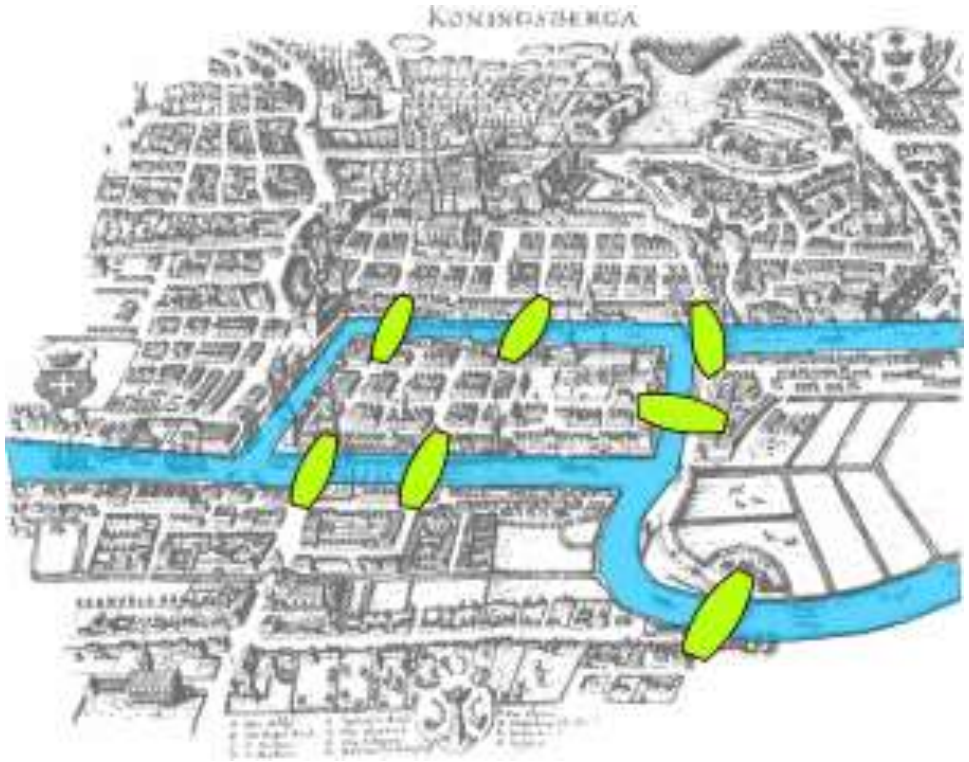
Esempio 1 Grafo di Facebook: V = {utenti Facebook},
E = {amicizie su Facebook}

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

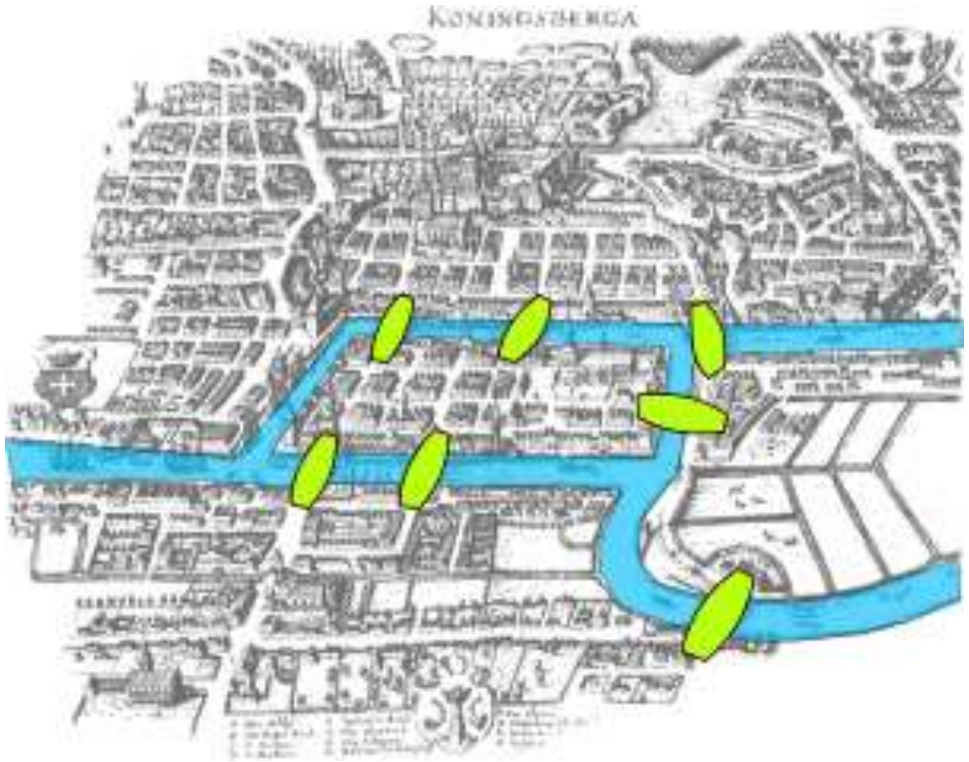
I sette ponti di Königsberg

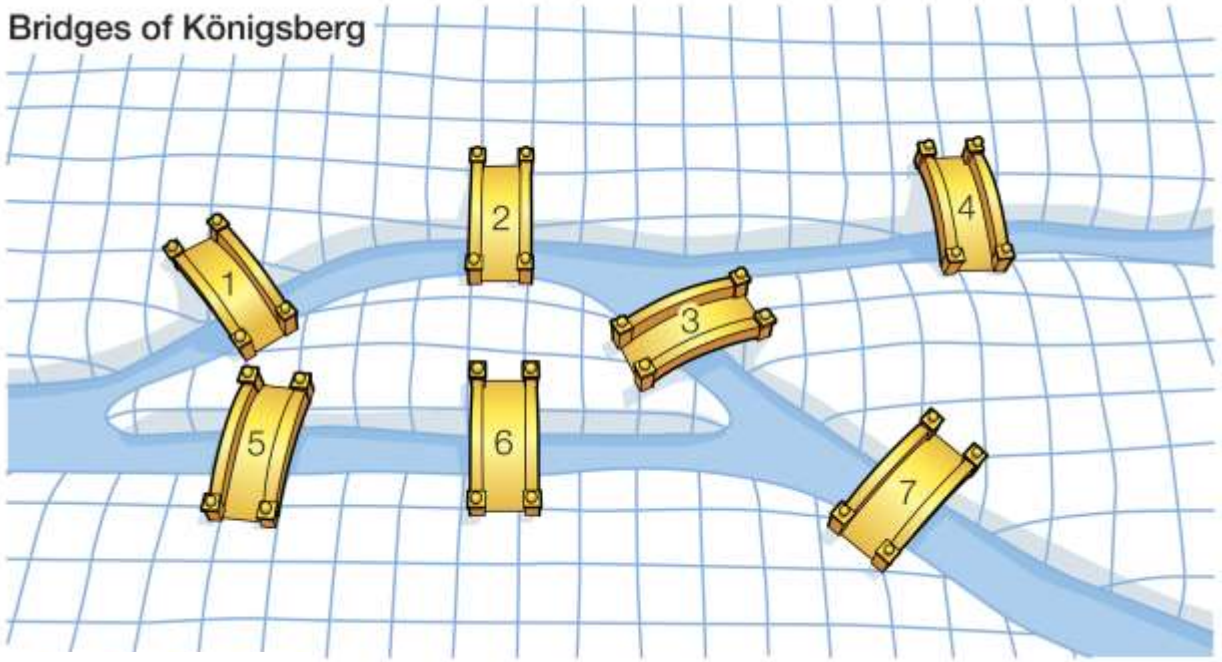


I sette ponti di Königsberg

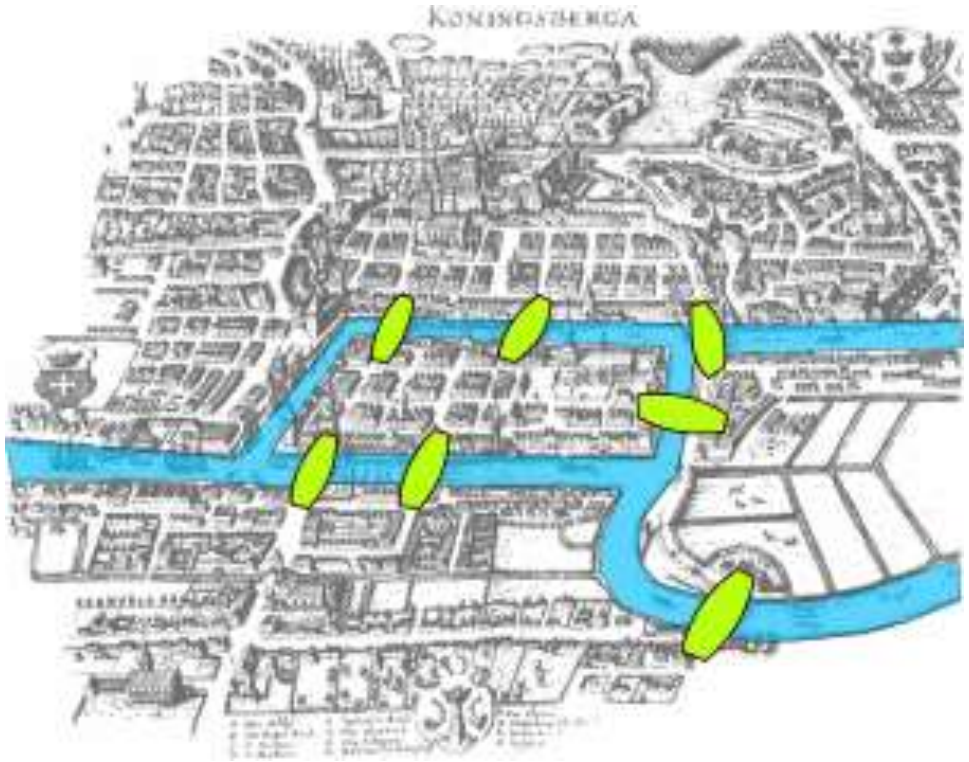


I sette ponti di Königsberg

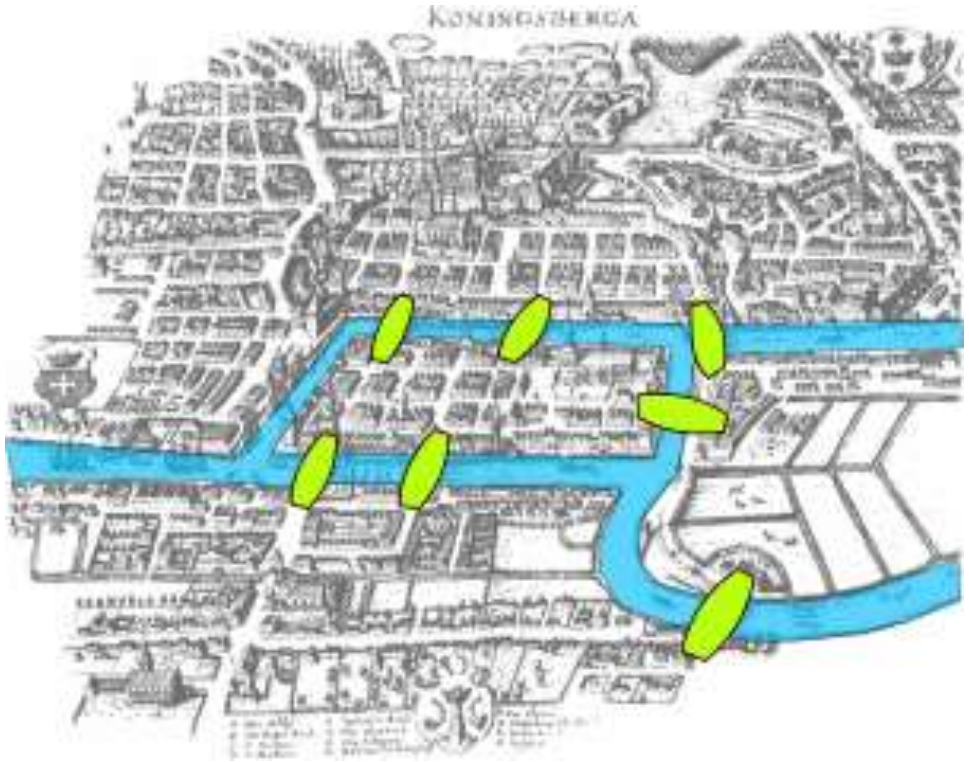




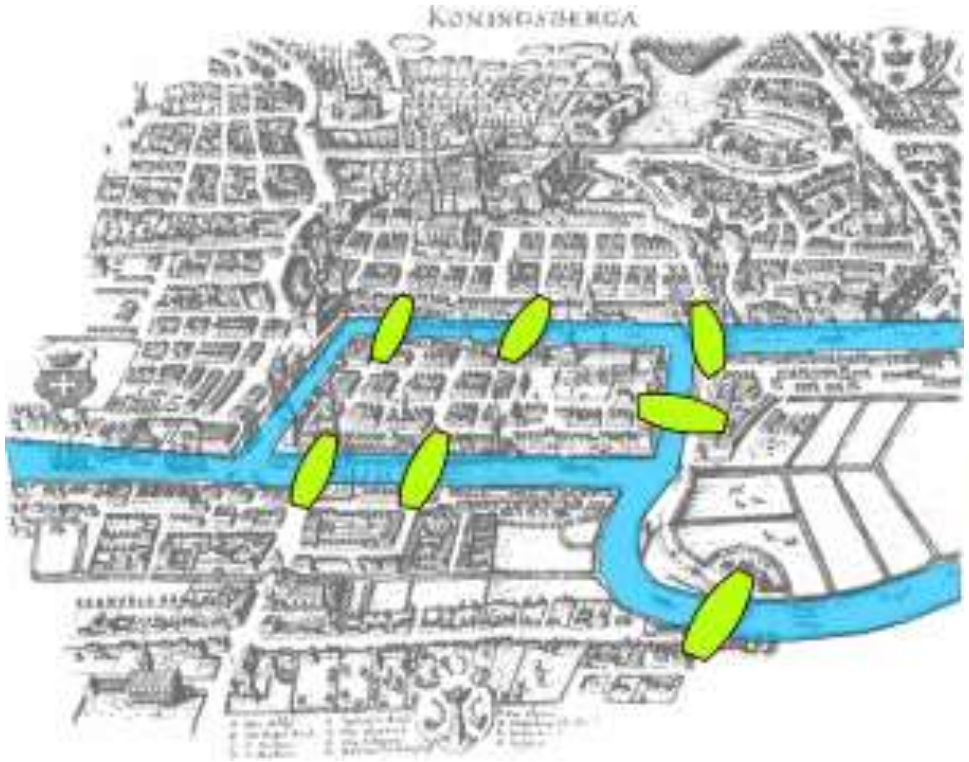
I sette ponti di Königsberg



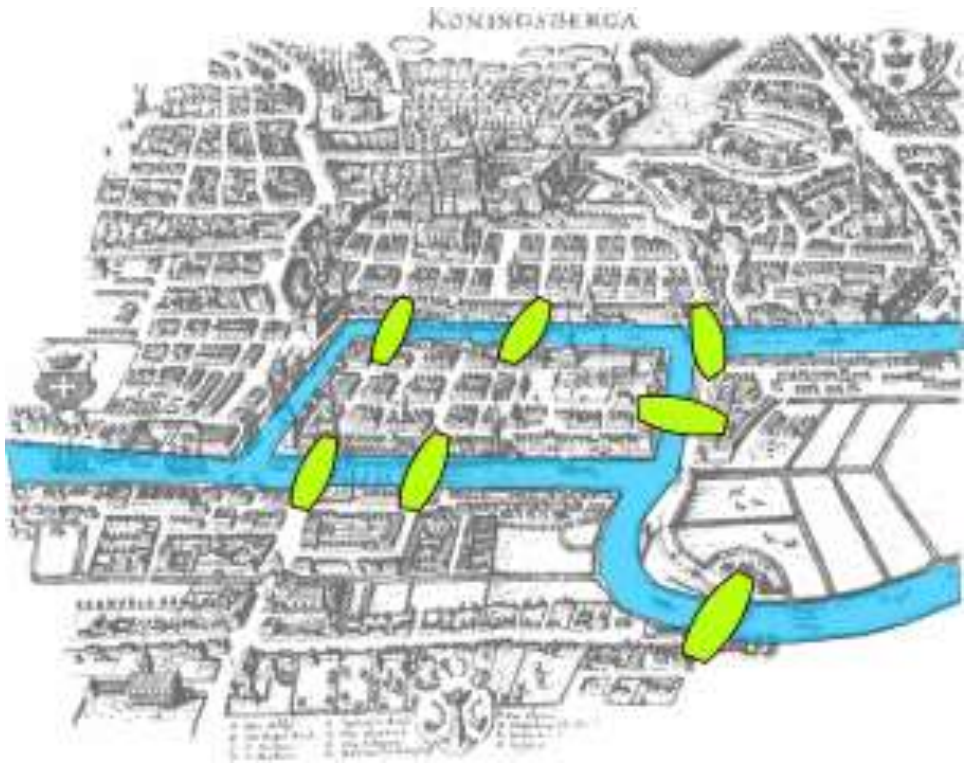
I sette ponti di Königsberg



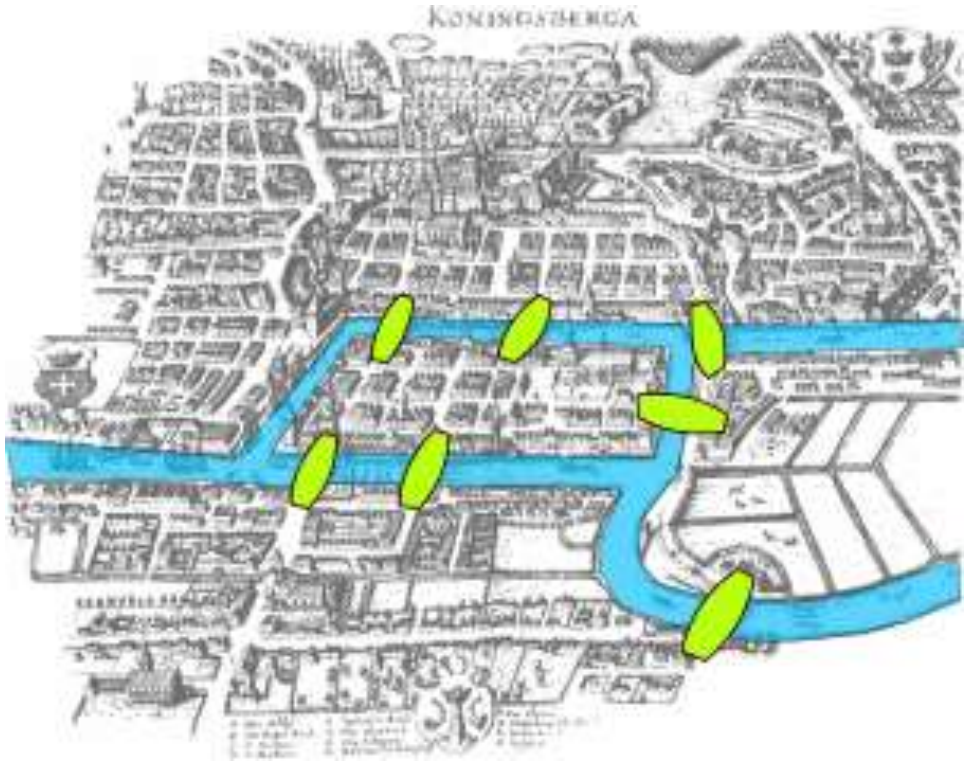
I sette ponti di Königsberg

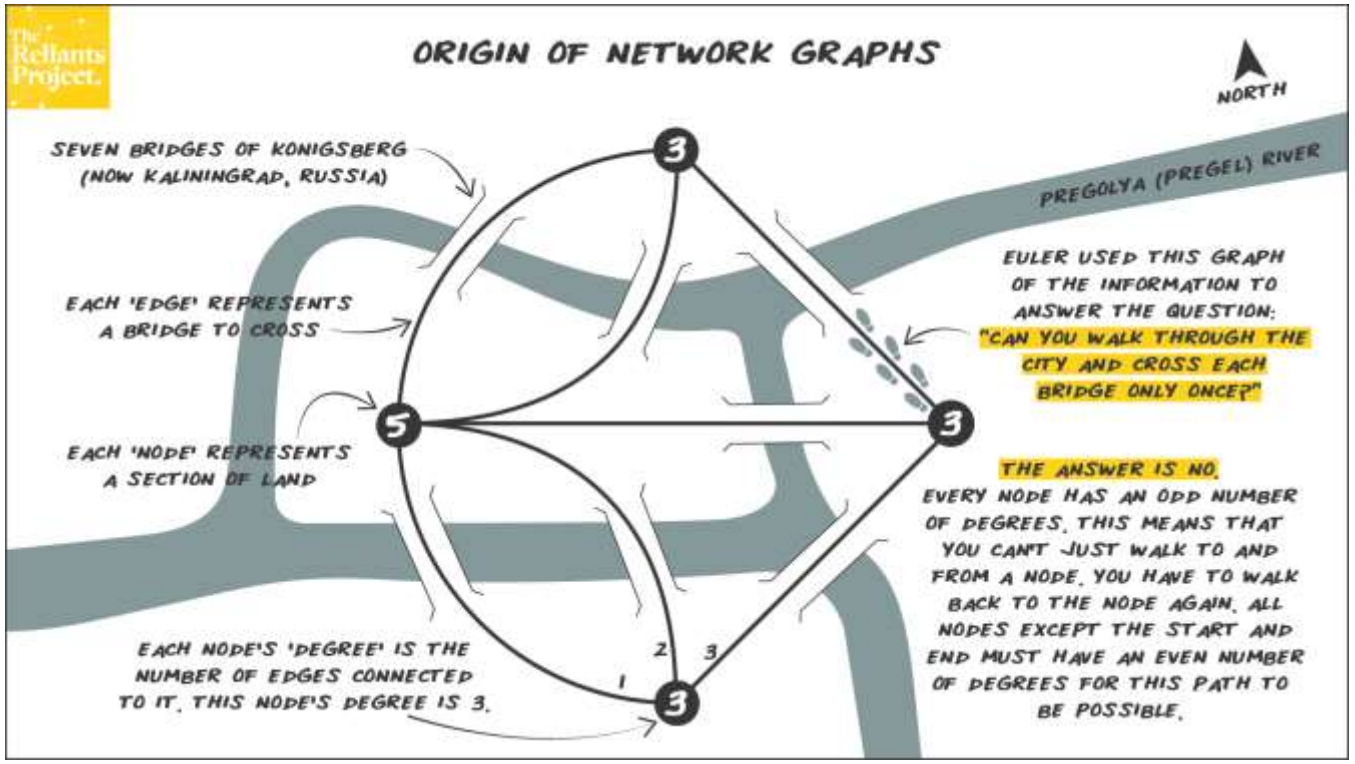


I sette ponti di Königsberg



I sette ponti di Königsberg



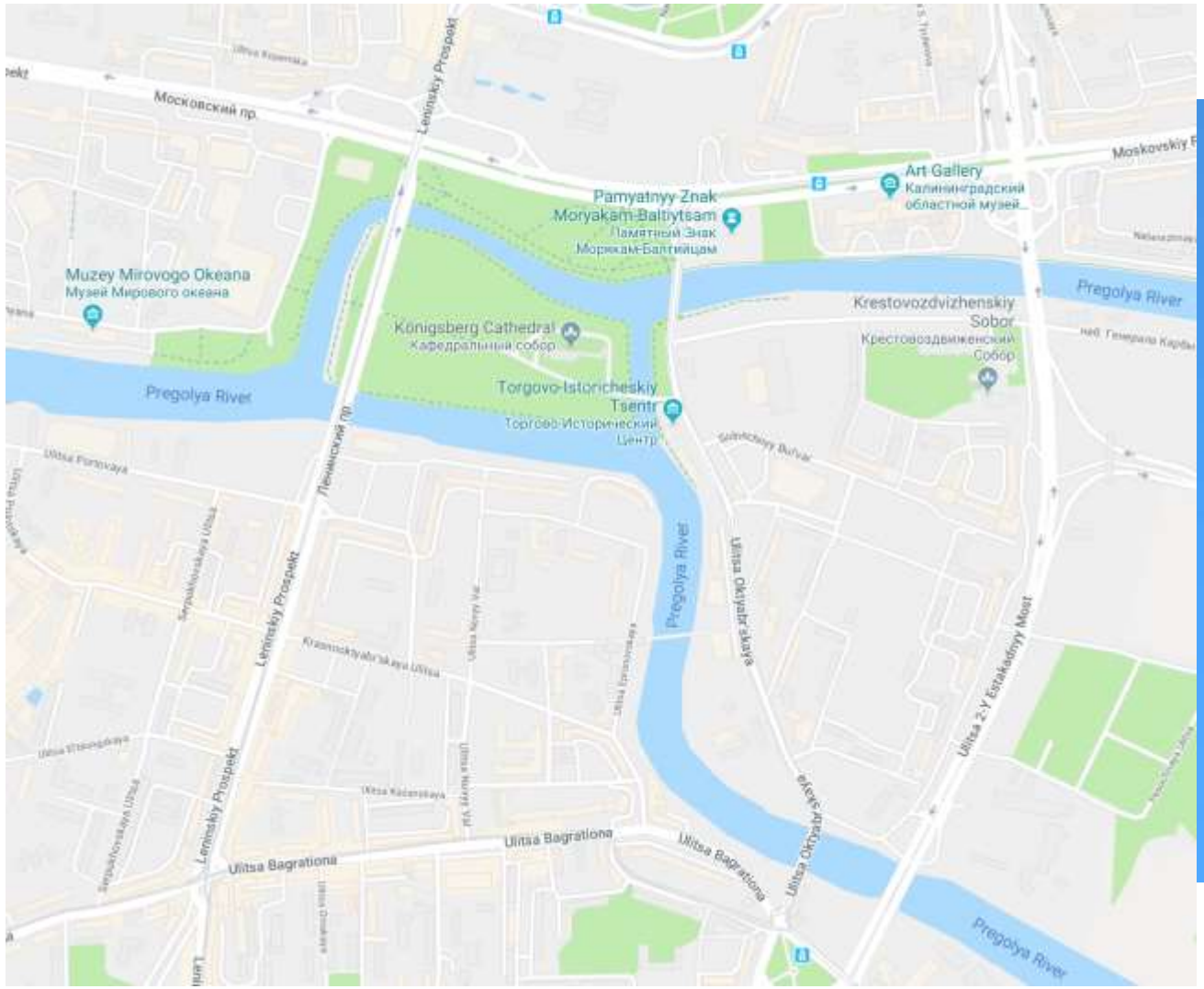


Teorema (Eulero). *Un grafo è Euleriano se e solo se:*

- Tutti i nodi hanno grado pari, oppure

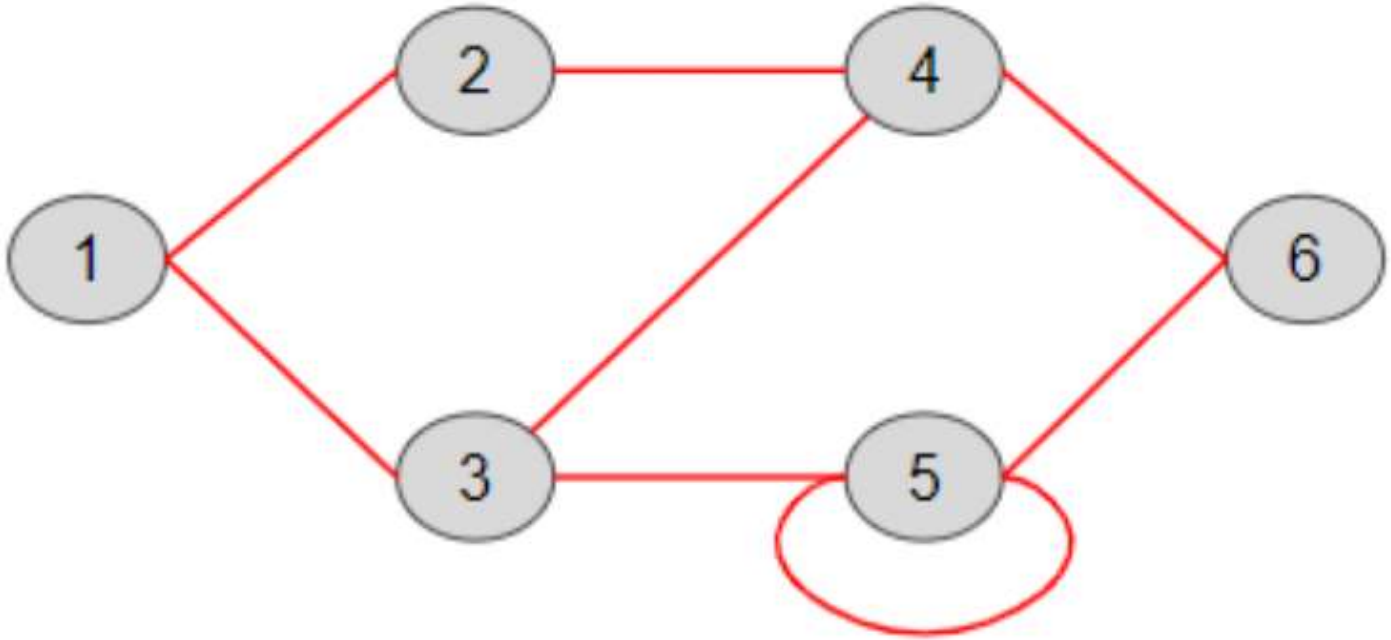
- Soltanto due nodi hanno grado dispari.

Königsberg oggi: Kaliningrad



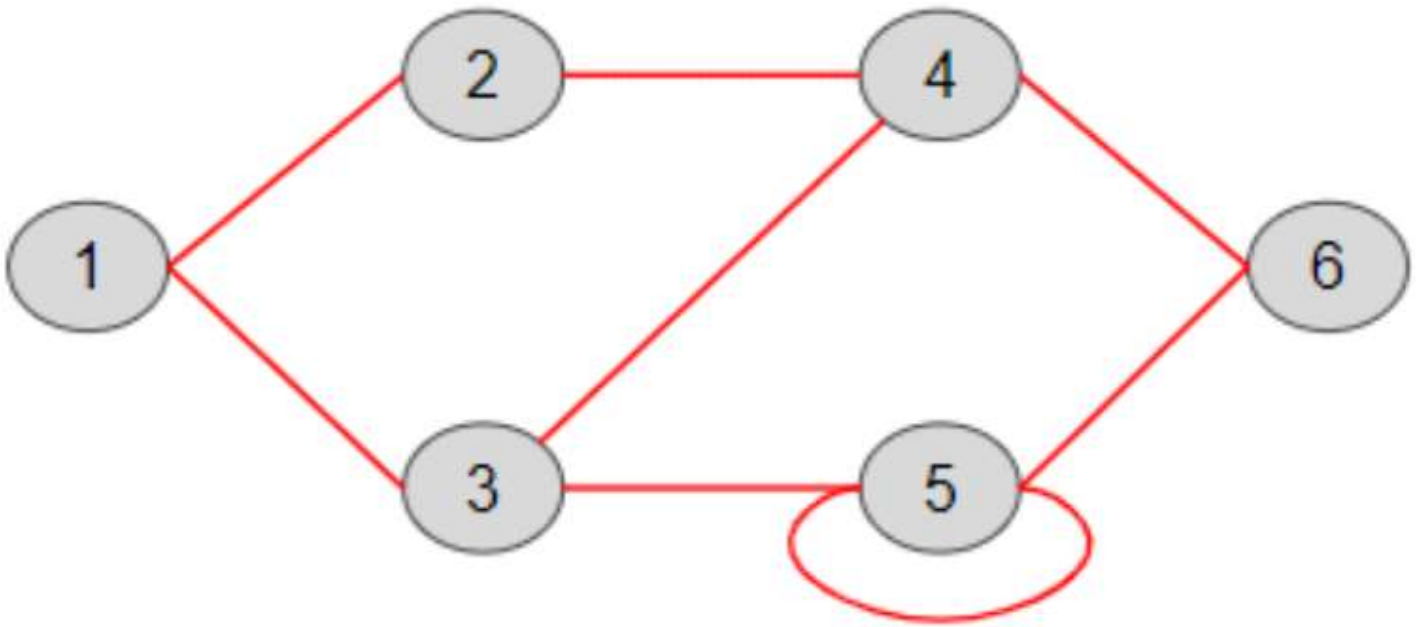
Strutture dati
per rappresentare grafi

GRAFO NON ORIENTATO

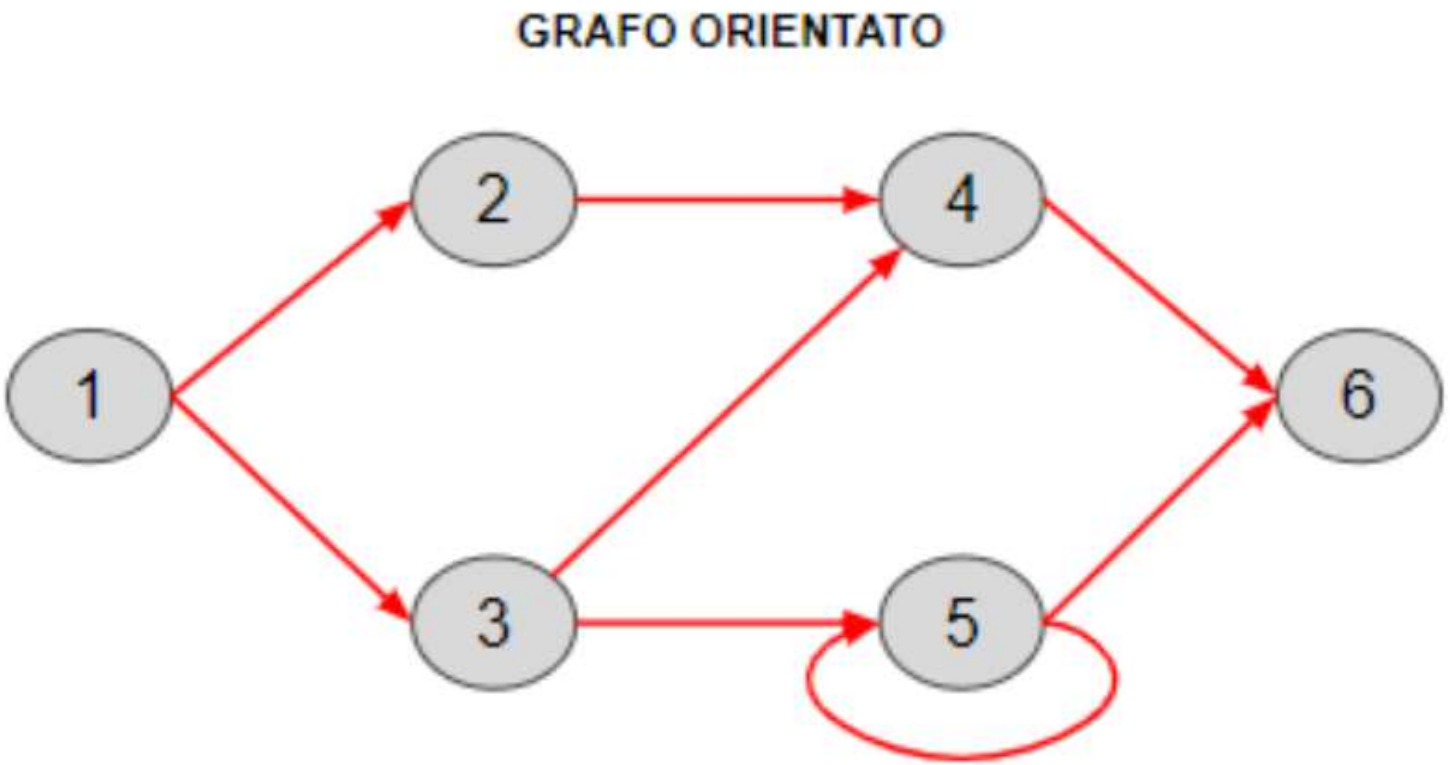


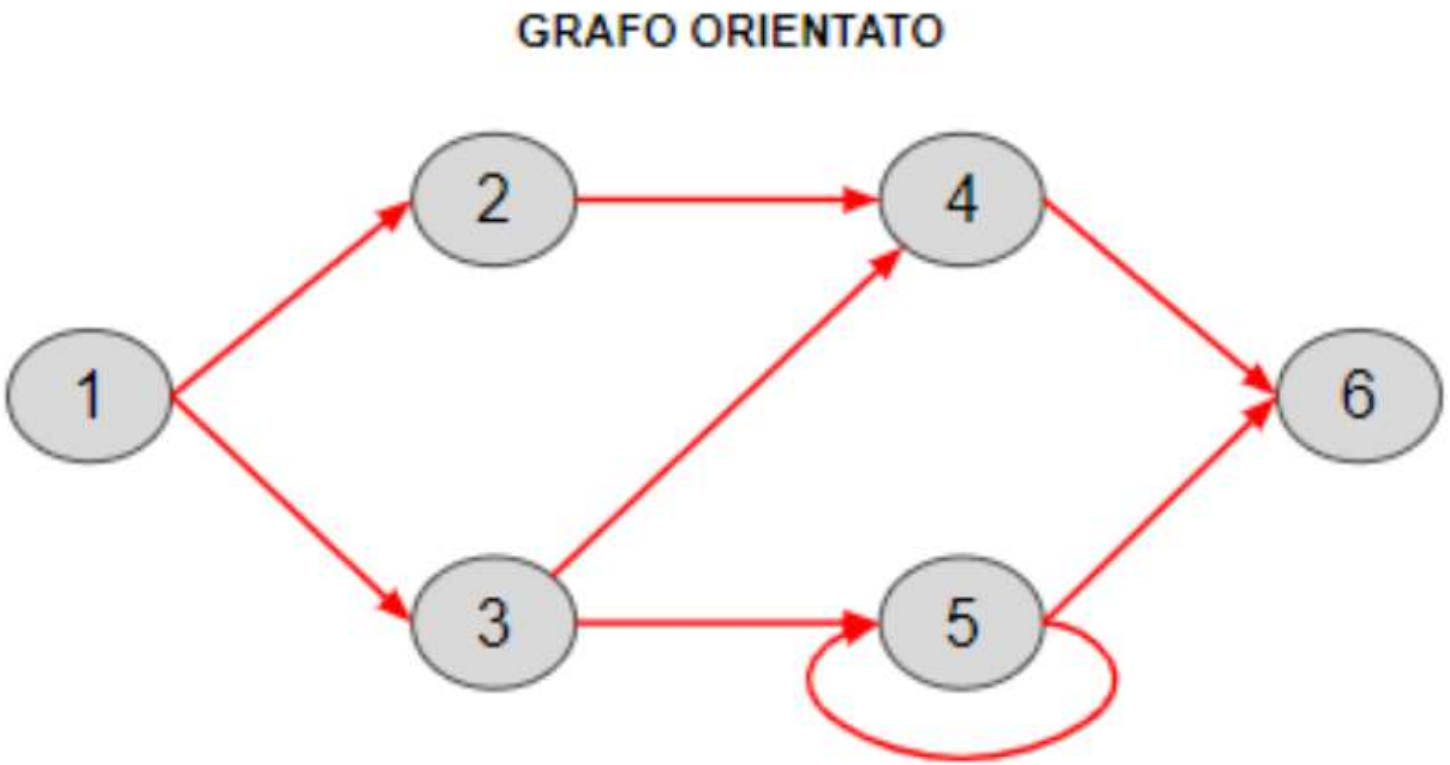
NODI	LISTE DI ADIACENZA
1	2 , 3
2	1 , 4
3	1 , 4 , 5
4	2 , 3 , 6
5	3 , 5 , 6
6	4 , 5

GRAFO NON ORIENTATO



nodi di partenza \ 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


GRAFO ORIENTATO
1
2
3
4
5
6

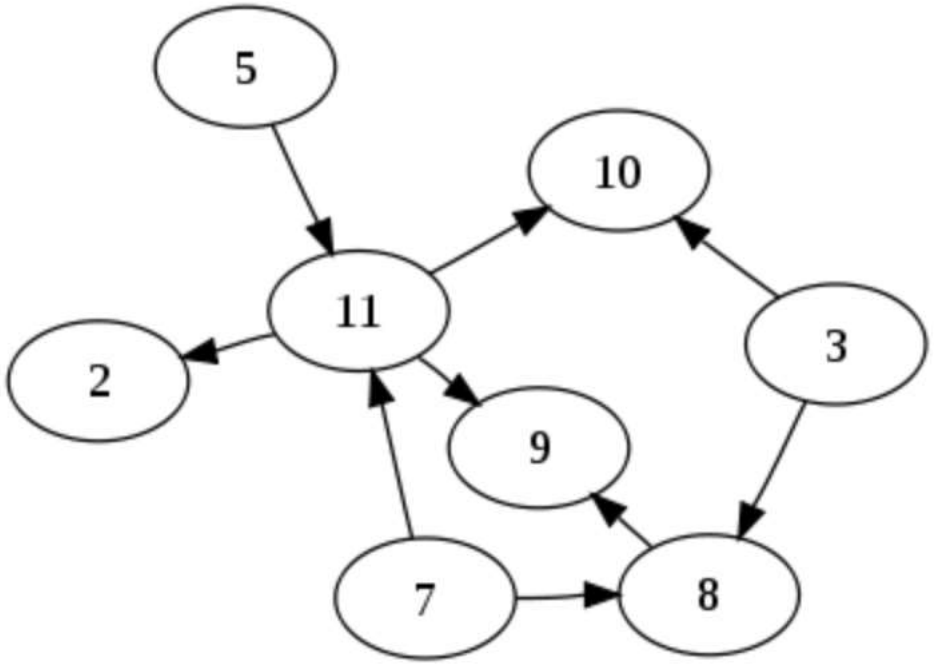

GRAFO ORIENTATO
1
2
3
4
5
6

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)

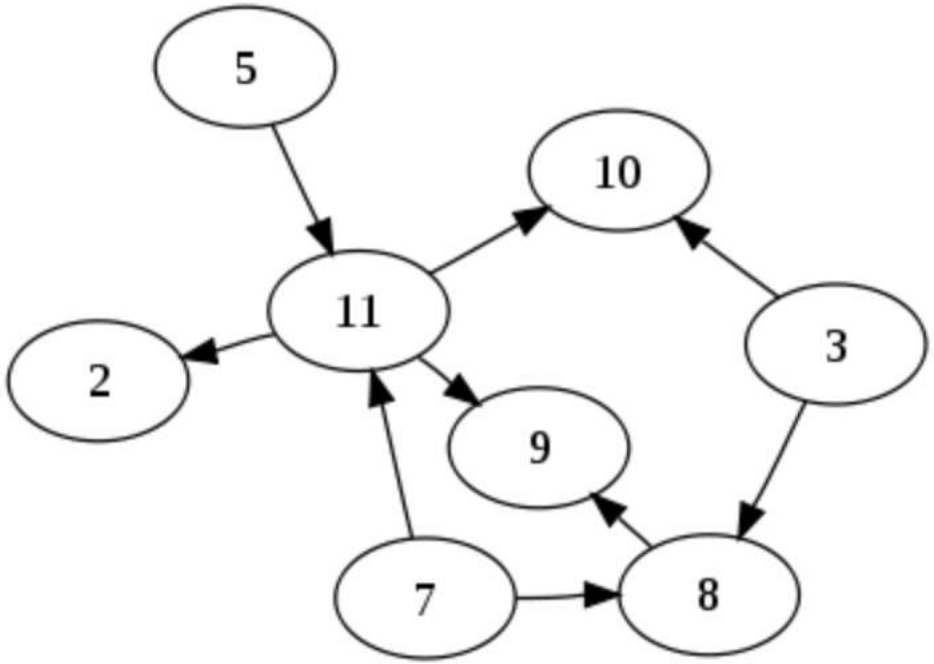


Grafo orientato che non contiene cicli: comunque scegliamo un nodo del grafo, non possiamo mai ritornare allo stesso nodo (percorrendo gli archi del grafo)

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

...

Selezioni regionali 2007

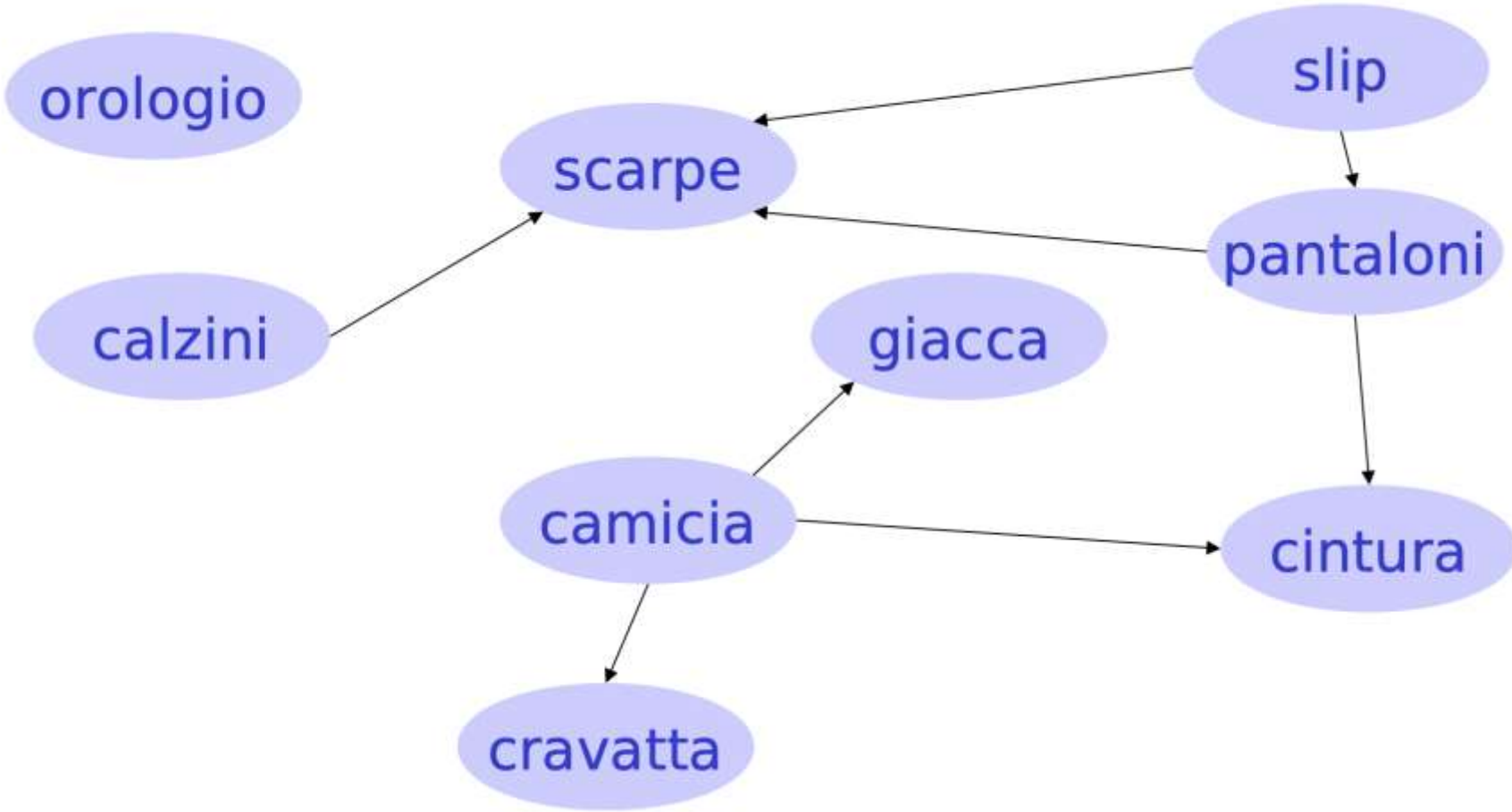
Torero Escamillo (`torero`)

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 consegni 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

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

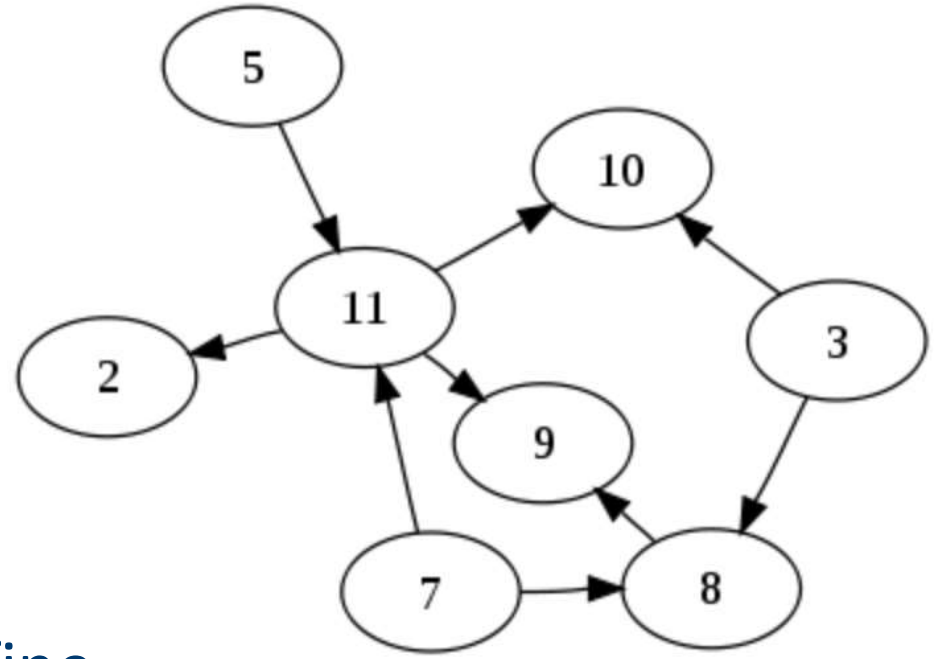
Esempi di input/output

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

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

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

Sunnydale
(Territoriali 2004)

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

Sunnydale (sunny)

Difficoltà D = 2.

Descrizione del problema

Sunnydale è una città che - per ragioni storiche e ambientali - ospita un elevatissimo numero di vampiri.Per ragioni cutanee i vampiri non possono sopportare la luce solare e, storicamente, hanno sempre avuto enormi difficoltà a viaggiare col sole alto nel cielo; l'attraversamento delle gallerie sotterranee di Sunnydale è sempre stato il mezzo preferito dai vampiri per muoversi nella città.I continui crolli delle gallerie hanno creato dei fori nei soffitti, rendendone alcune troppo luminose per un attraversamento tranquillo e sereno.Harmony, una ragazza-vampiro, passeggia per le gallerie di Sunnydale quando il suo amico Spike le telefona per invitarla a casa sua.Purtroppo ella si muove per le gallerie sotterranee secondo una regola tanto semplice quanto tassativa: ad ogni svincolo sceglie sempre e comunque la galleria meno luminosa per paura di rovinare la propria pelle.Sapendo che non esistono due gallerie egualmente luminose, bisogna determinare se Harmony possa raggiungere la casa sotterranea di Spike e, in caso affermativo, quante gallerie le sono necessarie per arrivare.

Dati di input

La prima riga del file `input.txt` è composta da quattro numeri interi N, M, H e S: il primo rappresenta il numero degli svincoli (numerati da 1 a N), il secondo rappresenta il numero delle gallerie, il terzo rappresenta l'indice dello svincolo in cui si trova Harmony quando riceve la telefonata; il quarto, infine, rappresenta l'indice dello svincolo della casa di Spike.Ognuna delle successive M righe descrive una galleria e contiene tre numeri interi A, B e L separati da uno spazio: i primi due rappresentano gli svincoli collegati dalla galleria mentre il terzo rappresenta il suo grado di luminosità.

Dati di output

Il file `output.txt` dovrà contenere un unico numero intero: -1 se Harmony non riuscirà a raggiungere Spike; altrimenti, il numero di gallerie che ella percorrerà prima di raggiungerlo.

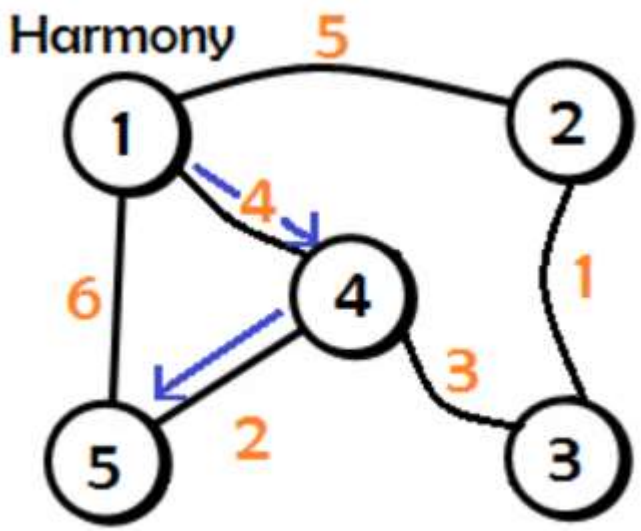
Assunzioni

- $2 \leq N \leq 50000$
- $1 \leq M \leq 50000$
- Non esistono due gallerie con la stessa luminosità L.
- Per ogni galleria, $1 \leq L \leq M$.
- $1 \leq H, S \leq N$

Esempi di input/output

File `input.txt`

```
5 6 1 5
1 2 5
2 3 1
3 4 3
4 5 2
5 1 6
1 4 4
```



File `output.txt`

```
2
```

File `input.txt`

```
3 2 2 1
3 1 2
2 3 1
```

File `output.txt`

```
-1
```

Riflessioni

- Ci serve proprio memorizzare tutti gli archi?

- Come memorizziamo il grafo?
  (Ogni vertice ha un solo arco uscente)

Riflessioni

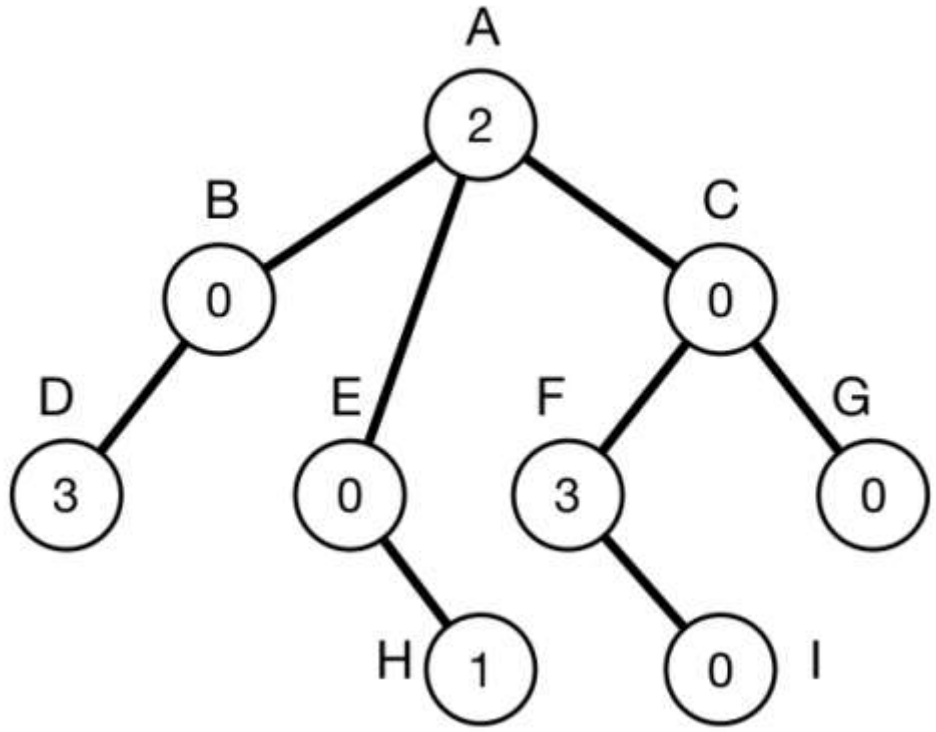
- Come facciamo a non ritornare sullo stesso vertice per evitare di ciclare all'infinito?

Monete a posto
(Gator 2016)

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

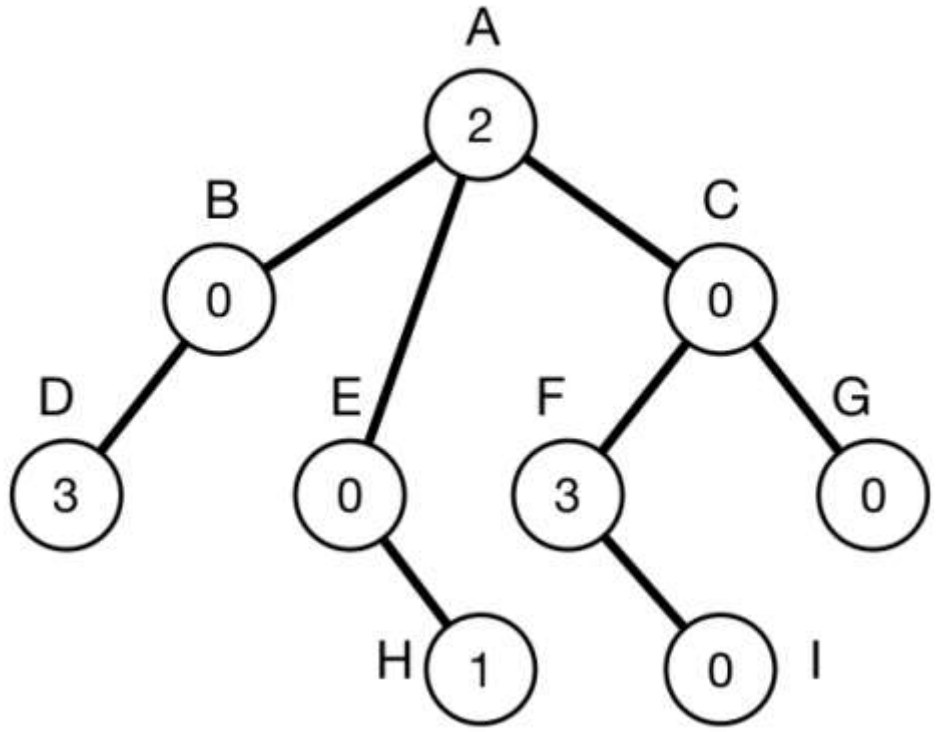
GATOR – Gara Allenamento TOR Vergata 2016

Gara online, 9-10 aprile 2016



Monete a posto (`monete`)

William colleziona monete, e le tiene in un espositore come quello mostrato in figura, in cui ci sono N contenitori numerati da 1 a N appesi uno all'altro tramite una barra di metallo. In particolare, ogni contenitore tranne quello più in alto (il contenitore numero 1) è appeso a esattamente un altro contenitore. Sua sorella ha giocato con le monete, che prima erano una in ogni posizione, e le ha lasciate nella situazione mostrata qui sotto. William ha però una regola per rimetterle a posto: può spostare una sola moneta per volta, da un contenitore a uno collegato. Quante operazioni sono sufficienti a William per fare in modo che in ogni contenitore ci sia una moneta?



Ad esempio, nella situazione mostrata qui sopra, William impiega:

- Una operazione per spostare una moneta da A a E.
- Una operazione per spostare una moneta da F a I.
- Una operazione per spostare una moneta da F a C.
- Una operazione per spostare una moneta da D a B.
- Quattro operazioni per spostare una moneta da D a G (passando per B, A e infine C).

Dall'esempio qui sopra si vede che servono otto operazioni in totale per rimettere a posto le monete, con una moneta per ogni contenitore.

Dati di input

Il file `input.txt` contiene tre righe. La prima riga contiene l'intero N, il numero di contenitori (e di monete) che ha William. La seconda riga contiene i numeri interi $a_1, \ldots, a_N$, che rappresentano il numero di monete contenute nel rispettivo contenitore dopo che queste sono state disordinate. Infine la terza riga contiene $N-1$ interi $b_2, \ldots, b_N$ che indicano che il contenitore j è appeso al contenitore b_j.

Dati di output

Sul file `output.txt` stampare il numero minimo di spostamenti necessari a William per rimettere a posto tutte le monete.

Assunzioni

- $1 \leq N \leq 100\,000$.
- $0 \leq a_i \leq N$ per ogni $i = 1, \ldots, N$.
- $a_1 + \cdots + a_N = N$.
- $1 \leq b_i < i$ per ogni $i = 1, \ldots, N$.

Esempi di input/output

input.txt	output.txt
9 2 0 0 3 0 3 0 1 0 1 1 2 1 3 3 5 6	8
10 0 0 1 2 1 1 2 0 3 0 1 2 2 2 4 3 5 1 8	11

Compiti a casa
(Esercizi Olimpiadi)

Compiti a casa

Prova a risolvere i seguenti problemi:

- Piano di studi https://training.olinfo.it/#/task/luiss_piano/statement
- Corsa mattutina https://training.olinfo.it/#/task/footing/statement
- Grattacieli commemorativi https://training.olinfo.it/#/task/oii_grattacieli/statement
- Imaginary Grasshopper: https://training.olinfo.it/#/task/ois_grasshopper/statement
- Bus trip https://training.olinfo.it/#/task/ois_trip/statement
- Appetito aracnide https://training.olinfo.it/#/task/tecla/statement
- Incendia lo trabucco https://training.olinfo.it/#/task/po_trabucco/statement