

Luiss
Libera Università Internazionale
degli Studi Sociali Guido Carli

Corso di preparazione per la selezione territoriale delle Olimpiadi di Informatica

Grafi: visite e cammini minimi

Irene Finocchi

LUISS 





Preferibile se hai problemi di connessione, ma non permette di condividere le tue soluzioni

Lezioni

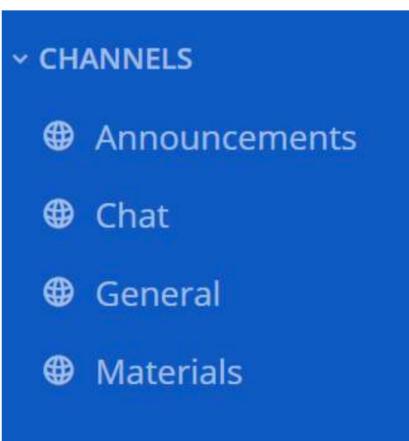
Puoi accedere alle lezioni tramite YouTube o Webex



Licenza limitata a 1000 utenti; puoi provare l'ebrezza di diventare **panelist** e condividere la tua soluzione ai problemi considerati!



Repetita iuvant (sed stufant!)



- Non avrai altra chat al di fuori di **mattermost**
- <https://coding.luiss.ml/mattermost/> (canale Chat)
 - Usa il canale Chat per le domande
 - Potete rispondere anche voi!
- Controlla i canali: spesso troverai qui le risposte alle tue domande!
- Non inquinare i canali! Contribuisci a mantenerli puliti e non postare contenuti impropri. Grazie!
- Oltre a quella di oggi, manca una sola lezione, poi il **contest!**



E finalmente... il contest!

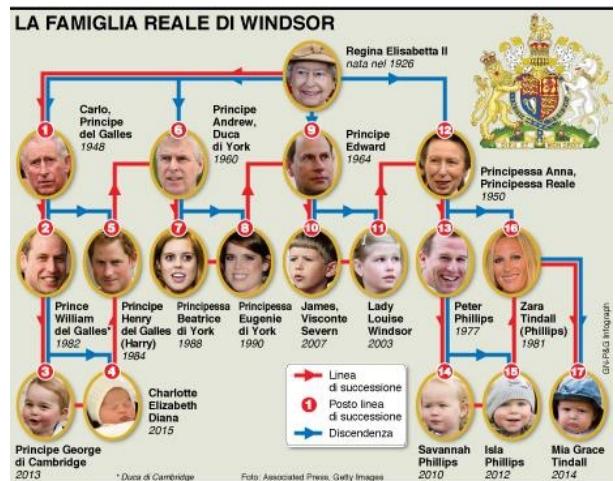
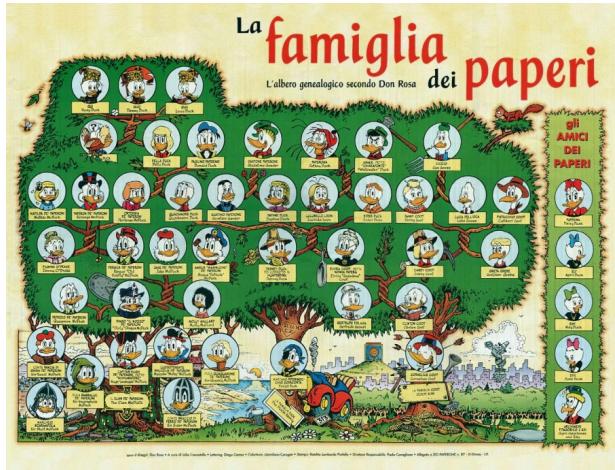
- Dovrete accedere alla piattaforma utilizzando la password generata al momento della registrazione (**la stessa password che usate per Mattermost**)
- Se qualcuno ha cambiato la password in Mattermost, contattate un tutor
- Il concorso inizia a **mezzanotte** e dura **24 ore**
- C'è una **finestra di 5 ore** dal primo accesso alla piattaforma
- Tra pochi giorni pubblicheremo il **link alla piattaforma** del concorso e ulteriori istruzioni nella **sezione degli annunci** su Mattermost

Programma di oggi

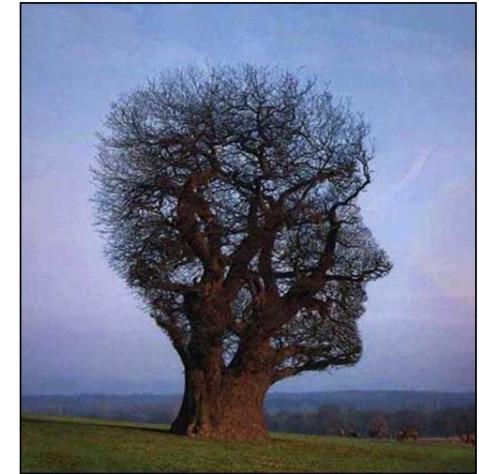
1. Soluzione di alcuni esercizi assegnati la scorsa settimana
2. Recap su visite di alberi
3. Estensione a visite di grafi: in profondità (DFS) e in ampiezza (BFS)
4. Due esempi di problemi su grafi
 - Un problema dalle OIS (2014): Spesa
 - Un problema dalle Territoriali (2009): Depurazione dell'acqua
5. Cammini minimi su grafi
 - Rilassamenti, l'algoritmo di Dijkstra, cammini minimi su DAG
 - Un problema dalle Territoriali (2012): Il tesoro del pirata Barbablù
6. Esercizi per casa

Esercizi dalla scorsa settimana

- 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



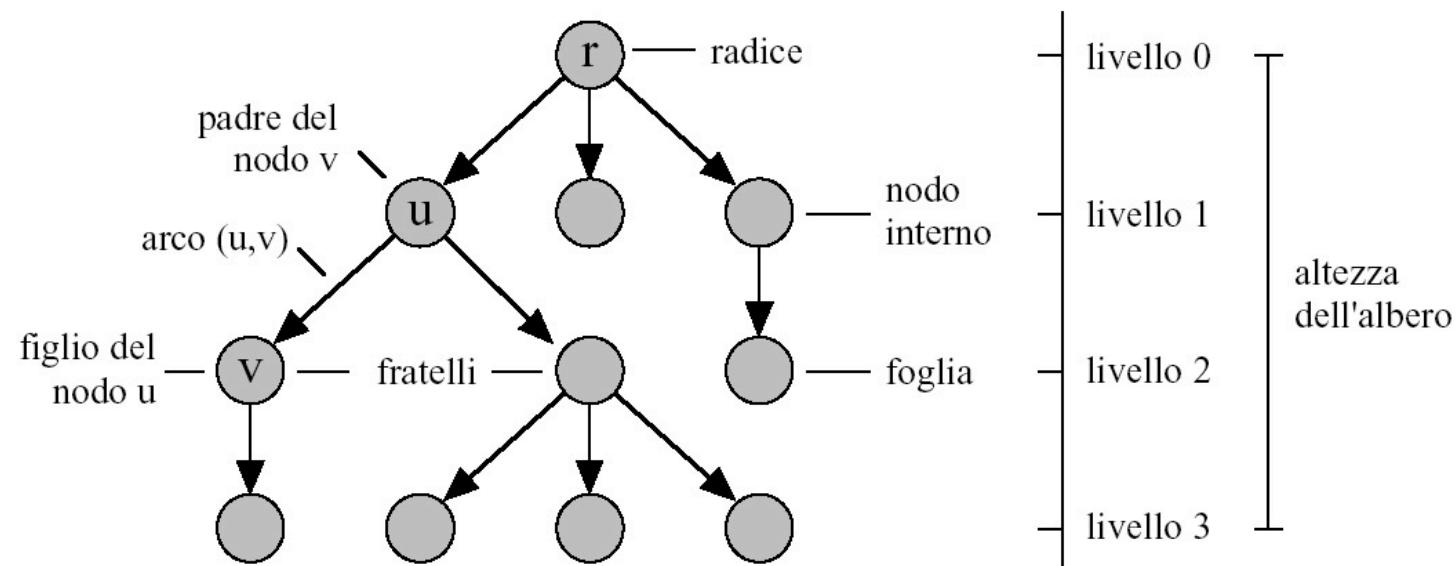
*Mandatory joke about
trees in computer science
turned upside-down.*



Alberi: ricapitoliamo

Alberi radicati: grafi aciclici con radice

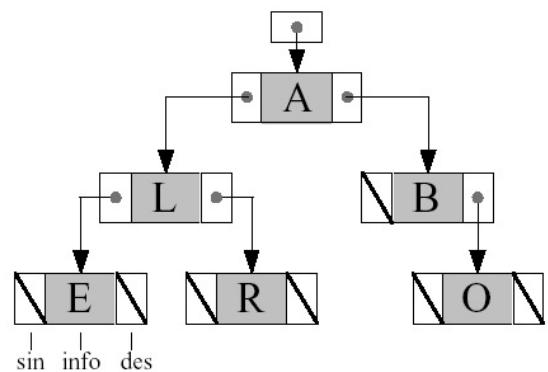
Organizzazione gerarchica dei dati



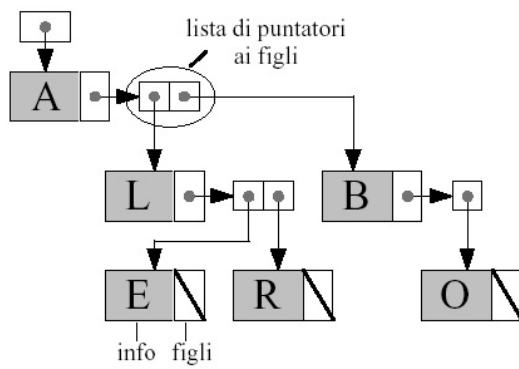
Dati contenuti nei nodi, relazioni gerarchiche definite dagli archi che li collegano

Rappresentazioni di alberi

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

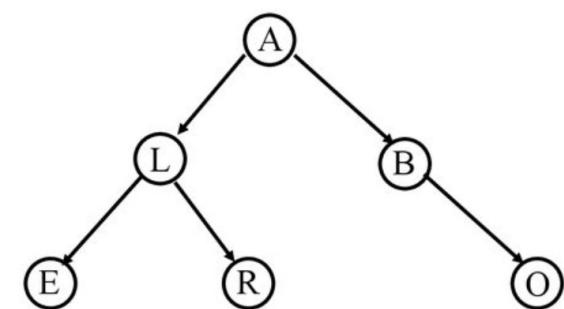


LUISS



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

Qual è la principale
differenza degli alberi
rispetto ai grafi?

Visite di alberi

- Algoritmi che consentono l'accesso sistematico ai nodi e agli archi di un albero
- Gli algoritmi di visita si distinguono in base al particolare ordine di accesso ai nodi



Algoritmo di visita in profondità

L'algoritmo di visita in profondità (**DFS = Depth First Search**):

- parte da r
- procede visitando nodi di figlio in figlio fino a raggiungere una foglia
- retrocede poi al primo antenato che ha ancora figli non visitati (se esiste)
- ripete il procedimento a partire da uno di quei figli

Può essere implementato usando una pila (stack)

Algoritmo di visita in profondità

Versione ricorsiva (per alberi binari):

algoritmo visitaDFSRicorsiva(*nodo r*)

1. **if** (*r* = null) **then return**

2. *visita il nodo r*

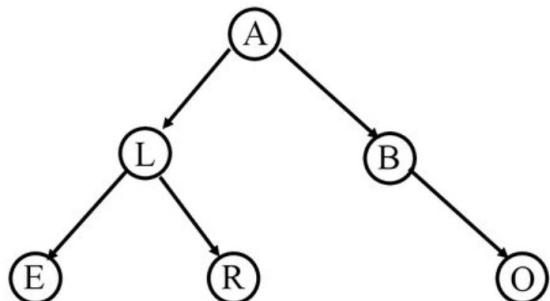
3. visitaDFSRicorsiva(*figlio sinistro di r*)

4. visitaDFSRicorsiva(*figlio destro di r*)

Preordine

Simmetrica

Postordine



- Visita in preordine: A L E R B O.
- Visita simmetrica: E L R A B O.
- Visita in postordine: E R L O B A.

Algoritmo di visita in ampiezza

L'algoritmo di visita in ampiezza (BFS):

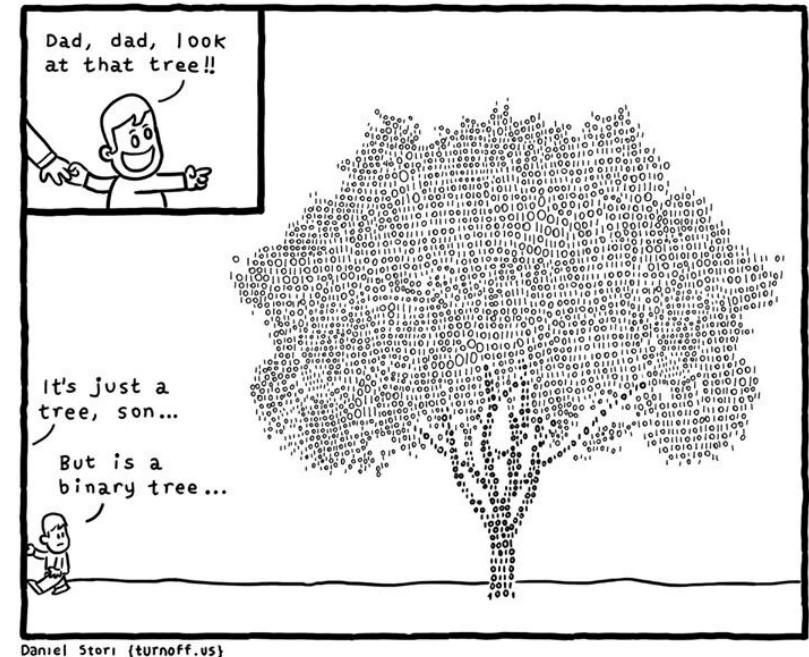
- parte da r
- procede visitando nodi per livelli successivi

Un nodo sul livello i può essere visitato solo se tutti i nodi sul livello $i-1$ sono stati visitati

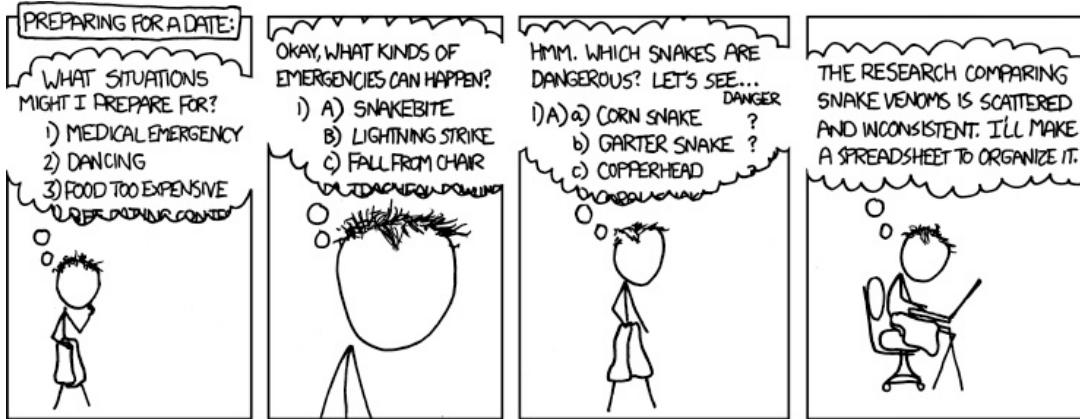
Algoritmo di visita in ampiezza

Versione iterativa (per alberi binari):

```
algoritmo visitaBFS(nodo r)
    Coda C
    C.enqueue(r)
    while (not C.isEmpty()) do
        u  $\leftarrow$  C.dequeue()
        if (u  $\neq$  null) then
            visita il nodo u
            C.enqueue(figlio sinistro di u)
            C.enqueue(figlio destro di u)
```



Daniel Storl {turnoff.us}



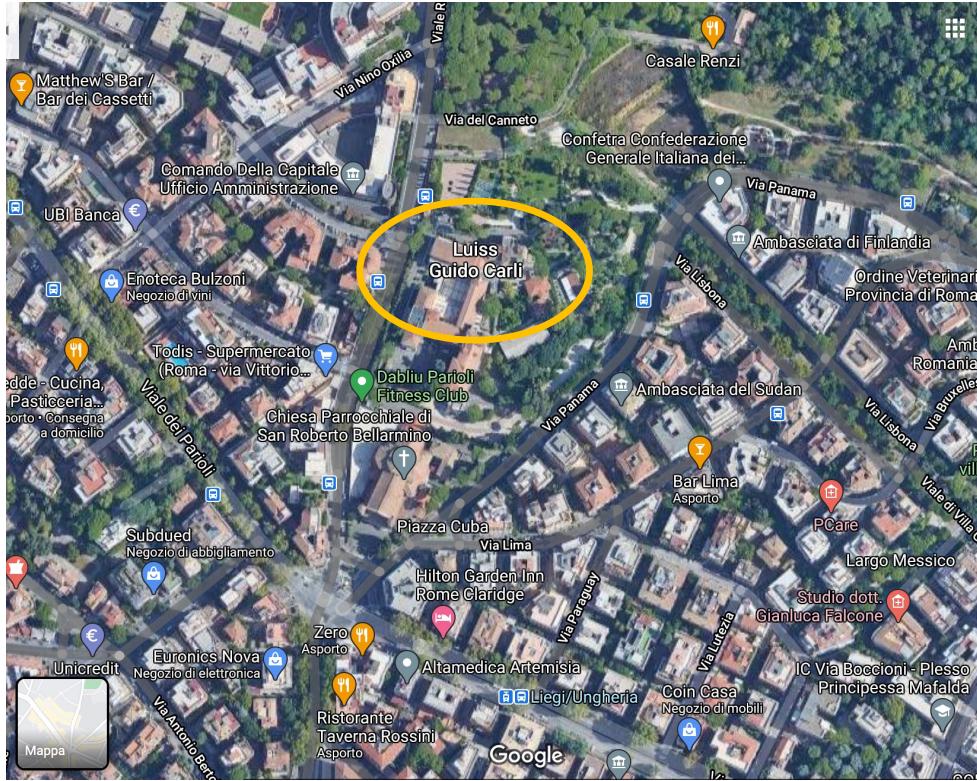
Visite di grafi: DFS vs BFS

xkcd.com

DFS

A breadth-first search makes a lot of sense for dating in general, actually; it suggests dating a bunch of people casually before getting serious, rather than having a series of five-year relationships one after the other.





Come possiamo esplorare in modo sistematico ogni incrocio in questa mappa a partire dalla sede della Luiss?

Analogie con gli alberi

- *Come per gli alberi*, una visita (o attraversamento) di un grafo G permette di esaminare i nodi e gli archi di G in modo sistematico
- Problema di base in moltissime applicazioni
- *Come per gli alberi*, esistono vari tipi di visite con diverse proprietà, in particolare:
 - visita in ampiezza (BFS=breadth first search)
 - visita in profondità (DFS=depth first search)

Differenze rispetto agli alberi

- *Diversamente dagli alberi, un nodo può essere incontrato varie volte durante l'esplorazione di un grafo*
- Viene quindi mantenuto un **sistema di marcatura**, per evitare di visitarlo più volte
- Un nodo viene marcato quando viene incontrato per la prima volta: la marcatura può essere mantenuta tramite un vettore di bit di marcatura
- La visita genera un **albero di copertura T** del grafo

Visita in ampiezza: BFS

algoritmo visitaBFS(*vertice s*) → *albero*

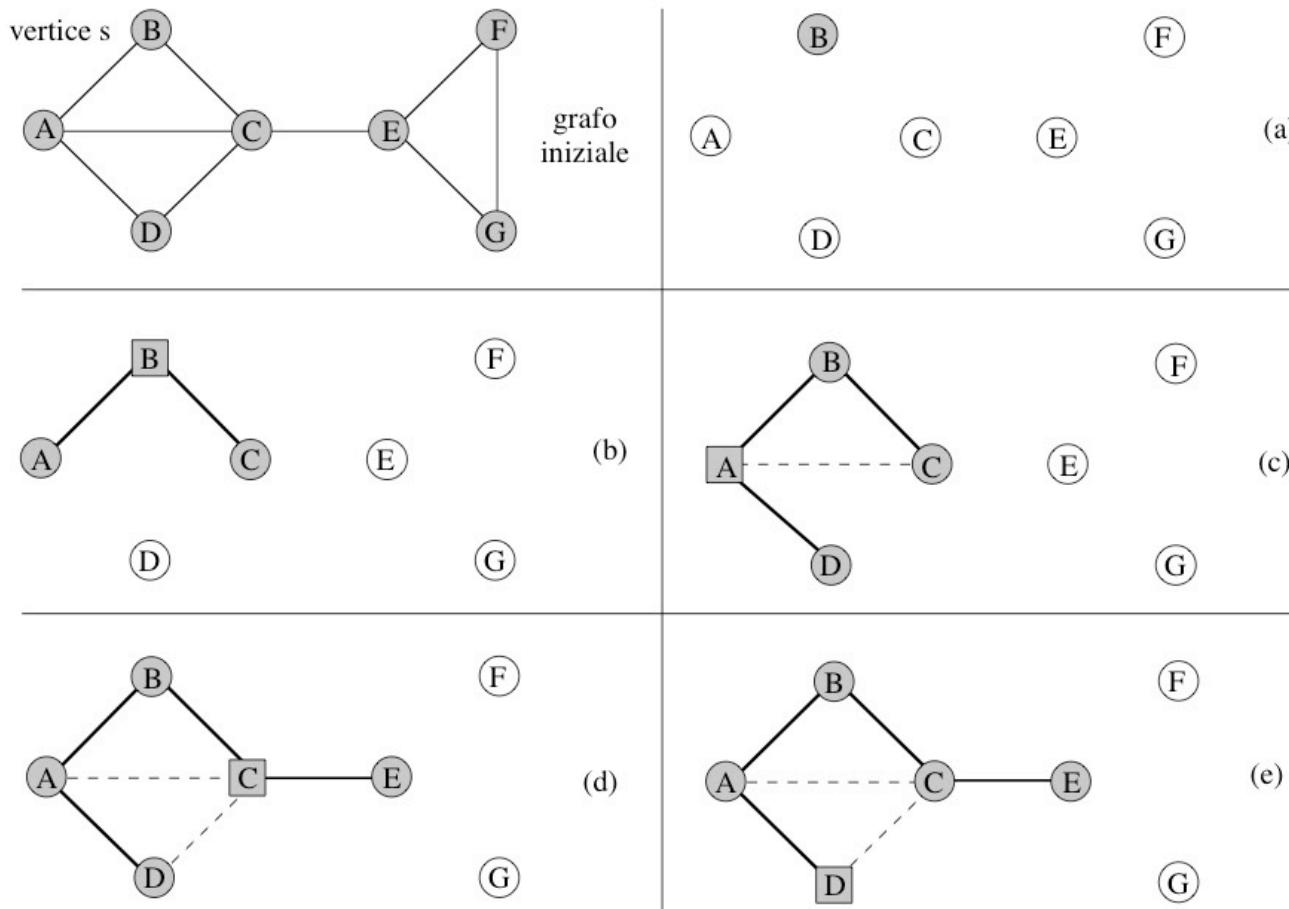
1. rendi tutti i vertici non marcati
2. $T \leftarrow$ albero formato da un solo nodo *s*
3. Coda F
4. marca il vertice *s*
5. F.enqueue(*s*)
6. **while** (not F.isEmpty()) **do**
7. $u \leftarrow$ F.dequeue()
8. **for each** (arco (*u, v*) in *G*) **do**
9. **if** (*v* non è ancora marcato) **then**
10. F.enqueue(*v*)
11. marca il vertice *v*
12. rendi *u* padre di *v* in *T*
13. **return** *T*

- Usa una coda (queue)
- Accesso FIFO (First in First Out)

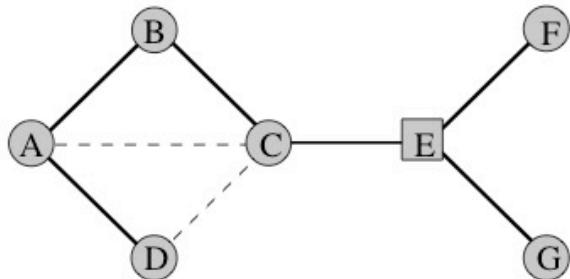
Esempio: animazione BFS

<https://www.cs.usfca.edu/~galles/visualization/BFS.html>

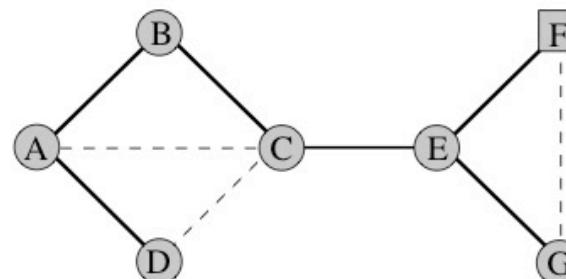
Esempio: grafo non orientato (1/2)



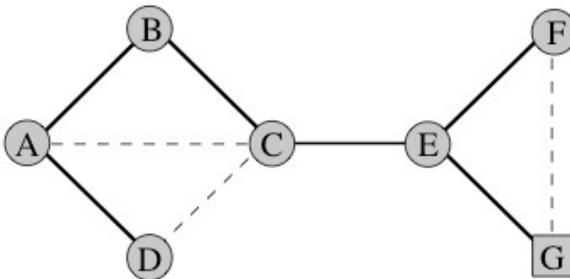
Esempio: grafo non orientato (2/2)



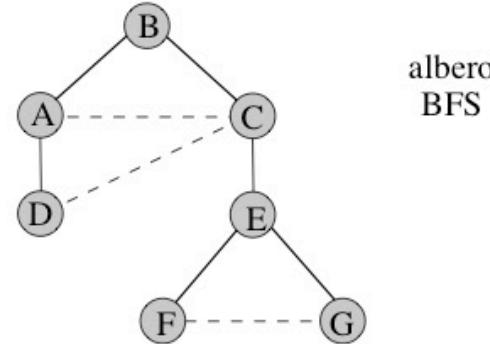
(f)



(g)



(h)



albero
BFS

Una proprietà molto utile

- Per ogni nodo v , il **livello** di v nell'albero di copertura BFS è **pari alla distanza di v dalla sorgente s**
- Distanza tra s e v = numero di archi nel cammino più breve da s a v
- (Vedremo poi un'estensione della definizione di distanza su grafi pesati)

Visita in profondità: DFS

procedura visitaDFSRicorsiva(*vertice v, albero T*)

1. *marca e visita il vertice v*
2. **for each** (arco (v, w)) **do**
3. **if** (*w* non è marcato) **then**
4. aggiungi l'arco (v, w) all'albero *T*
5. visitaDFSRicorsiva(*w, T*)

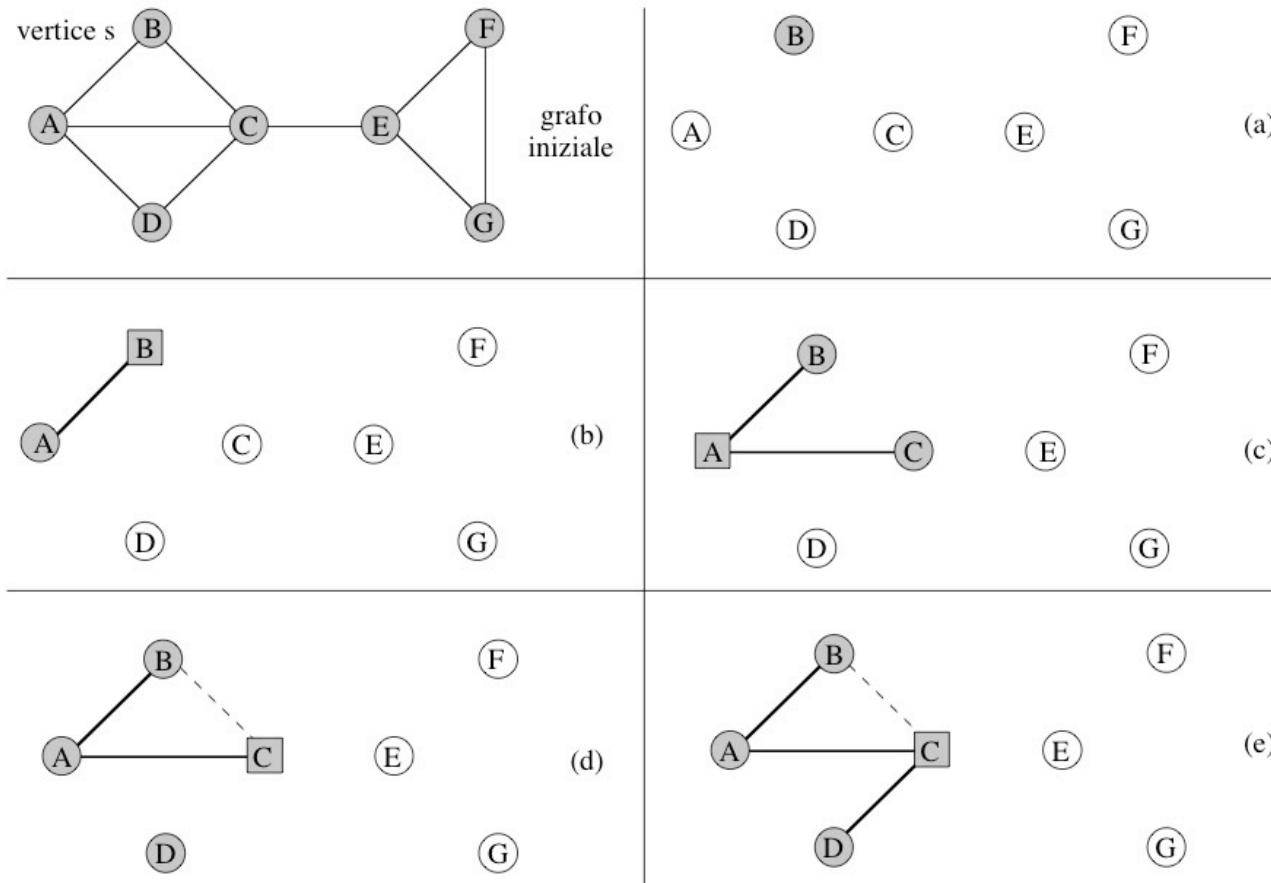
algoritmo visitaDFS(*vertice s*) \rightarrow *albero*

6. $T \leftarrow$ albero vuoto
7. visitaDFSRicorsiva(*s, T*)
8. **return** *T*

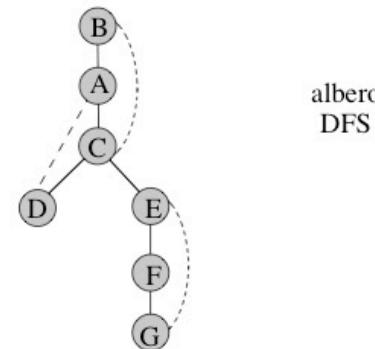
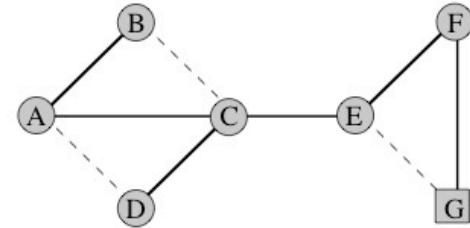
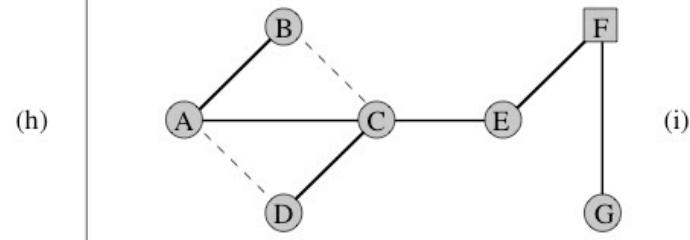
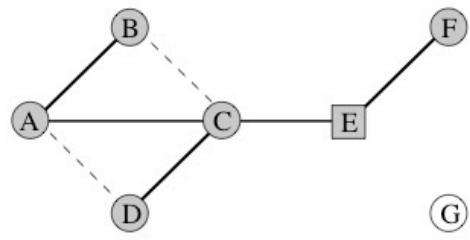
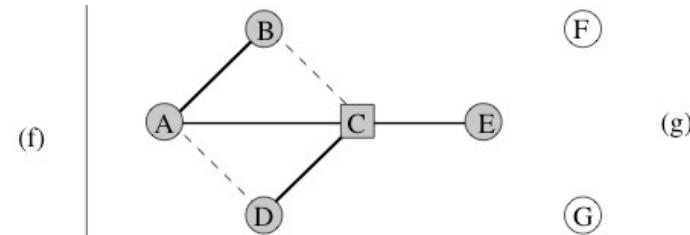
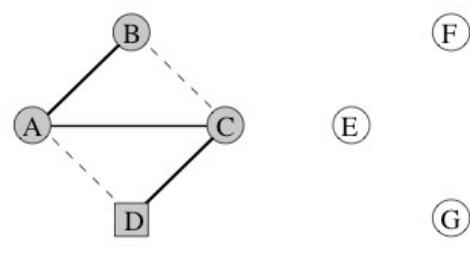
Esempio: animazione DFS

<https://www.cs.usfca.edu/~galles/visualization/DFS.html>

Esempio: grafo non orientato (1/2)



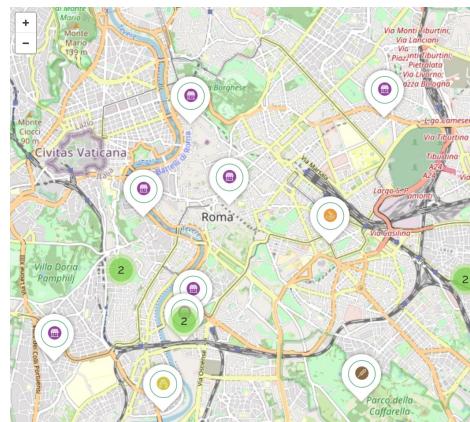
Esempio: grafo non orientato (2/2)



Irene Finocchi / Preparazione Olimpiadi Informatica

Spesa

Olimpiadi di Informatica a Squadre (2014)



Spesa lampo (spesa)

Limite di tempo: 1.0 secondi

Limite di memoria: 256 MiB

Gabriele ha appena finito la sua ultima consulenza, e si prepara ad andare a casa della nonna per un meritato pranzo. Date le precedenti esperienze, decide però che è saggio prima passare da un supermercato a procurarsi un po' di bicarbonato, nel caso in cui i suoi propositi di mangiare il meno possibile falliscano miseramente dopo la dodicesima fetta di tiramisù.

Per non fare aspettare la nonna, questa deviazione deve costare il minor tempo possibile. Per questo Gabriele si è procurato la mappa dei supermercati della città, composta da N punti di interesse (incroci, piazze, edifici importanti) e M vie (a doppio senso di marcia) che collegano i punti di interesse. In particolare, K di questi punti rappresentano dei supermercati, mentre il punto di interesse indicato col numero 1 rappresenta il palazzo della *SteamPower S.P.A.*, dove si trova Gabriele in questo momento, e il punto numerato N è il condominio dove abita la nonna.

Data la mappa sopra descritta, Gabriele si chiede quanto tempo ci metterà (al minimo) per raggiungere la nonna, dovendo prima passare da un supermercato, e sapendo che ci vuole 1 minuto a percorrere ognuna delle M strade.

Implementazione

Dovrai sottoporre esattamente un file con estensione .c, .cpp o .pas.

☞ Tra gli allegati a questo task troverai un template (`spesa.c`, `spesa.cpp`, `spesa.pas`) con un esempio di implementazione da completare.

Se sceglierai di utilizzare il template, dovrai implementare la seguente funzione:

C/C++	<code>int compra(int N, int M, int K, int supermercati[], int da[], int a[]);</code>
Pascal	<code>function compra(N, M, K: longint; var supermercati, da, a: array of longint): longint;</code>

In cui:

- L'intero N rappresenta il numero di punti di interesse della città (numerati da 1 a N).
- L'intero M rappresenta il numero di strade (bidirezionali) che collegano i punti di interesse.
- L'intero K rappresenta il numero di supermercati presenti.
- L'array `supermercati`, indicizzato da 0 a $K - 1$, contiene i numeri dei punti di interesse che corrispondono ai supermercati della città.
- Gli array `da` e `a`, indicizzati da 0 a $M - 1$, contengono gli estremi delle strade, cioè i numeri dei punti di interesse che queste strade collegano.
- La funzione dovrà restituire il minimo numero di minuti necessari per raggiungere un supermercato e poi andare a casa della nonna, che verrà stampato sul file di output.

Dati di input

Il file `input.txt` è composto da $M + 2$ righe. La prima riga contiene gli interi N, M, K . La seconda riga contiene i K interi `supermercati`[i] separati da uno spazio. Le successive M righe contengono le descrizioni delle strade: la i -esima di queste righe contiene i due interi `da`[i] e `a`[i].

Dati di output

Il file `output.txt` è composto da un'unica riga contenente un unico intero, la risposta a questo problema.

Assunzioni

- $3 \leq N \leq 10\,000$.
- $2 \leq M \leq 100\,000$.
- $1 \leq K \leq N - 2$.
- $1 < \text{supermercati}_i < N$ per ogni $i = 0 \dots K - 1$.
- Nei punti 1 ed N non sono presenti supermercati, e i valori `supermercati`[i] non sono ripetuti.
- Tutte le strade sono bidirezionali e non sono duplicate nell'input.
- È possibile da ogni punto raggiungere ogni altro punto della città.
- Gabriele può percorrere la stessa strada più di una volta, o passare più volte da uno stesso punto.
- Gabriele impiega 1 minuto a percorrere ogni strada.
- Il tempo speso all'interno del supermercato è trascurabile.

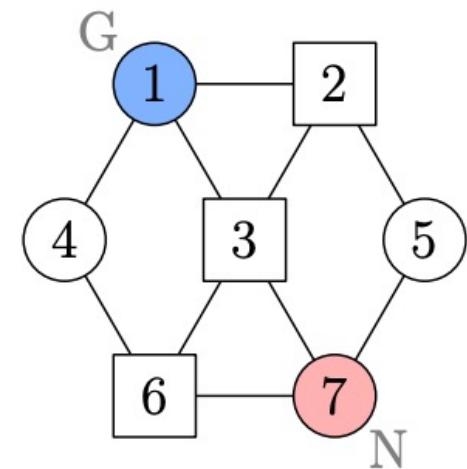
Assegnazione del punteggio

Il tuo programma verrà testato su diversi test case raggruppati in subtask. Per ottenere il punteggio relativo ad un subtask, è necessario risolvere correttamente tutti i test relativi ad esso.

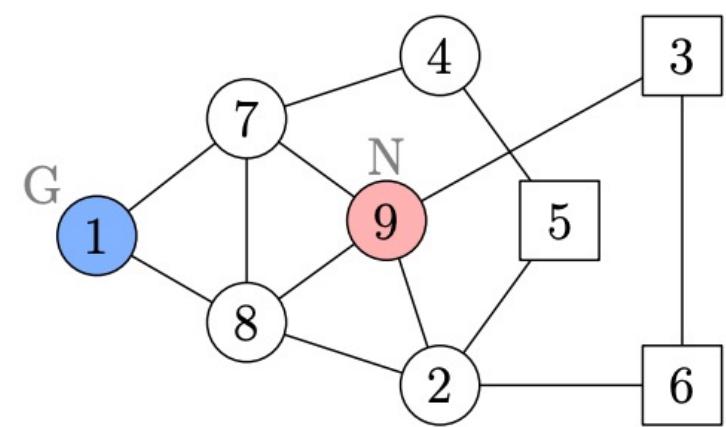
- **Subtask 1 [10 punti]**: Casi d'esempio.
- **Subtask 2 [20 punti]**: $N, K \leq 100$.
- **Subtask 3 [40 punti]**: $K \leq 10$.
- **Subtask 4 [30 punti]**: Nessuna limitazione specifica.

Esempi di input/output

input.txt	output.txt
7 10 3 3 6 2 1 2 2 5 7 3 4 6 3 6 6 7 7 5 1 4 3 1 2 3	2



input.txt	output.txt
9 13 3 3 6 5 2 5 9 2 7 1 6 3 9 3 1 8 2 6 6 5 7 9 2 8 4 7 4 5 8 9	4



Depurazione dell'acqua

Selezioni territoriali 2009



Depurazione dell'acqua (depura)

Difficoltà D = 2.

Descrizione del problema

Bisogna realizzare un procedimento chimico per la depurazione dell'acqua, avendo a disposizione un certo numero di sostanze, numerate da 1 in avanti. Per un'efficace depurazione, è necessario inserire nell'acqua la sostanza chimica purificante numero 1, tenendo presente che nell'acqua sono già presenti K sostanze chimiche.

Per quanto riguarda il procedimento adottato, valgono R precise regole per poter inserire le sostanze chimiche nell'acqua. Tali regole prevedono che una certa sostanza A possa essere inserita solo se nell'acqua sono già presenti un dato insieme di sostanze, ad esempio, A_1, A_2, \dots, A_n (dove $A_i \neq A$ per $1 \leq i \leq n$). In tal caso, scriviamo tale regola di inserimento nel seguente modo

$A :- A_1, A_2, \dots, A_n$

e diciamo che A compare nella parte sinistra della regola. Al fine di un corretto inserimento delle sostanze, valgono le seguenti osservazioni:

- l'eventuale presenza di ulteriori sostanze non inibisce l'applicabilità della regola suddetta;
- se A compare nella parte sinistra di una regola, allora non può comparire nella parte sinistra di altre regole e non può essere una delle K sostanze già presenti nell'acqua;
- qualora una sostanza sia priva di regole (ossia non compaia mai nella parte sinistra di una qualche regola) e non sia già presente nell'acqua, tale sostanza non può essere inserita;
- non è necessario usare tutte le regole e/o tutte le sostanze a disposizione.

Per esempio, ipotizzando che le sostanze 2 e 3 siano già presenti nell'acqua ($K=2$) e che valgano le seguenti regole ($R=4$):

4 :- 2

5 :- 2, 3

7 :- 2, 4

1 :- 3, 7, 4

possiamo inserire la sostanza 4 perché la sostanza 2 è già presente (prima regola); in seguito, possiamo inserire anche la sostanza 7 perché le sostanze 2 e 4 sono presenti nell'acqua (terza regola); a questo punto, possiamo aggiungere la sostanza 1 perché le sostanze 3, 7 e 4 sono presenti (ultima regola). Quindi abbiamo inserito un totale di $S=3$ sostanze, ossia 4, 7 e 1 (oltre alle $K=2$ già presenti), per purificare l'acqua.

Scrivere un programma che calcoli il numero minimo S di sostanze da inserire per purificare l'acqua, conoscendo le K sostanze già presenti nell'acqua e le R regole di inserimento. Tale numero sarà $S = 0$ se la sostanza 1 è già presente nell'acqua; sarà $S = 1$ se la sostanza 1 può essere inserita direttamente e non è già presente; in generale, sarà $S = m$ se è necessario inserire $m-1$ sostanze prima di poter inserire la sostanza 1. Nel caso in cui non sia possibile purificare l'acqua, bisogna restituire il valore $S = -1$.

Dati di input

Il file `input.txt` è composto da $K+R+1$ righe.

La prima riga contiene due interi positivi separati da uno spazio, rispettivamente il numero K delle sostanze chimiche già presenti nell'acqua e il numero R di regole di inserimento.

Le successive K righe contengono le K sostanze già presenti nell'acqua, dove ogni riga è composta da un solo intero positivo che rappresenta una di tali sostanze.

Le ultime R righe rappresentano le R regole, al massimo una regola per ciascuna sostanza non presente nell'acqua. Ciascuna riga è composta da $n+2$ interi positivi $A, n, A_1, A_2, \dots, A_n$ separati da uno spazio (dove $A_i \neq A$ per $1 \leq i \leq n$), i quali rappresentano la regola $A \rightarrow A_1, A_2, \dots, A_n$.

Dati di output

Il file `output.txt` è composto da una sola riga contenente un intero S , il minimo numero di sostanze inserite (oltre alle K già presenti) per purificare l'acqua secondo le regole descritte sopra.

Assunzioni

- $1 \leq K, R \leq 1000$
- Il numero di sostanze chimiche a disposizione è al massimo 2000.
- I casi di prova non contengono mai situazioni cicliche: in tal modo, non accade mai che una sostanza A possa essere inserita solo se A stessa è già presente nell'acqua.

Esempi di input/output

File input.txt

```
2 4
2
3
4 1 2
5 2 2 3
7 2 2 4
1 3 3 7 4
```

File output.txt

```
3
```

File input.txt

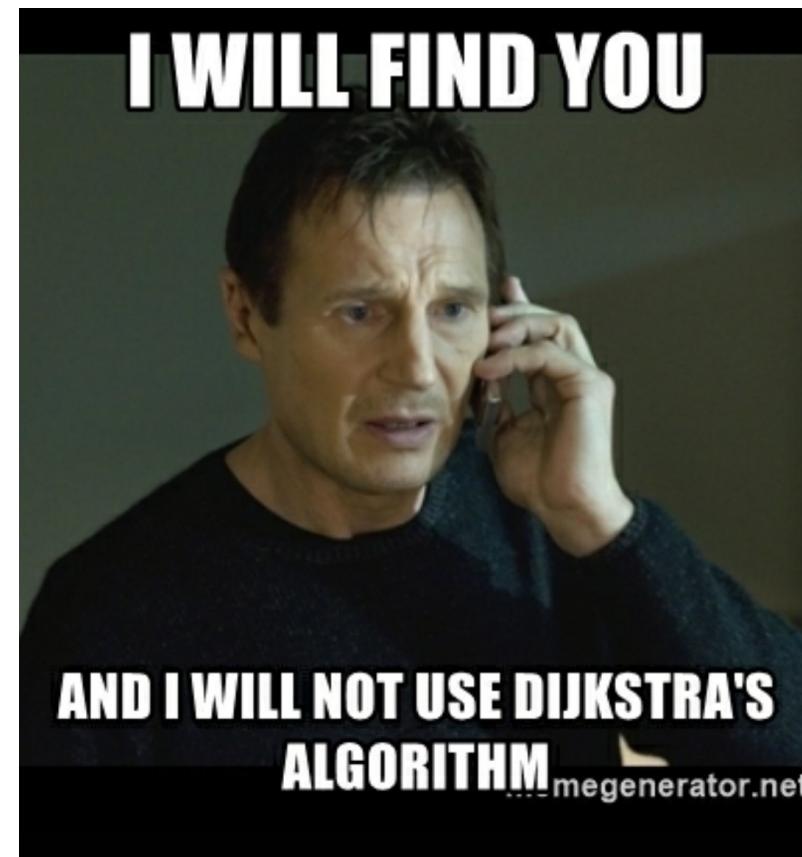
```
4 2
6
2
8
3
5 2 2 6
1 2 3 6
```

File output.txt

```
1
```

File input.txt	File output.txt
<pre>2 3 1 3 4 1 2 5 1 3 6 2 2 4</pre>	0
File input.txt	File output.txt
<pre>3 4 2 4 8 5 2 2 4 7 2 4 3 6 2 5 7 1 3 5 2 6</pre>	-1

Cammini
minimi

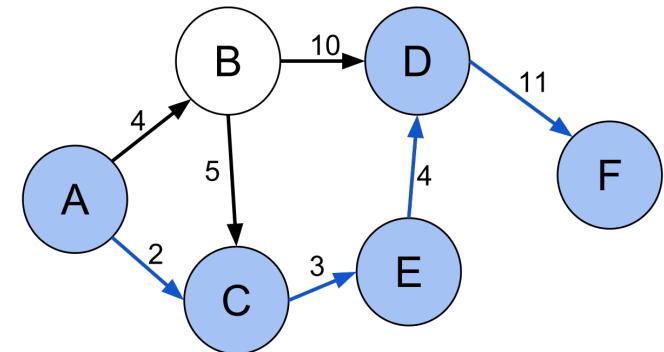


Cammini e costi

$G=(V,E)$ grafo orientato con costi w sugli archi

Costo di un cammino =
somma dei costi degli archi del
cammino

Cammino minimo tra una coppia di vertici x e y =
cammino di costo minore o uguale a quello di ogni altro
cammino tra x e y



Due proprietà dei cammini minimi

- **Sottostruttura ottima:** ogni sottocammino di un cammino minimo è anch'esso minimo
- (In quale tecnica è utile la sottostruttura ottima?)
- **Cosa accade se c'è un ciclo negativo?**
Se due vertici x e y appartengono a un ciclo di costo negativo, non esiste nessun cammino minimo finito tra di essi!

Distanza fra vertici

- La distanza d_{xy} tra due vertici x e y è il **costo di un cammino minimo tra da x a y** , o $+\infty$ se i due vertici non sono connessi
- Disuguaglianza triangolare: per ogni x , y e z

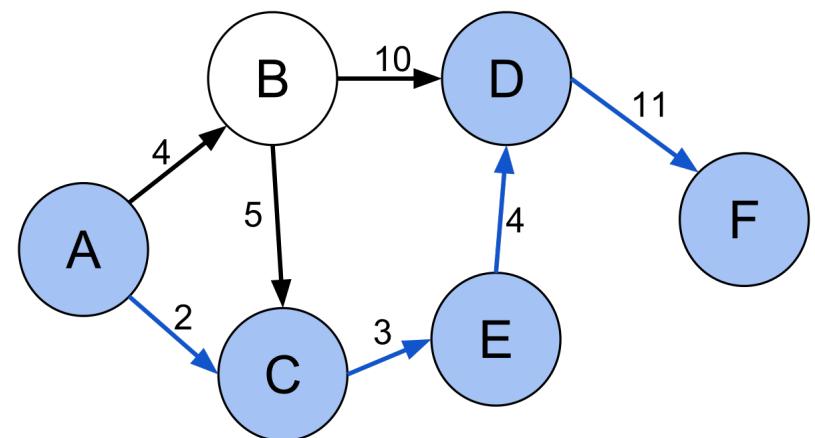
$$d_{xz} \leq d_{xy} + d_{yz}$$

- **Condizione di Bellman**: per ogni arco (u,v) e per ogni vertice s

$$d_{su} + w(u, v) \geq d_{sv}$$

Albero dei cammini minimi

- Un arco (u,v) appartiene a un cammino minimo a partire da un vertice s se e solo se u è raggiungibile da s e $d_{su} + w(u,v) = d_{sv}$
- 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**



Tecnica del rilassamento

- Partendo da **stime per eccesso** delle distanze $D_{xy} \geq d_{xy}$, aggiornare le stime, decrementandole progressivamente fino a renderle esatte.
- Aggiornamento delle stime basato sul seguente **passo di rilassamento**:

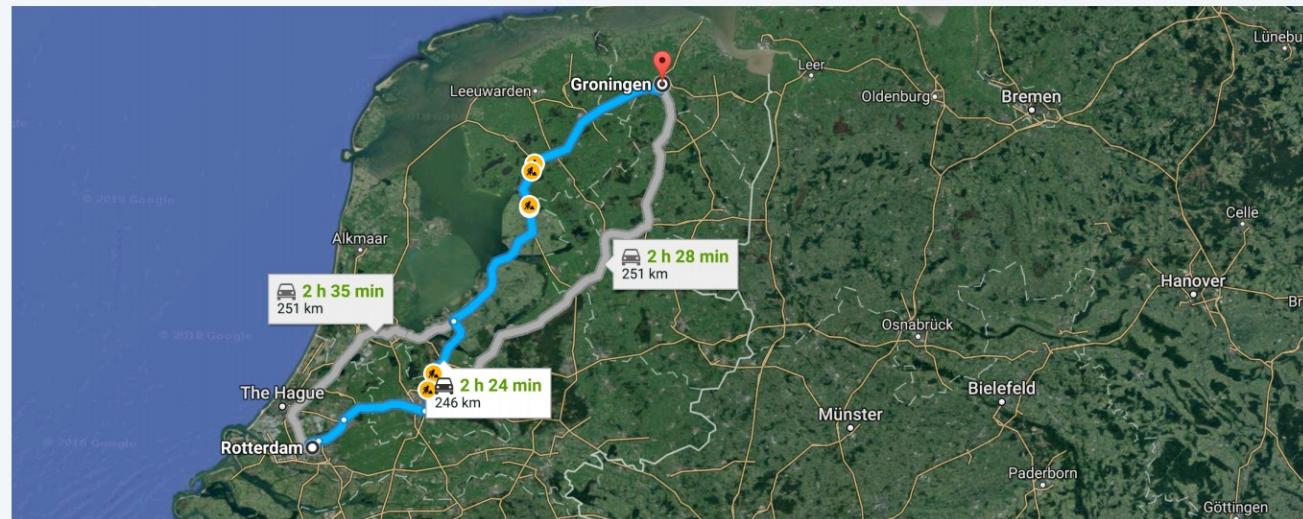
(RILASSAMENTO) **if** $(D_{xv} + w(\pi_{vy}) < D_{xy})$
 then $D_{xy} \leftarrow D_{xv} + w(\pi_{vy})$

- Negli algoritmi che vedremo, π_{vy} è un singolo arco (v,y)

Algoritmo di Dijkstra

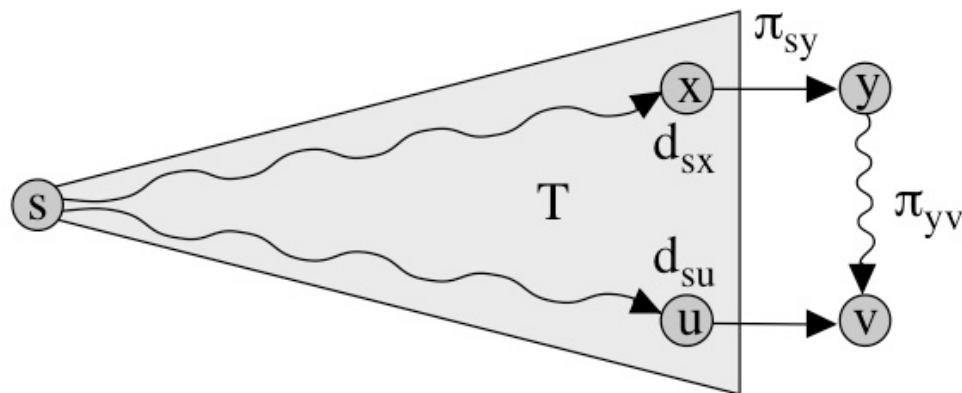
(per cammini minimi a sorgente singola in grafi con pesi non negativi)

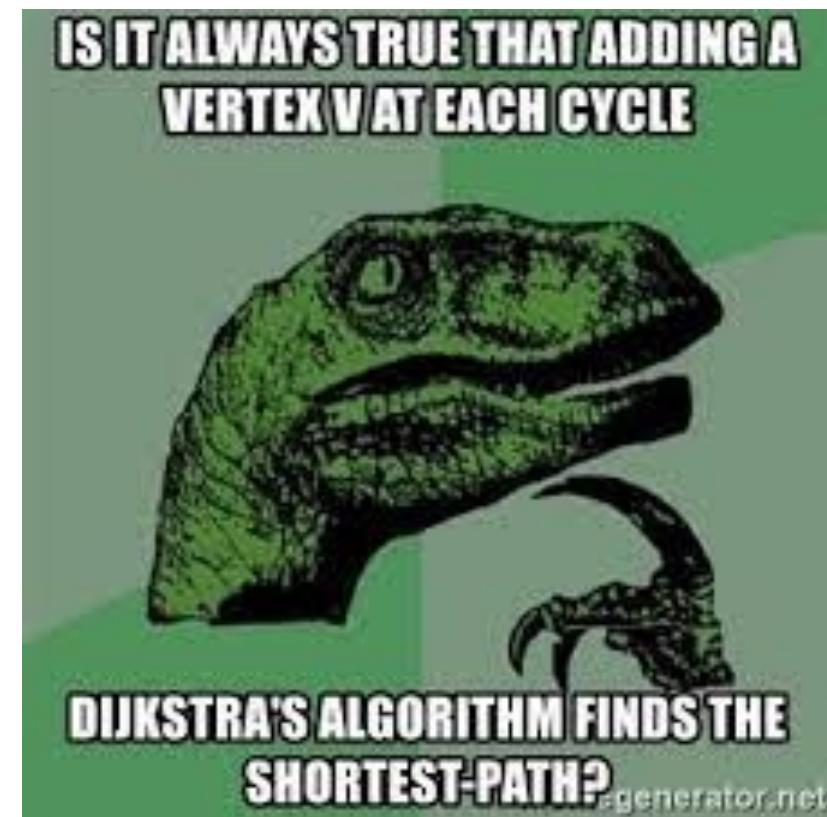
“What’s the shortest way to travel from Rotterdam to Groningen? It is the algorithm for the shortest path, which I designed in about 20 minutes. One morning I was shopping in Amsterdam with my young fiancée, and tired, we sat down on the café terrace to drink a cup of coffee and I was just thinking about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



Estensione dei cammini minimi

Se T è un albero dei cammini minimi radicato in s che non include tutti i vertici raggiungibili da s , l'arco (u,v) tale che $u \in T$ e $v \notin T$ che minimizza la quantità $d_{su} + w(u,v)$ appartiene a un cammino minimo da s a v





Non ci sono archi di costo negativo!
Se scegliamo il minimo non potremo mai migliorarlo!

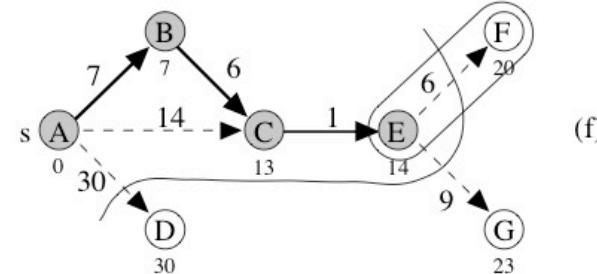
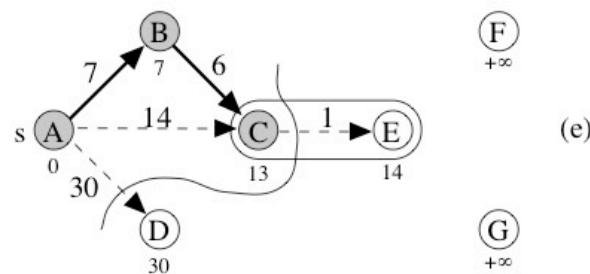
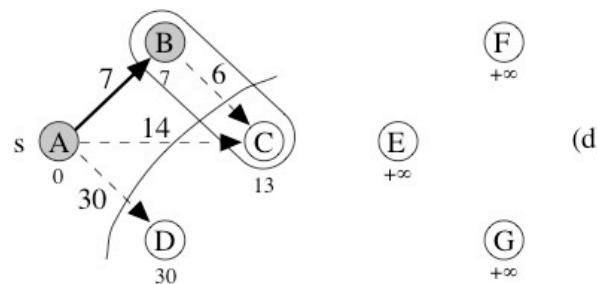
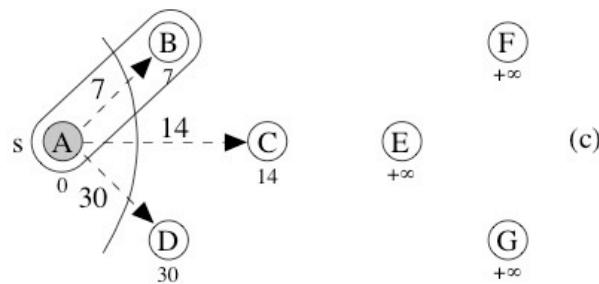
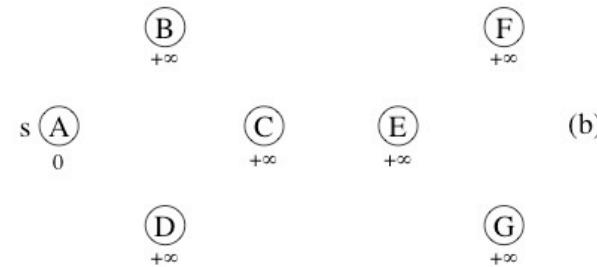
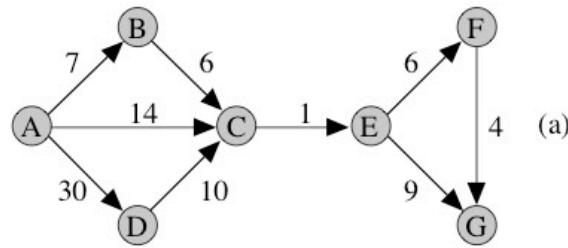
Approccio di Dijkstra

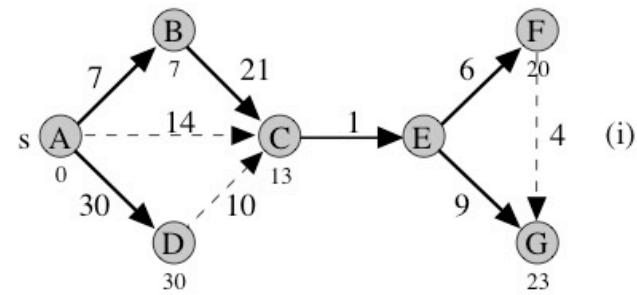
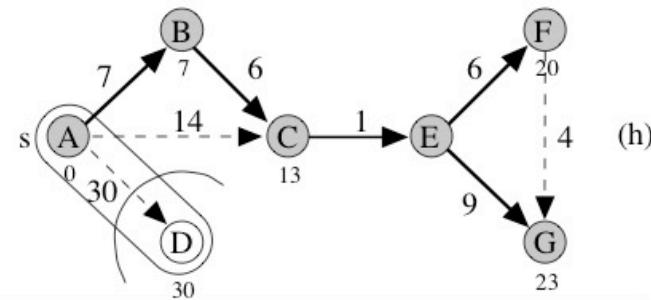
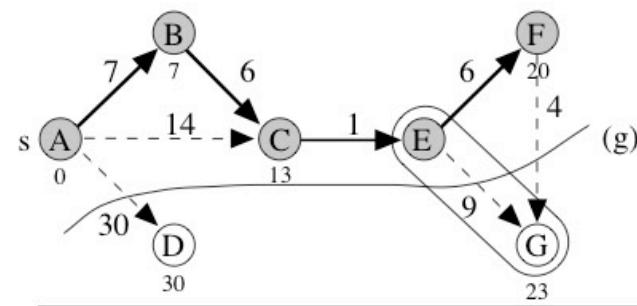
Scegli un arco (u,v) con $u \in T$ e $v \notin T$ che minimizza la quantità $D_{su} + w(u,v)$, effettua il passo di rilassamento $D_{sv} \leftarrow D_{su} + w(u,v)$, ed aggiungilo a T

Archi incidenti a T mantenuti in una **coda con priorità**:
priorità = costo stimato D_{xy} del cammino minimo

Pseudocodice

```
algoritmo Dijkstra(grafo G, vertice s) → albero
  for each ( vertice  $u$  in  $G$  ) do  $D_{su} \leftarrow +\infty$ 
     $\hat{T} \leftarrow$  albero formato dal solo nodo  $s$ 
    CodaPriorita S
     $D_{ss} \leftarrow 0$ 
    S.insert( $s, 0$ )
    while ( not S.isEmpty() ) do
       $u \leftarrow$  S.deleteMin()
      for each ( arco  $(u, v)$  in  $G$  ) do
        if ( $D_{sv} = +\infty$ ) then
          S.insert( $v, D_{su} + w(u, v)$ )
           $D_{sv} \leftarrow D_{su} + w(u, v)$ 
          rendi  $u$  padre di  $v$  in  $\hat{T}$ 
        else if ( $D_{su} + w(u, v) < D_{sv}$ ) then
          S.decreaseKey( $v, D(u) + w(u, v)$ )
           $D_{sv} \leftarrow D_{su} + w(u, v)$ 
          rendi  $u$  nuovo padre di  $v$  in  $\hat{T}$ 
    return  $\hat{T}$ 
```

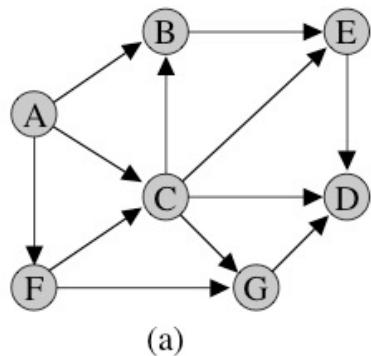




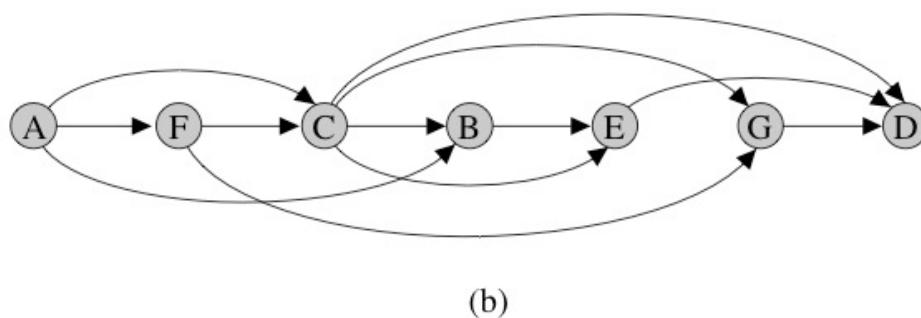
Algoritmo per grafi diretti aciclici (anche in grafi con pesi negativi)

Ricordiamo: DAG e ordinamento topologico

- Se esiste un arco da A a B , allora A comparirà nell'ordinamento prima di B
- L'ordinamento topologico di un DAG esiste sempre



(a) Esempio di grafo aciclico; (b) Ordinamento topologico del grafo (a).



Cammini minimi in grafi aciclici?

Basta eseguire i rilassamenti in ordine topologico!

```
algoritmo distanzeAciclico(grafo  $G$ , vertice  $s$ )  $\rightarrow$  distanze
    inizializza  $D$  tale che  $D_{sv} = +\infty$  per  $v \neq s$ , e  $D_{ss} = 0$ 
     $ord \leftarrow$  ordinamentoTopologico( $G$ )
    for  $i = 1$  to  $n$  do
        sia  $u$  l' $i$ -esimo vertice nell'ordinamento topologico  $ord$ 
        for each ( $(u, v) \in E$ ) do
            if ( $D_{su} + w(u, v) < D_{sv}$ ) then  $D_{sv} \leftarrow D_{su} + w(u, v)$ 
    return  $D$ 
```

Il tesoro del pirata Barbablù

Selezioni territoriali 2012



Il tesoro del Pirata Barbablù (barbablu)

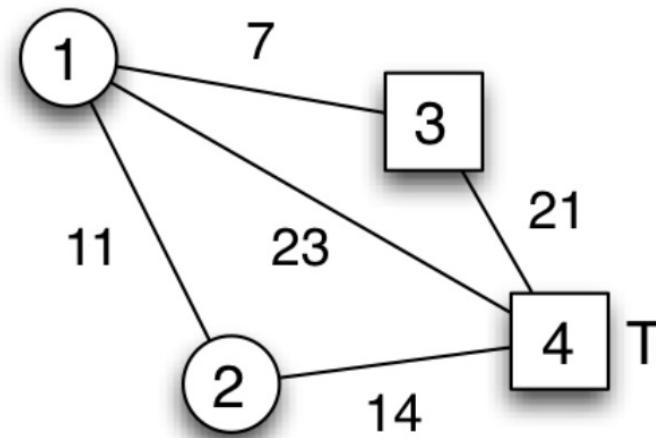
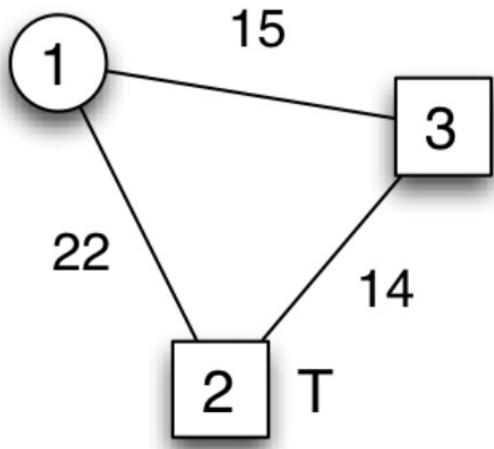
Difficoltà D = 2.

Descrizione del problema

John Steam della compagnia "Oriental Steam Navigation" decide di organizzare una spedizione di recupero del tesoro del Pirata Barbablù, custodito nel relitto del galeone del pirata, affondato al largo di Gobal, che si trova adagiato su un fianco a 30 metri di profondità. L'unico punto di accesso al relitto è uno squarcio sulla fiancata, in corrispondenza della cabina numero 1. Nel galeone sono presenti cabine e corridoi che le collegano. Tutti i corridoi sono totalmente sommersi dall'acqua a causa della rottura degli oblo mentre in alcune delle cabine sono rimaste delle sacche d'aria. A causa degli spazi angusti non è possibile, per i sommozzatori, esplorare la nave con le bombole d'aria; sono quindi costretti a nuotare in apnea, sfruttando le sacche d'aria presenti nel tragitto per respirare.

Prima di procedere con le operazioni di recupero ti viene commissionata la realizzazione di un programma in grado di individuare il percorso più breve all'interno del galeone che permetta ai sommozzatori di raggiungere la cabina con il tesoro a partire dall'apertura. In alcune cabine sono presenti sacche d'aria che possono essere usate per respirare. Un sommozzatore riesce a nuotare senza aria per 20 metri al massimo prima di dover riprendere fiato. In Figura 1 sono mostrati due possibili scenari. La cabina di ingresso è come detto la numero 1, mentre la cabina del tesoro (rappresentato da una T) è la numero 2 per l'esempio di sinistra, e la numero 4 per l'esempio di destra. Le cabine con la sacca d'aria sono quadrate, mentre quelle senza sacca sono tonde. A fianco di ogni corridoio è segnata la sua lunghezza in metri. L'esempio di sinistra ammette una sola soluzione, di lunghezza 29 metri, mentre quello di destra non ha soluzioni.

Figure 1:



Le cabine della nave sono numerate da 1 ad N e sono collegate tra loro da M corridoi. L'apertura è la numero 1 mentre il tesoro si trova nella cabina numero C (con $1 \leq C \leq N$). Di ogni cabina si conosce l'eventuale presenza di aria e di ogni corridoio la lunghezza in metri.

Il tuo compito è quello di trovare la lunghezza in metri del percorso più breve che permette ad un sommozzatore di partire dalla cabina con l'apertura e di raggiungere il tesoro, in apnea, sfruttando le eventuali sacche d'aria trovate nel percorso. La cabina del tesoro ha sempre una sacca d'aria, che consente al sommozzatore di recuperare il tesoro.

Dati di input

Il file `input.txt` è composto da $M+2$ righe. La prima riga contiene quattro interi positivi separati da uno spazio, che rappresentano il numero N delle cabine, il numero M dei corridoi, il numero C che rappresenta la cabina del tesoro e il numero K che rappresenta quante cabine hanno sacche d'aria al loro interno. La seconda riga contiene K numeri separati da uno spazio che rappresentano i numeri (distinti) delle cabine che contengono aria. Ciascuna delle rimanenti M righe contiene tre interi I, J, L separati da uno spazio che indicano la presenza di un corridoio che collega le cabine I e J di lunghezza L (in metri).

Dati di output

Il file `output.txt` è composto da una sola riga contenente la lunghezza in metri del percorso più breve che permetta, a partire dall'apertura, di raggiungere la cabina del tesoro in apnea. Riportare -1 se non esiste nessun percorso che soddisfa i vincoli.

Assunzioni

- $2 \leq N \leq 30$
- $2 \leq M \leq 100$
- $1 \leq C \leq N$
- $0 \leq K \leq N$

Esempi di input/output

File input.txt

```
3 3 2 2
2 3
1 2 22
1 3 15
2 3 14
```

File output.txt

```
29
```

File input.txt

```
4 5 4 2
3 4
1 2 11
1 3 7
1 4 23
2 4 14
3 4 21
```

File output.txt

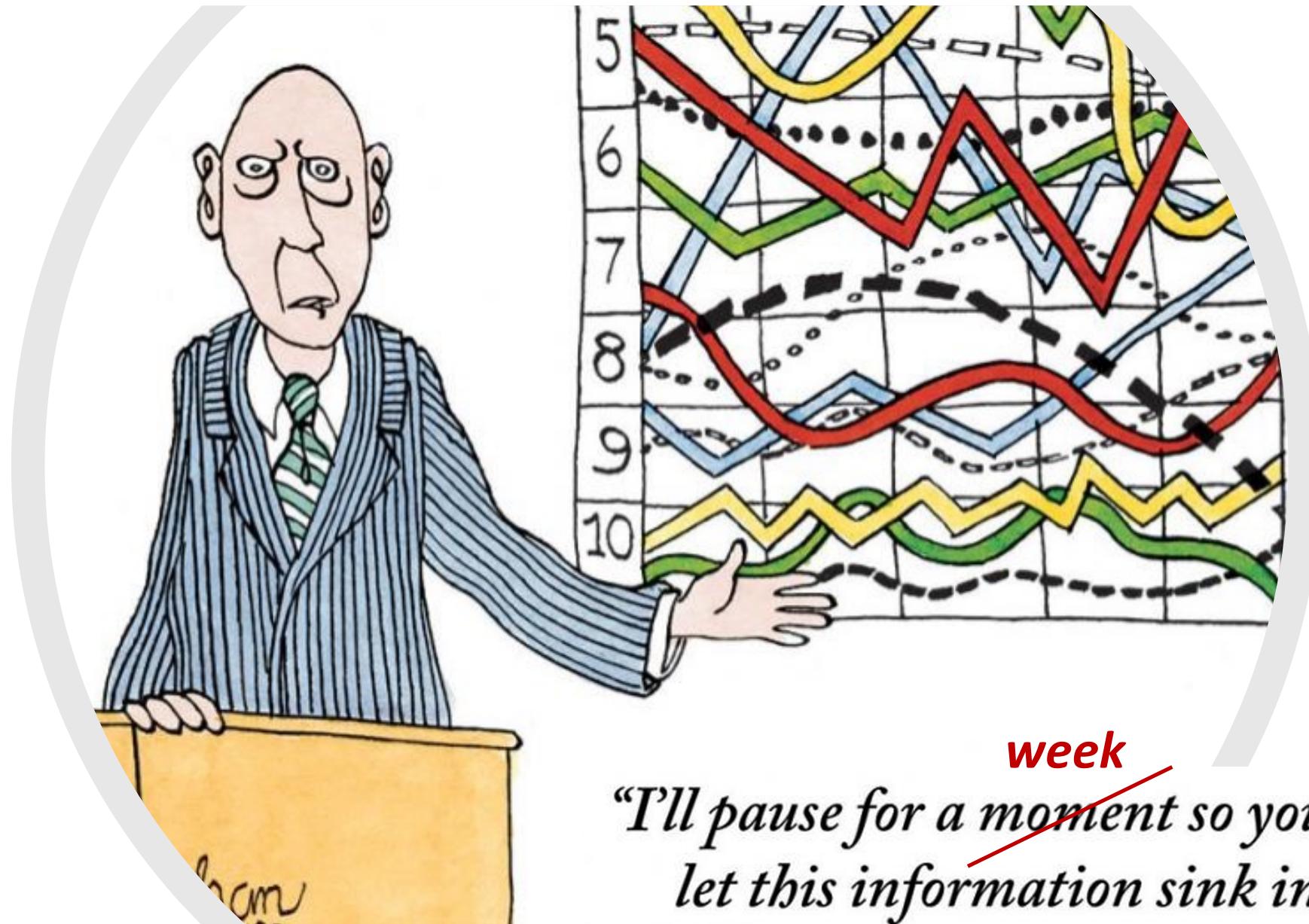
```
-1
```

Esercizi per casa



Esercizi

- Alberi (OII 2003)
<https://training.olinfo.it/#/task/alberi/statement>
- Flood forecasting (OIS 2020)
https://training.olinfo.it/#/task/ois_rainstorm/statement
- Paper (ABC 2017 – Algorithm Bergamo Contest)
https://training.olinfo.it/#/task/abc_paper/statement
- Islands (IOI 2008)
<https://training.olinfo.it/#/task/islands/statement>



*"I'll pause for a moment so you can
let this information sink in."*

week