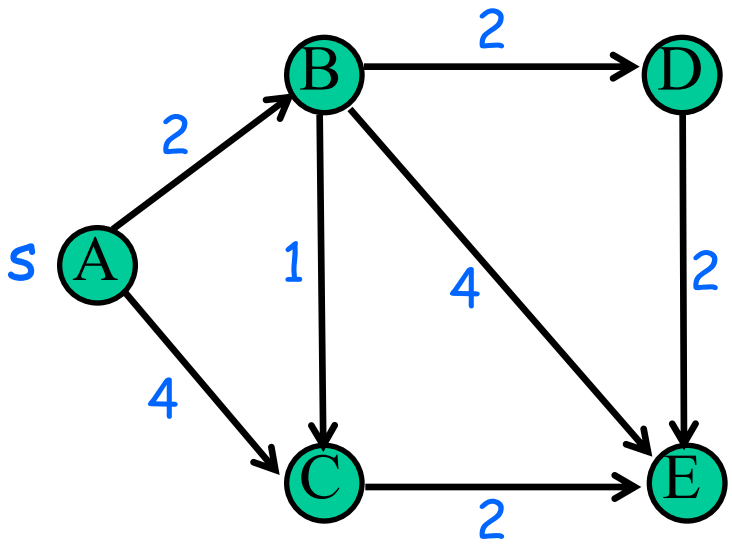
Cammini minimi in grafi:
una trilogia

Cammini minimi in grafi:
Episodio III: la fine della trilogia

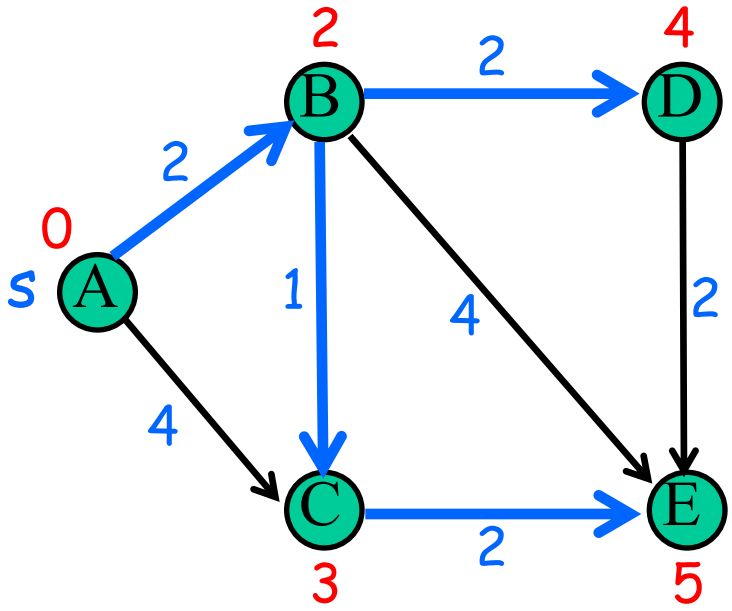
…nelle puntate precedenti

- Input:
  - grafo pesato $G=(V,E,w)$, $s \in V$
- Output:
  - albero dei cammini minimi radicato in s e/o le distanze di tutti i nodi da s, ovvero, $d_G(s,v)$ per ogni $v \in V$



…nelle puntate precedenti

- Input:
  - grafo pesato $G=(V,E,w)$, $s \in V$
- Output:
  - albero dei cammini minimi radicato in s e/o le distanze di tutti i nodi da s, ovvero, $d_G(s,v)$ per ogni $v \in V$



Nelle puntate precedenti:

- Visita BFS: albero dei cammini minimi per grafi non pesati. Tempo: $O(m+n)$.
- Algoritmo di Dijkstra: grafi con pesi non negativi. Tempo: $O(m+n \log n)$ con heap di Fibonacci. Altre implementazioni meno efficienti:
  - $O(n^2)$ - con array/liste non ordinate
  - $O(m \log n)$ con heap binari/binomiali
- Algoritmo di Bellman-Ford: cammini minimi per grafi con pesi qualsiasi (senza cicli negativi). Tempo $O(mn)$ – con liste di adiacenza.
- Algoritmo basato sull'ord. topologico: albero dei cammini minimi per DAG con pesi qualsiasi. Tempo $O(m+n)$.

Cammini minimi fra tutte le coppie:
Algoritmo di Floyd e Warshall e Algoritmo di Johnson

distanza fra tutte le coppie: soluzioni semplici

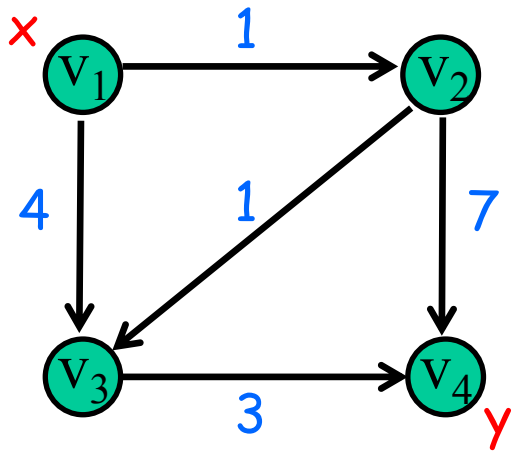
- grafi non pesati:
  - eseguo n volte visita BFS:
  - tempo $O(mn)$
- grafi con pesi non negativi:
  - eseguo n volte Dijkstra:
  - tempo: $O(mn+n^2 \log n)$
- grafi con pesi qualsiasi (senza cicli negativi)
  - eseguo n volte Bellman-Ford
  - tempo: $O(mn^2)$

Algoritmo di Floyd e Warshall

(cammini minimi fra tutte le coppie di nodi per grafi senza cicli negativi)

Approccio

- Elegante applicazione della tecnica della programmazione dinamica
- Numeriamo i vertici di *G* da 1 a n, cioè $V=\{v_1, v_2, \ldots v_n\}$.
- Un cammino minimo k-vincolato da x a y è un cammino di costo minimo tra tutti i cammini da x a y che usano come vertici intermedi un sottoinsieme qualsiasi (anche vuoto) dei vertici $I_k=\{v_1, v_2, \ldots v_k\}$.



tra x e y il cammino minimo
- 0-vincolato è lungo $+\infty$
- 1-vincolato è lungo $+\infty$
- 2-vincolato è lungo 8: $\langle x,v_2,y\rangle$
- 3-vincolato è lungo 5: $\langle x,v_2,v_3,y\rangle$
- 4-vincolato (ovvero senza vincoli) è lungo 5: $\langle x,v_2,v_3,y\rangle$

- Idea di Floyd e Warshall (sottoproblemi): calcolare cammini minimi k-vincolati per k=0,1,…, n

Relazioni tra distanze vincolate

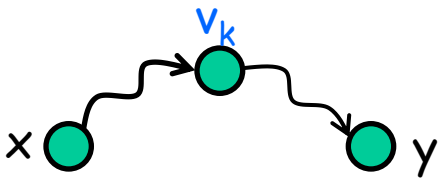
- Sia d_{xy}^{k} il costo di un cammino minimo k-vincolato da x a y. Valgono le seguenti proprietà:
  - $d_{xy}^{0} = w(x,y)$ se $(x,y) \in E$, $+\infty$ altrimenti
  - $d_{xv_k}^{k-1} = d_{xv_k}^{k}$ e $d_{v_kx}^{k-1} = d_{v_kx}^{k}$
  - $d_{xy}^{n} = d_{xy}$

- Per le proprietà di cui sopra e per la proprietà di minimalità dei sottocammini di cammini minimi, si ha:

$$d_{xy}^{k} = \min\{ d_{xy}^{k-1} \; , \; d_{xv_k}^{k} + d_{v_ky}^{k} \} = \min\{ d_{xy}^{k-1} \; , \; d_{xv_k}^{k-1} + d_{v_ky}^{k-1} \}$$

cammino minimo k-vincolato non passa per v_k

passa per v_k



$\Rightarrow$L'algoritmo calcola d_{xy}^{k} incrementando k da 0 a n

Pseudocodice

algoritmo `FloydWarshall`$(grafo\ G) \rightarrow distanze$
  inizializza D tale che $D_{xy} = w(x,y)$ se $(x,y) \in E$, e $D_{xy} = +\infty$ altrimenti
  for each $v \in V$ **do**
    for each $((x,y) \in V \times V)$ **do**
      if $(D_{xv} + D_{vy} < D_{xy})$ **then** $D_{xy} \leftarrow D_{xv} + D_{vy}$
  return D

Tempo di esecuzione: $\Theta(n^3)$

(sia con liste di adiacenza che con matrice di adiacenza)

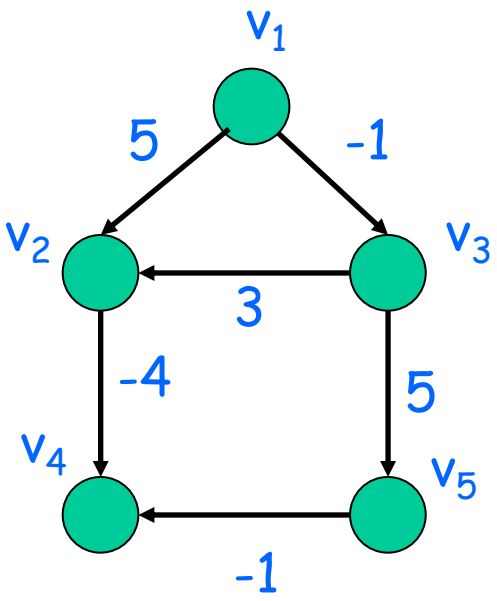
D: Meglio di B&F ripetuto n volte?
R: Sì! $O(n^3)=O(mn^2)$

D: Meglio dell'applicazione ripetuta di Dijkstra?
R: Utilizzando gli Heap di Fibonacci, n applicazioni dell'algoritmo di Dijkstra richiedono tempo $O(mn+n^2 \log n) = O(n^3)$. Quindi, Dijkstra è più efficiente. Però si applica solo su un sottoinsieme delle istanze ammissibili per F&W.

e se G contiene un ciclo di peso negativo posso accorgermene?

Esercizio: pensarci…

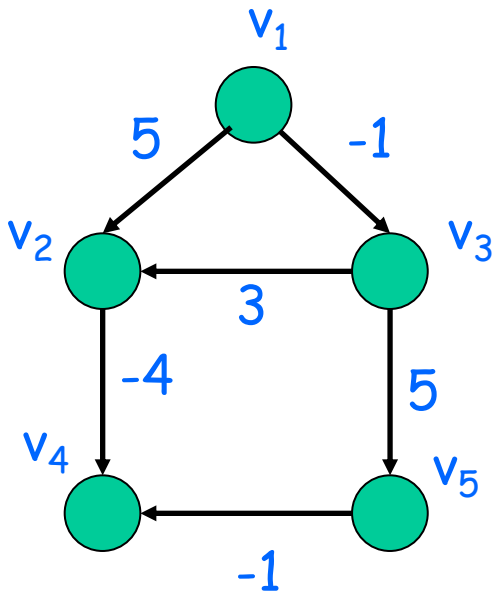
Esempio: applicare l'algoritmo di Floyd e Warshall al seguente grafo:



Posso applicare F&W?

Sì, non ci sono cicli negativi!

Inizializziamo la matrice delle distanze:



$$D_0 = \begin{bmatrix} 0 & 5 & -1 & +\infty & +\infty \\ +\infty & 0 & +\infty & -4 & +\infty \\ +\infty & 3 & 0 & +\infty & 5 \\ +\infty & +\infty & +\infty & 0 & +\infty \\ +\infty & +\infty & +\infty & -1 & 0 \end{bmatrix} \quad D_1 = \begin{bmatrix} 0 & 5 & -1 & +\infty & +\infty \\ +\infty & 0 & +\infty & -4 & +\infty \\ +\infty & 3 & 0 & +\infty & 5 \\ +\infty & +\infty & +\infty & 0 & +\infty \\ +\infty & +\infty & +\infty & -1 & 0 \end{bmatrix}$$

$$D_2 = \begin{bmatrix} 0 & 5 & -1 & \mathbf{1} & +\infty \\ +\infty & 0 & +\infty & -4 & +\infty \\ +\infty & 3 & 0 & \mathbf{-1} & 5 \\ +\infty & +\infty & +\infty & 0 & +\infty \\ +\infty & +\infty & +\infty & -1 & 0 \end{bmatrix} \quad D_3 = \begin{bmatrix} 0 & 2 & -1 & -2 & 4 \\ +\infty & 0 & +\infty & -4 & +\infty \\ +\infty & 3 & 0 & -1 & 5 \\ +\infty & +\infty & +\infty & 0 & +\infty \\ +\infty & +\infty & +\infty & -1 & 0 \end{bmatrix}$$

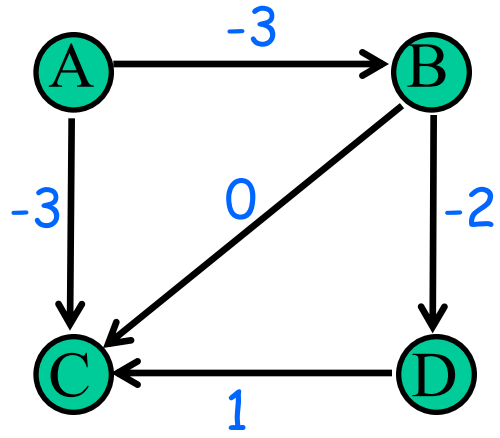
$$D_4 = \begin{bmatrix} 0 & 2 & -1 & -2 & 4 \\ +\infty & 0 & +\infty & -4 & +\infty \\ +\infty & 3 & 0 & -1 & 5 \\ +\infty & +\infty & +\infty & 0 & +\infty \\ +\infty & +\infty & +\infty & -1 & 0 \end{bmatrix} \quad D_5 = \begin{bmatrix} 0 & 2 & -1 & -2 & 4 \\ +\infty & 0 & +\infty & -4 & +\infty \\ +\infty & 3 & 0 & -1 & 5 \\ +\infty & +\infty & +\infty & 0 & +\infty \\ +\infty & +\infty & +\infty & -1 & 0 \end{bmatrix}$$

Algoritmo di Johnson

Idea: rendere i pesi non negativi e poi applicare
n volte l'algoritmo di Dijkstra

riferimento: capito 25 del libro
Introduzione agli algoritmi e strutture dati,
di Cormen, Leiserson, Rivest, Stein
McGraw-Hill

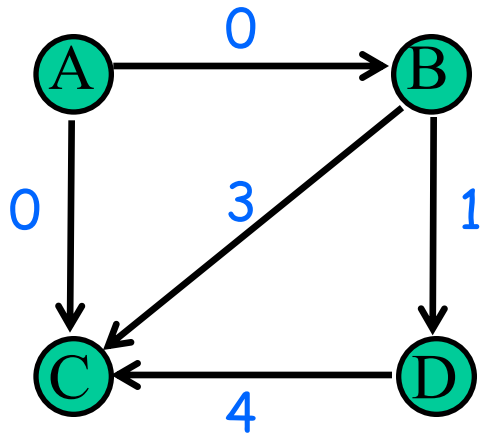
rendere i pesi non negativi: un primo tentativo



Idea: aggiungere un valore opportuno β al peso di tutti archi.

aggiungo β=3

funziona?

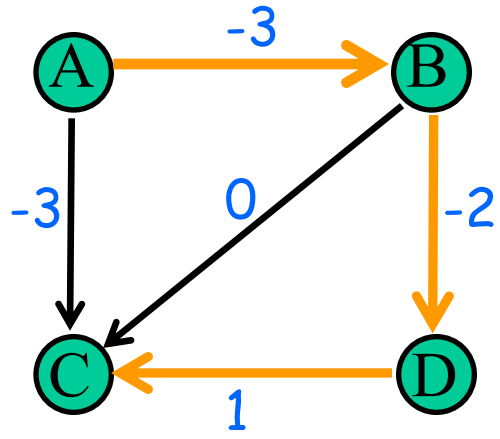


no: non preservo i cammini minimi!

cammini con un # diverso di archi cambiano costo in modo diverso

rendere i pesi non negativi: un primo tentativo

cammino minimo fra A e C

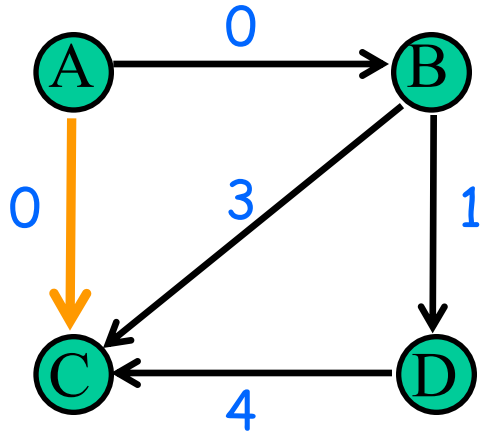


Idea: aggiungere un valore opportuno β al peso di tutti archi.

aggiungo β=3

funziona?

cammino minimo fra A e C



no: non preservo i cammini minimi!

cammini con un # diverso di archi cambiano costo in modo diverso

un modo migliore per ri-pesare gli archi: aggiungere pesi sui nodi

Dato un grafo pesato $G=(V,E,w)$ e una funzione $h:V \rightarrow R$

per ogni arco $(u,v) \in E$, definiamo un nuovo peso:

$$w_h(u,v)=w(u,v) + h(u)-h(v)$$

notazione:

$d(u,v)$: distranza da u a v in $G=(V,E,w)$

$d_h(u,v)$: distranza da u a v in $G=(V,E,w_h)$

Lemma

Un cammino $p=\langle v_1, v_2, \ldots, v_k \rangle$ è minimo in $G=(V,E,w)$ se e solo è minimo in $G=(V,E,w_h)$

dim

$$w_h(p) = \sum_{i=1}^{k-1} w_h(v_i, v_{i+1})$$

$$= \sum_{i=1}^{k-1} \left(w(v_i, v_{i+1}) + h(v_i) - h(v_{i+1}) \right)$$

$$= \sum_{i=1}^{k-1} w(v_i, v_{i+1}) + \sum_{i=1}^{k-1} \left(h(v_i) - h(v_{i+1}) \right)$$

$$= w(p) + h(v_1) - h(v_k)$$

$\blacksquare$

$\Rightarrow$ $d_h(u,v)=d(u,v)+h(u)-h(v)$

scegliere h() in modo da rendere i pesi non negativi

per ogni arco $(u,v) \in E$, vorremmo:

$$w_h(u,v) \geq 0$$

$$w(u,v) + h(u) - h(v) \geq 0$$

$$h(v) \leq h(u) + w(u,v)$$

ricorda qualcosa?

...la condizione di Bellman:

$$d(s,v) \leq d(s,u) + w(u,v)$$

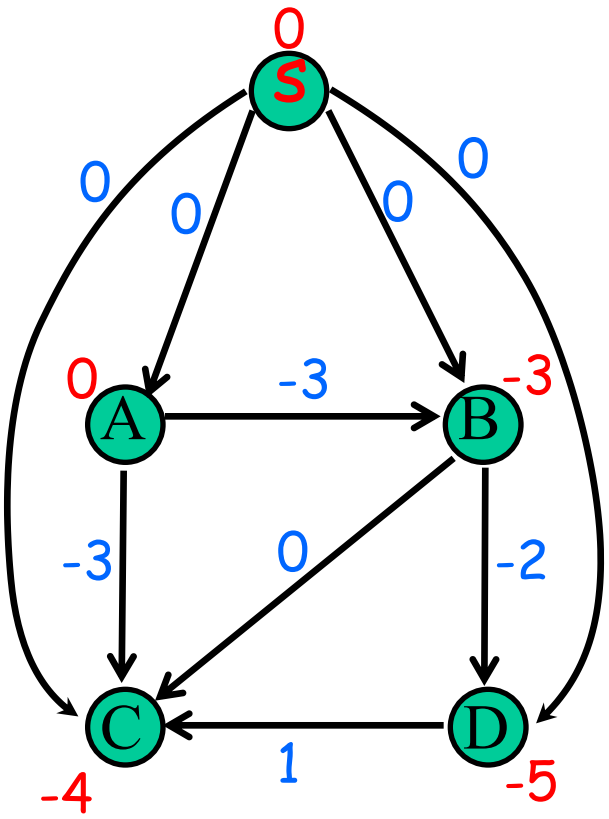
calcolare h()

assunzione: G non ha cicli di peso negativo

aggiungo una sorgente fittizia s

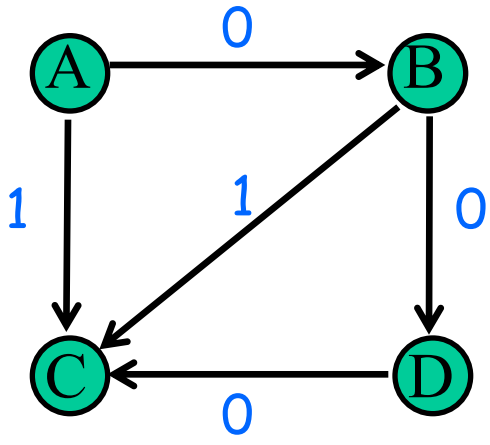
calcolo le distanze da s

e le uso come funzione h()



$G=(V,E,w)$

$G=(V,E,w_h)$



i pesi degli archi sono non negativi e i cammini minimi fra tutte le coppie sono preservati!

Nota: per ogni arco (u,v):

$d(s,v) \leq d(s,u) + w(u,v) \Rightarrow w_h(u,v) \geq 0$

Algoritmo di Johnson

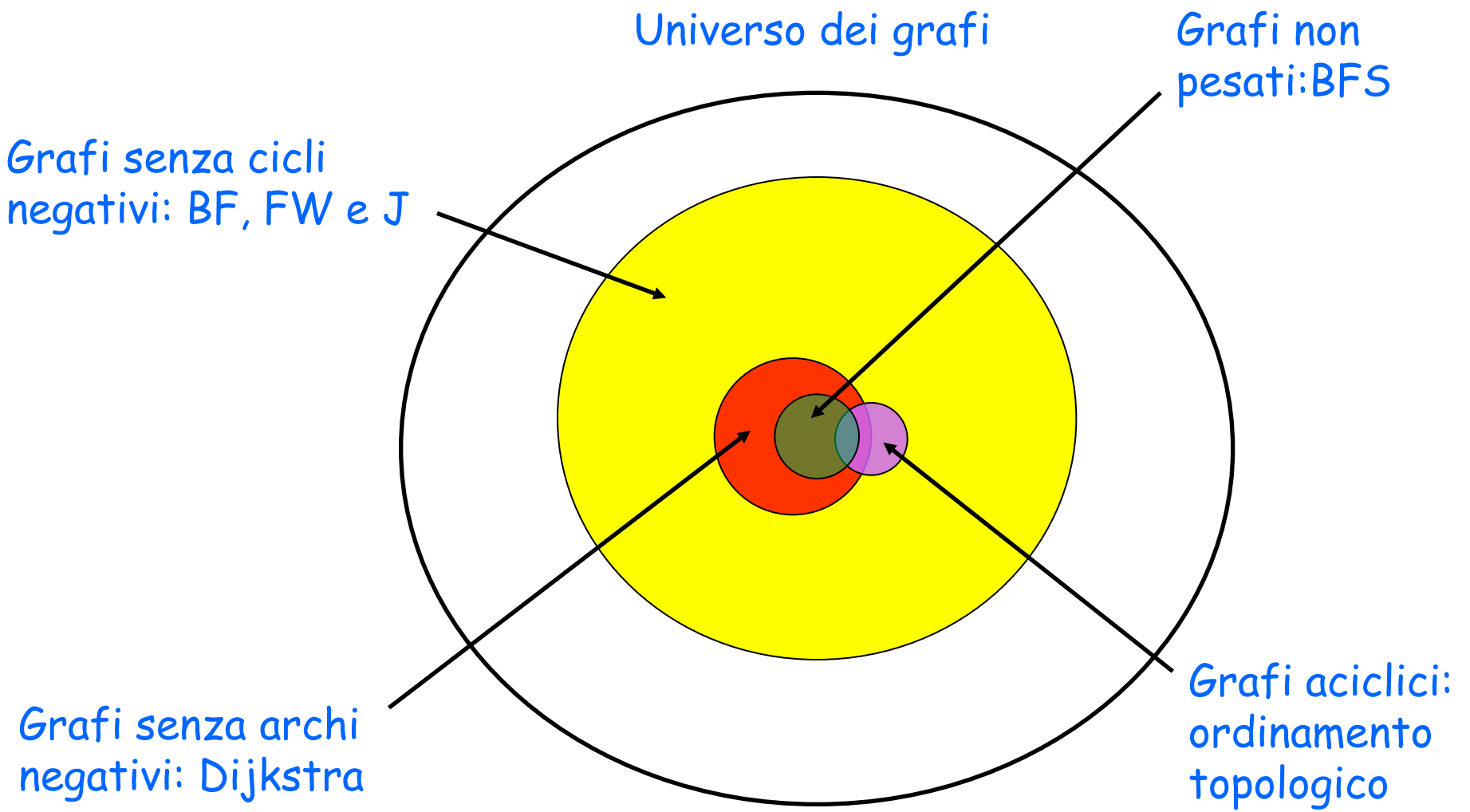
input: $G=(V,E,w)$ %assunzione: G non ha cicli negativi

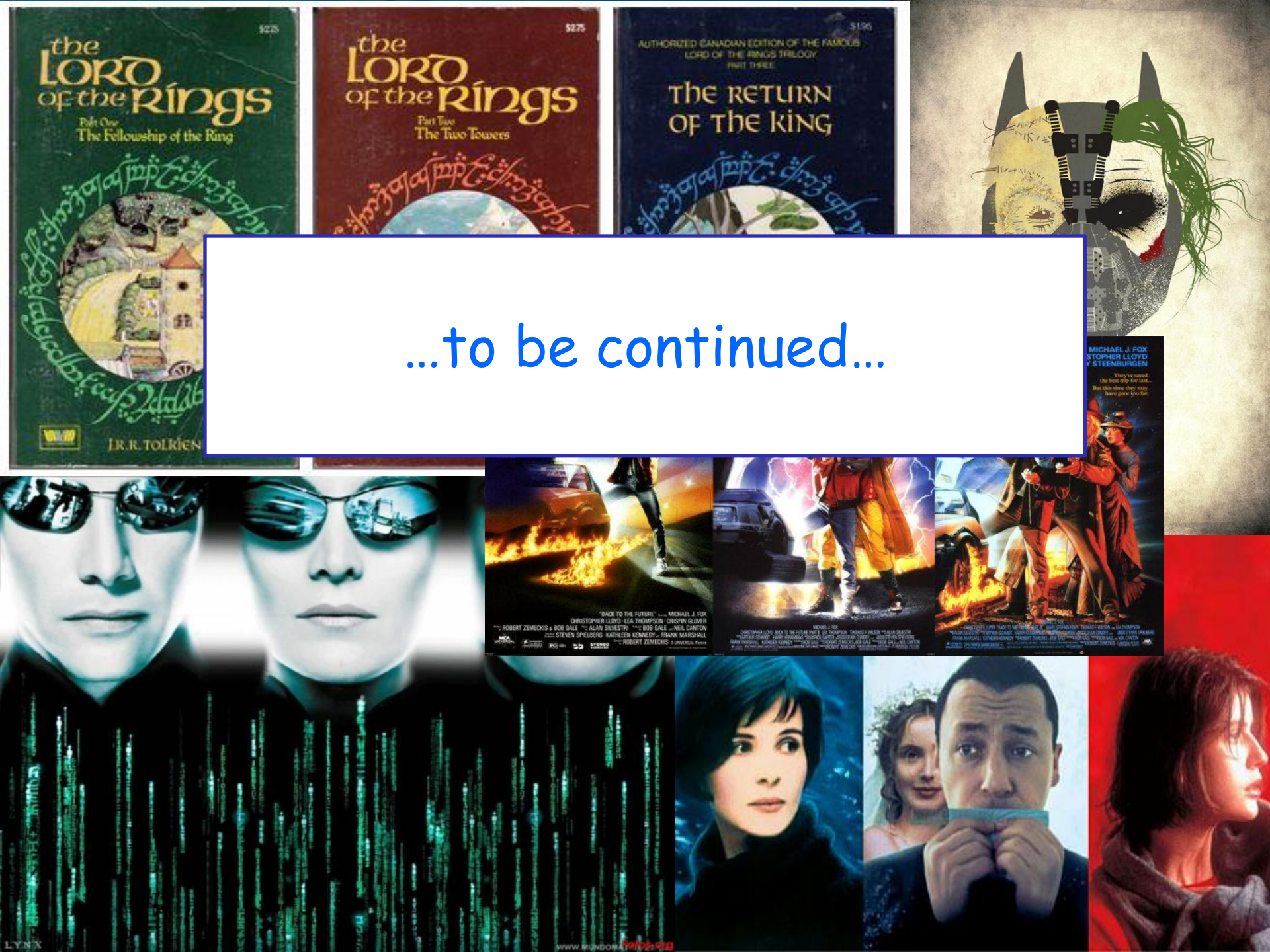
output: distanze in $G=(V,E,w)$ fra tutte le coppie

1. costruisci $G'=(V',E')$ aggiungendo un nodo s e tutti gli archi (s,v), $\forall v \in V$ di peso 0
2. Usa l'alg. di Bellman-Ford per trovare le distanze in $G'=(V',E',w)$ da s e assegnale ad $h()$; ovvero: $h(v)= d(s,v)$ $\forall v \in V$
3. Calcola le distanze fra tutte le coppie in $G=(V,E,w_h)$ usando n volte l'alg. di Dijkstra
4. for each $u,v \in V$, do
   1. calcola la distanza in $G=(V,E,w)$ da u a v: $d(u,v)=d_h(u,v)-h(u)+h(v)$
5. **return** (la matrice del)le distanze $d(u,v)$ $\forall u,v \in V$

Complessità: $O(mn + n^2 \log n)$

Sommario grafico




the LORD of the RINGS
Part One
The Fellowship of the Ring
J.R.R. TOLKIEN
the LORD of the RINGS
Part Two
The Two Towers
AUTHORIZED CANADIAN EDITION OF THE FAMOUS LORD OF THE RINGS TRILOGY
PART THREE
THE RETURN OF THE KING
…to be continued…
MICHAEL J. FOX
"BACK TO THE FUTURE"
LYNX

Esercizio

Dato un grafo $G = (V, E)$ con pesi positivi sugli archi ed un insieme di k centri $C = \{c_1, c_2, \ldots, c_k\} \subseteq V$, si richiede di partizionare l'insieme V in k insiemi $V_1, V_2, \ldots, V_k$ in modo che tutti i vertici in un insieme V_i siamo più vicini al vertice c_i che ad ognuno degli altri centri in C. Formalmente, se $d(u, v)$ indica la distanza tra i vertici u e v in G, deve valere:

$$d(u, c_i) \leq d(u, c_j) \; \forall i, j = 1, \ldots, k \; \forall u \in V_i$$

Progettare un algoritmo che risolva tale problema in tempo $O(|E|+|V| \log |V|)$.

Esercizio

Sia $G = (V, E, w)$ un grafo orientato con pesi positivi sugli archi in cui alcuni archi sono speciali e sono chiamati archi *blu*. Progettare un algoritmo che, dato G, due nodi $s, t \in V$ e un intero k, calcoli un cammino di costo minimo da s a t che usa *al più* k archi blu.