GRAFI: CAMMINI MINIMI E ALGORITMO DI BELLMAN FORD

[Deme, seconda edizione] cap. 14

Sezioni 14.3 e 14.4

Due proprietà importanti

Disuguaglianza Triangolare delle Distanze: le distanze in un grafo rispettano la disuguaglianza triangolare.

Per ogni tripla di vertici s, u, v

$\delta(s, v) \leq \delta(s, u) + \delta(u, v)$

DIMOSTRAZIONE: un cammino minimo $s \rightsquigarrow v$ ha peso ($\delta(s, v)$) **minore o uguale di qualsiasi altro cammino tra s e v**, quindi anche della concatenazione dei cammini $s \rightsquigarrow u$ e $u \rightsquigarrow v$.

Condizione di Bellman

Per ogni arco (u,v) e per ogni vertice s

$\delta(s, v) \leq \delta(s, u) + W(u,v)$

DIMOSTRAZIONE: per definizione **$\delta(u,v) \leq W(u,v)$**, quindi sostituendo $\delta(u,v)$ con $W(u,v)$ nella disuguaglianza triangolare, la disuguaglianza continua a valere.

Appartenenza ad un cammino minimo

Dalla condizione di Bellman possiamo derivare il seguente lemma.

Lemma 1: un arco **(u,v) appartiene ad un cammino minimo** da s a v **se e solo se u è raggiungibile da s** e la condizione di **Bellman** è soddisfatta con **l'uguaglianza** per (u,v), cioè se vale

$$\delta(s, v) = \delta(s, u) + W(u,v)$$

Questo (tra l'altro) ci permette di ricostruire i cammini minimi a partire dalle distanze (per curiosità su come, vedere algoritmo su [Deme]).

Tecnica del Rilassamento

Consideriamo una **stima** D(s,v) **per eccesso di δ(s, v)**, che sia il peso di un cammino sul grafo tra s e v. In pratica, sia **D(s,v) ≥ δ(s, v)** per ogni nodo v appartenente al grafo (s è unico, è il nodo di partenza).

Supponiamo che esista un **arco (u,v)** per cui valga

D(s,v) > D(s,u) + W(u,v)

D(s,u) e W(u,v) sono non minori di δ(s, u) e δ(u, v), quindi **D(s,v) non può essere δ(s, v)**, perché la disuguaglianza triangolare non varrebbe per s, u e v.

Per renderla vera, almeno **localmente** per l'arco (u,v), possiamo **decrementare D(s,v)** con l'assegnazione

D(s,v) <- D(s,u) + W(u,v)

Questa operazione prende il nome di **rilassamento dell'arco (u,v)**.

Nota: se partiamo da D(s,v) = ∞, siamo sicuri che D(s,v) sia sempre (maggiore o uguale al) **la lunghezza di un cammino sul grafo** tra s e v.

Nota

Non è sempre detto che (u,v) sia un arco. In certe applicazioni, esso può essere un cammino, in questo caso la verifica della disuguaglianza triangolare diventa

D(s,v) > D(s,u) + D(u,v)

e la relativa assegnazione

D(s,v) <- D(s,u) + D(u,v)

Questa è la definizione più generica di Rilassamento

Rilassamento e cammini minimi

Applicando esaustivamente l'operazione di rilassamento, per grafi senza cicli negativi, **D(s,v) converge a δ(s, v)**.

Nel momento in cui D(s,v) = δ(s, v) l'ultimo arco (u,v) per cui è stato applicato il rilassamento è quello per cui la condizione di Bellman vale con l'uguaglianza, e **(u,v) appartiene ad un cammino minimo tra s e v**.

Molti degli algoritmi per il calcolo dei cammini minimi si basano sull'operazione di rilassamento, la differenza principale sta nel **come scegliere (u,v)**.

Ad esempio, l'algoritmo di **Dijkstra** sceglie (u,v) garantendo che (u,v) sia l'arco che minimizza D(s,u) + W(u,v) tra tutti gli archi nei quali u è stato raggiunto da un cammino minimo (D(s,u) = δ(s, u)) e v no. In questo modo compie **O(m) operazioni di rilassamento**.

Nota: D è equivalente al vettore d che abbiamo usato in Dijkstra. In particolare **D(s,u) equivale a d[u]**.

Cammini minimi in grafi con archi di peso negativo

Tuttavia, l'algoritmo di **Dijkstra** trova l'albero dei cammini minimi (a partire da un nodo s) per grafi pesati in cui **i pesi degli archi siano solo positivi** (abbiamo anche spiegato il perché).

È possibile calcolare i cammini minimi anche in caso di **pesi negativi**?

In genere sì, ma solo per **grafi che non contengano cicli di peso negativo**. Infatti, in presenza di **cicli di peso negativo non esistono soluzioni finite**. Per i nodi raggiungibili attraverso il ciclo negativo si creano **soluzioni di lunghezza infinita e peso che tende a - ∞**.

Vediamo quindi come trovare i cammini minimi per grafi senza cicli negativi, o in alternativa rilevare la presenza di cicli negativi.

Sottostruttura Ottima

(da Dijkstra)

PROPRIETÀ: Un **sottocammino** di un **cammino minimo è un cammino minimo**.

Abbiamo già detto che la proprietà dei sottocammini minimi di un cammino minimo è la **sottostruttura ottima** per il problema di trovare i cammini minimi.

Proviamo ad applicare la tecnica della **Programmazione Dinamica**.

Costruiamo un algoritmo di PD

Sia $\langle s, v_1, v_2, \ldots v_k \rangle$ un **cammino minimo (di k elementi) da s a v_k**.

Grazie alla **condizione di Bellman**, possiamo scrivere

$\delta(s, v_k) = \delta(s, v_{k-1}) + W(v_{k-1}, v_k)$ **(Lemma 1)**

Ma allora possiamo **ricondurre il problema di trovare $\delta(s, v_k)$ a quello di trovare $\delta(s, v_{k-1})$**!

Il problema, è che noi **non sappiamo quale vertice sarà v_{k-1}** nel nostro cammino minimo. Come fare?

Costruiamo un algoritmo di PD

Il problema, è che noi **non sappiamo quale vertice sarà v_{k-1}** nel nostro cammino minimo. Come fare?

Sappiamo però che il cammino (minimo) $\langle s, v_1, v_2, \ldots v_{k-1}\rangle$ avrà lunghezza k-1, quindi **il sottoproblema di trovare** $\langle s, v_1, v_2, \ldots v_{k-1}\rangle$ **dovrà considerare solo cammini di lunghezza k-1**. A sua volta in sottoproblema di trovare $\langle s, v_1, v_2, \ldots v_{k-2}\rangle$ considererà cammini di lunghezza k-2 e così via **fino a k = 0**.

Ora ragioniamo **Bottom-up**.

Con k = 0, il problema è banale, si tratta di trovare il cammino minimo da s a s con 0 archi. La soluzione è banalmente $\langle s\rangle$.

Usiamo **le stime D(s,v) come struttura di memoizzazione**, ed inizializziamo **$D(s,s) = 0$ e $D(s,v) = \infty$** per gli altri vertici v nel grafo, rispettando la condizione per cui **$D(s,v) \geq \delta(s, v)$**.

Banalmente, a questo punto **$D(s,s) = 0 = \delta(s, s)$.**

Costruiamo un algoritmo di PD

Ora, per k = 1, vogliamo trovare il cammino minimo tra s e v_1, che contenga un solo arco (ricordate che però **non conosciamo v_1**).

Applicando il **rilassamento su tutti gli archi una volta sola**, tra i vari rilassamenti fatti, sicuramente troveremo

$\infty = D(s,v_1) > D(s,s) + W(s,v_1)$ Poiché $W(s, v_1) < \infty$.

Il rilassamento allora porrà $D(s,v_1) = D(s,s) + W(s,v_1)$.

Ma $D(s,s) = \delta(s, s)$ e $\langle s, v_1 \rangle$ **è un cammino minimo** (per la proprietà dei sottocammini minimi), quindi

$D(s,v_1) = \delta(s, s) + W(s, v_1) = \delta(s, v_1)$

Notate che, essendo la lunghezza di un cammino minimo **nessun rilassamento successivo potrà aggiornarla**.

Costruiamo un algoritmo di PD

Al **passo successivo**, cerchiamo $\delta(s, v_2)$. Applichiamo di nuovo il **rilassamento su tutti gli archi**. Tra i vari rilassamenti, troveremo

$D(s,v_2) > D(s,v_1) + W(v_1,v_2)$

Poiché **$D(s, v_1) = \delta(s, v_1)$** e **$\langle s, v_1, v_2 \rangle$ è un cammino minimo**.

Il rilassamento allora porrà **$D(s,v_2) = D(s,v_1) + W(v_1,v_2)$**. Anche qui possiamo dire che **$D(s, v_2) = \delta(s, v_2)$** e $D(s, v_2)$ non sarà più aggiornato.

Iterando il processo, al **k-esimo passo** avremo **$D(s, v_k) = \delta(s, v_k)$**. Ciò sarà vero per ogni vertice v_k raggiungibile da un cammino minimo di al più k archi
(dopo lo dimostreremo, per ora vi basta capire il funzionamento).

Costruiamo un algoritmo di PD

Iterando il processo, al **k-esimo passo** avremo **$D(s, v_k) = \delta(s, v_k)$**. Ciò sarà vero per ogni vertice v_k raggiungibile da un cammino minimo di al più k archi.

Ma noi non sappiamo neanche **qual è la lunghezza k del cammino minimo**!

Tuttavia, **un cammino minimo semplice** (cioè senza vertici duplicati) **può contenere al massimo** tutti i vertici del grafo (n), quindi **n-1 archi**. Di conseguenza, bastano **n-1 iterazioni per trovare i cammini minimi da s a tutti i vertici del grafo**.

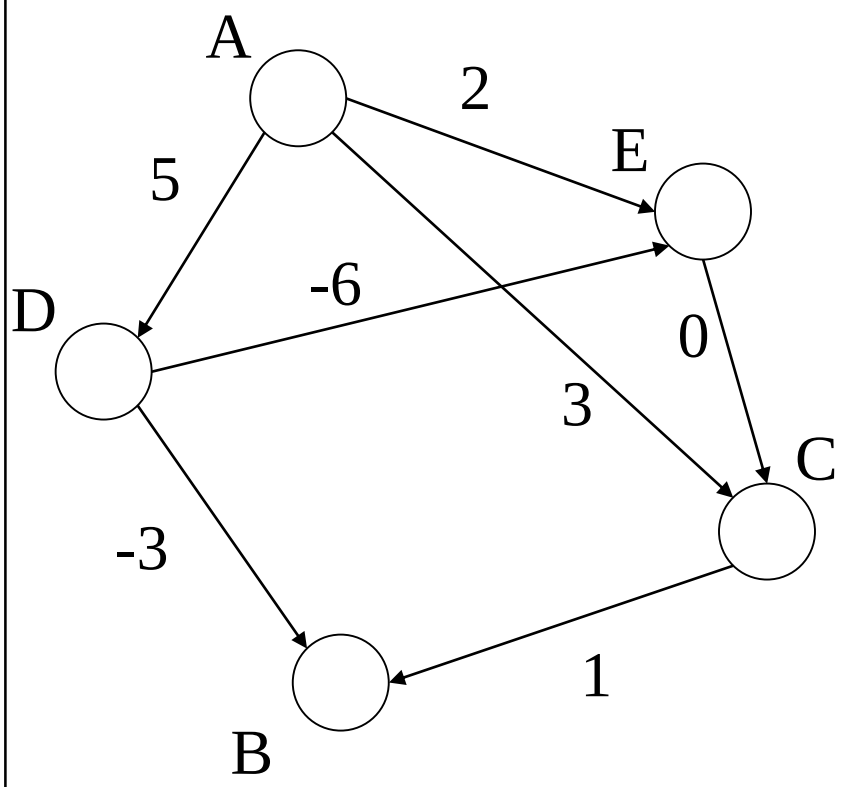
Algoritmo di Bellman-Ford

```
Bellman-Ford (G, W, s)
   INIZIALIZZA (G)              //pone d[v] = ∞, π[v] = NULL per ogni v
   d[s] <- 0
   for i = 1..n-1                                  //k ≤ n-1 per ogni c.m.
      for ogni (u,v) in G then
         if d[v] > d[u] + W(u,v) then          //verifica per rilassamento
            π[v] <- u
            d[v] <- d[u] + W(u,v)              //rilassamento
end
```

Nota: ricordate che d[v] = D(s,v).

Nota2: si vede facilmente che d[v] può essere solo diminuita rispetto al valore iniziale, e che ogni volta è la lunghezza di un cammino.

Esempio

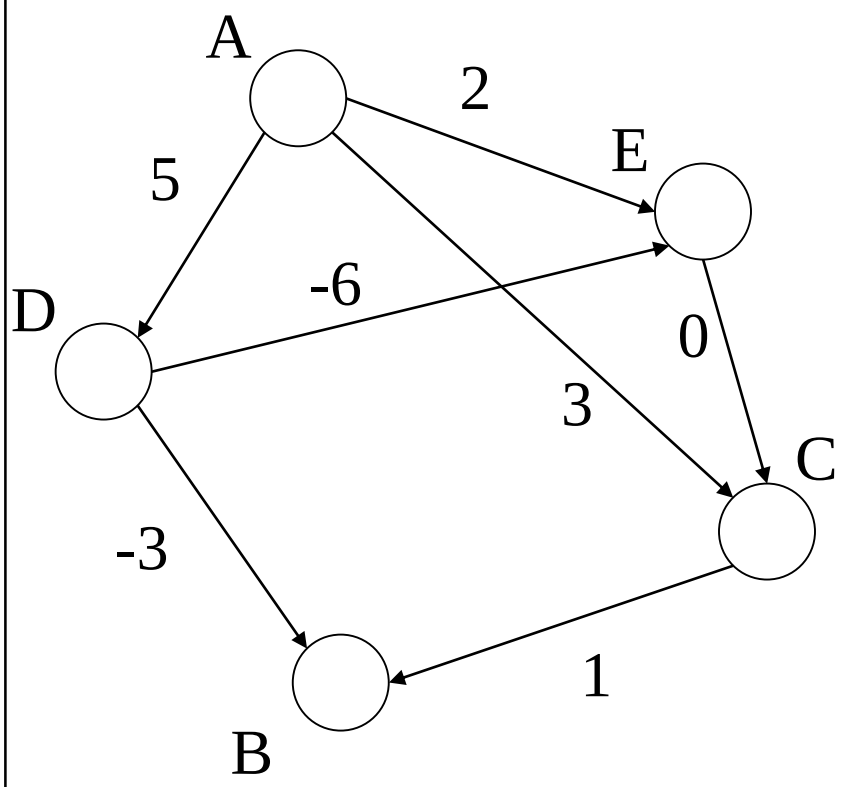


(A.C)	(A,D)	(A,E)	(C,B)	(D,B)	(D,E)	(E,C)

v	A	B	C	D	E
d(v)	0	∞	∞	∞	∞

v	A	B	C	D	E
π(v)	NULL	NULL	NULL	NULL	NULL

Esempio (dopo I ciclo)

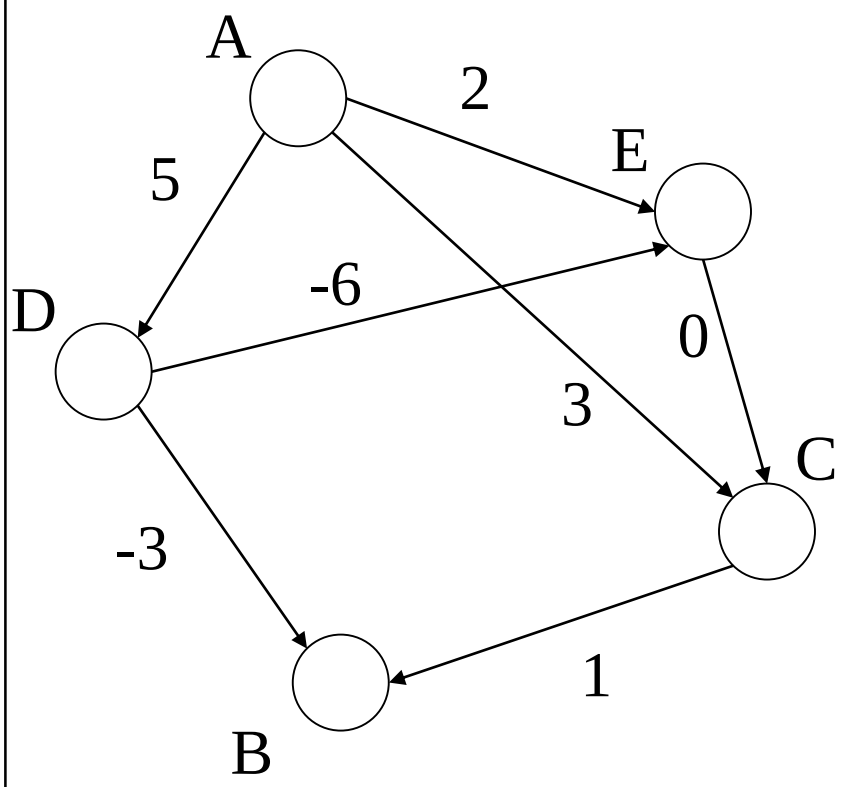


(A.C)	(A,D)	(A,E)	(C,B)	(D,B)	(D,E)	(E,C)

v	A	B	C	D	E
d(v)	0	2	-1	5	-1

v	A	B	C	D	E
π(v)	NULL	D	E	A	D

Esempio (dopo II ciclo)

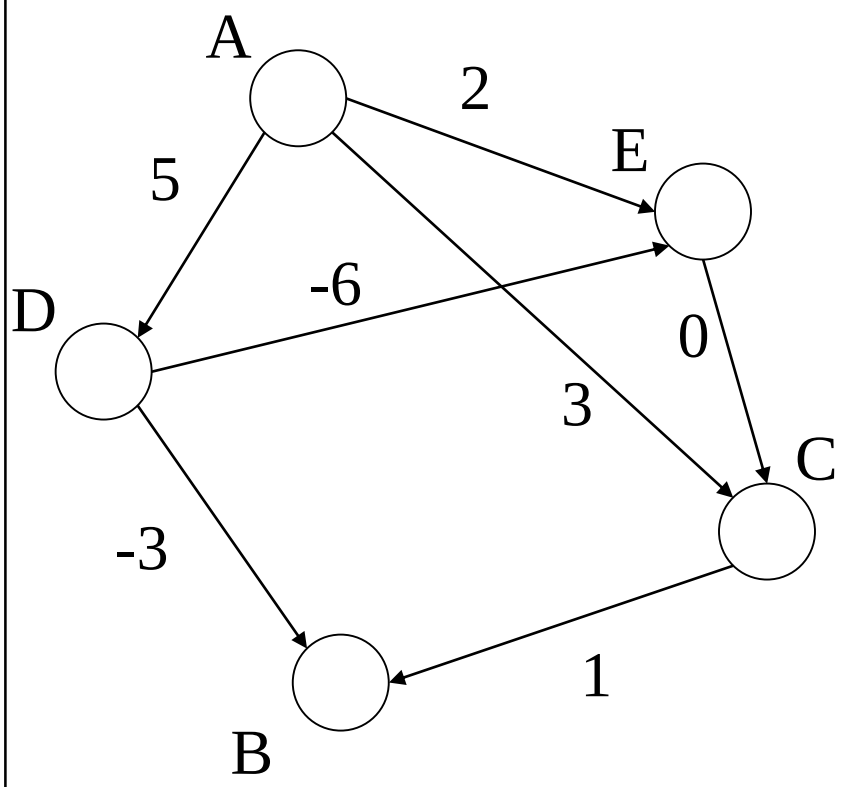


(A.C)	(A,D)	(A,E)	(C,B)	(D,B)	(D,E)	(E,C)

v	A	B	C	D	E
d(v)	0	0	-1	5	-1

v	A	B	C	D	E
π(v)	NULL	C	E	A	D

Esempio (dopo III ciclo – non cambia più)



(A.C)	(A,D)	(A,E)	(C,B)	(D,B)	(D,E)	(E,C)

v	A	B	C	D	E
d(v)	0	0	-1	5	-1

v	A	B	C	D	E
π(v)	NULL	C	E	A	D

Correttezza

Sia **$\delta(s, v_k)$** la distanza (peso del cammino minimo) da s a v_k.

Definiamo l'invariante **KPATH**:

Dopo la k-esima iterazione del ciclo esterno (for i = 1..n-1) si ha **$d[v_k] = \delta(s, v_k)$** per ogni nodo v_k per cui il **cammino minimo da s a v_k** è composto al più da **k archi**.

DIMOSTRAZIONE. (per induzione)

BASE: per **k = 0 (prima del ciclo)**, per il cammino da s ad s (l'unico cammino composto da 0 archi) **d[s] = 0 e d[u] = ∞** per tutti gli altri nodi u.

PASSO: supponendo che KPATH sia vera dopo k iterazioni, dimostriamo che rimane vera anche dopo k+1 iterazioni.

Correttezza - II

Sia v_{k+1} un nodo il cui cammino minimo $s \rightsquigarrow v_k \rightsquigarrow v_{k+1}$ **è composto da k+1 archi**.

Sappiamo che

$\delta(s, v_{k+1}) = \delta(s, v_k) + W(v_k, v_{k+1})$ per il Lemma 1 e

$d[v_k] = \delta(s, v_k)$ per ipotesi induttiva

Quindi possiamo dire che

$\delta(s, v_{k+1}) = d[v_k] + W(v_k, v_{k+1})$

Al passo k+1, ci sono 2 casi:

Caso 1: **$d[v_{k+1}]$ viene aggiornata**,

Allora $d[v_{k+1}] = d[v_k] + W(v_k, v_{k+1}) = \delta(s, v_k) + W(v_k, v_{k+1}) = \delta(s, v_{k+1})$

Caso 2: **$d[v_{k+1}]$ non viene aggiornata**,

poiché $d[v_{k+1}]$ è il peso di un cammino da s a v_{k+1} in G,

$d[v_{k+1}] \geq \delta(s, v_{k+1})$

poiché non è stata aggiornata,

$d[v_{k+1}] \leq d[v_k] + W(v_k, v_{k+1}) = \delta(s, v_{k+1})$,

possiamo avere solo

$d[v_{k+1}] = \delta(s, v_{k+1})$ quindi **il cammino trovato è comunque minimo**.

Correttezza - III

Quindi, dopo la k-esima iterazione, **KPATH vale** e **$d[v_k] = \delta(s, v_k)$** per ogni nodo v_k per cui il **cammino minimo da s a v_k, che è sicuramente un cammino semplice, è composto al più da k archi**.

Ma in un grafo, un cammino minimo può contenere al più tutti i nodi appartenenti a V, quindi un cammino minimo non può contenere più di **n-1 archi**.

Allora, dopo n-1 iterazioni $d[u] = \delta(s, u)$ per ogni nodo u. Quindi i cammini restituiti dall'algoritmo sono effettivamente quelli minimi.

CVD.

Complessità

L'algoritmo fa **n-1 iterazioni del ciclo esterno**.

Per ogni iterazione, è facile notare che considera **tutti gli archi del grafo** con una serie di operazioni di costo O(1).

Quindi la complessità dell'algoritmo di Bellman-Ford è

O((n-1)*m) = **O(nm)**

(la complessità è maggiore di Dijkstra, quindi nel caso di grafi con archi di peso solo positivo Dijkstra è più efficiente).

Domanda: perché Dijkstra «batte» Bellman-Ford? Da qualche parte in questa presentazione è spiegato…potrebbe essere una domanda all'esame orale!

Trovare cicli negativi

Il limite di n-1 archi per un cammino minimo vale solo per cammini semplici, cioè senza vertici ripetuti.

Se un cammino minimo contiene vertici ripetuti, vuol dire che nel grafo esiste un ciclo, ed il cammino percorre il ciclo più volte. Abbiamo quindi escluso possibili soluzioni «migliori»?

Se il ciclo ha peso strettamente positivo, il cammino minimo è solo quello ottenuto percorrendolo una sola volta (ad ogni iterazione del ciclo si aggiunge il suo peso a quello del cammino), quindi gli n-1 cicli dell'algoritmo sono più che sufficienti.

Se il ciclo ha peso 0, viene rilevato il cammino minimo ottenuto percorrendo il ciclo una volta sola. I cammini trovati eseguendo il ciclo più volte saranno anch'essi minimi, ma a noi basta trovarne uno qualsiasi.

Trovare cicli negativi - II

Nel caso in cui il ciclo abbia peso negativo, ogni volta che lo percorriamo troviamo un cammino minimo più lungo e più «minimo». Per **cicli negativi non sarà mai possibile trovare una soluzione ottima**, perché essa tende ad una lunghezza infinita e a peso $-\infty$.

Ma da ciò possiamo ricavare un metodo efficiente per **rilevare cicli negativi**. Infatti ci basta **controllare se l'n-esima iterazione aggiorna qualche distanza. Se sì, esiste un ciclo negativo.**

Bellman-Ford – cicli negativi

```
BF-NEG (G, W, s)
   INIZIALIZZA (G)
   d[s] <- 0
   for i = 1..n-1                                  //k ≤ n-1 per ogni c.m.
      for ogni (u,v) in G then
         if d[v] > d[u] + W(u,v) then              //verifica per rilassamento
            π[v] <- u
            d[v] <- d[u] + W(u,v)                  //rilassamento
    for ogni (u,v) in G then                              //controlliamo cicli neg.
      if d[v] > d[u] + W(u,v) then return errore
end
```

Ottimizzazione di Bellman-Ford per DAG

È facile notare che l'algoritmo di **Bellman-Ford fa molti passi di rilassamento inutili**.

In particolare, l'assegnazione

$D(s, v_k) <- D(s, v_{k-1}) + W(v_{k-1}, v_k)$

È **inutile se $D(s, v_{k-1}) > \delta(s, v_{k-1})$**.

Infatti sarà poi necessario almeno un ulteriore **rilassamento dello stesso arco** per calcolare $\delta(s, v_k)$.

Per ottimizzare i rilassamenti, **Dijkstra** considera solo archi (v_{k-1},v_k) per cui **$D(s, v_{k-1}) = \delta(s, v_{k-1})$**, ma Dijkstra funziona solo con archi di peso positivo. È possibile ottimizzare l'ordine di rilassamento per altri tipi di grafi?

Sì, in particolare vediamo un'ottimizzazione di Bellman-Ford per **grafi diretti aciclici**.

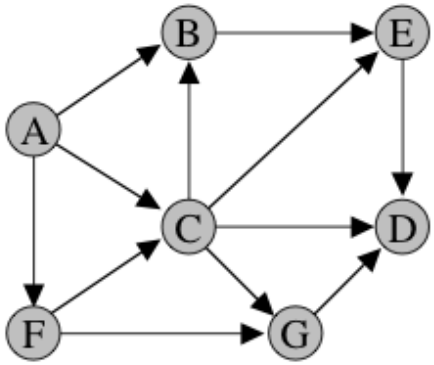
(da Ordinam. Topol.) Ordine topologico

Dato un **grafo orientato aciclico**, un **ordine topologico** è un ordine lineare dei suoi nodi tale che se nel grafo vi è un arco (u, v), allora u precede v nell'ordine

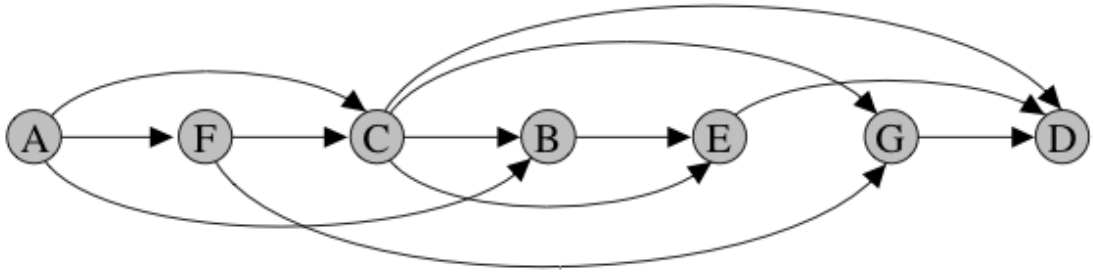
Proprietà ed esempi

Un grafo orientato aciclico **possiede sempre un ordine topologico**.

Un DAG **può possedere diversi** ordini topologici.



(a)



(b)

(a) Un grafo aciclico; **(b) Ordine topologico dei nodi del grafo (a)**

Algoritmo di ordinamento topologico basato su DFS.

```
TOPOLOGICAL-SORT (G)
    INIZIALIZZA (G)
    ord <- un vettore di lunghezza n
    t <- n-1
    for ogni u ∈ V do
        if color [u] = white
        then DFS-TOPOLOGICAL (G, u, ord, t)
    return ord

DFS-TOPOLOGICAL (G, u, ord, t)
  color [u] <- gray
  d[u] <- time <- time+1
  for ogni v adiacente ad u
    if color[v] = white
      π[v] <- u
      DFS-TOPOLOGICAL (G, v, ord, t)
  color[u] <- black
  f[u] <- time <- time+1
  ord[t] <- u
  t--
```

Verso l'ottimizzazione

Dato un ordinamento topologico ord, in cui un vertice u precede un vertice v (**u < v**), allora **in qualsiasi cammino che contenga sia u che v (compresi i cammini minimi)** su quel grafo, **u precederà v**.

Questa proprietà ci suggerisce che se rilassiamo gli archi in maniera che **D(s,u) sia uguale a δ(s, u) prima che l'arco (u,v) sia considerato** per il rilassamento, non effettueremo rilassamenti inutili (da ripetere).

Allora, **se un grafo è un DAG**, possiamo prima calcolare un **ordinamento topologico**, e poi **rilassare** tutti **gli archi** una **sola volta seguendo l'ordine topologico**.

Bellman-Ford - DAG

```
BF-DAG (G, W, s)
   INIZIALIZZA (G)
   d[s] <- 0
   ord <- TOPOLOGICAL-SORT (G)
   for i = 0..n-1
      for ogni (ord[i],v) then     //ovviamente, se (ord[i],v) esiste
         if d[v] > d[ord[i]] + W(ord[i],v) then
            π[v] <- ord[i]
            d[v] <- d[ord[i]] + W(ord[i],v)
end
```

L'algoritmo BF-DAG ha complessità **O(m+n)**.

Esempio

Potete vedere l'esecuzione di Bellman-Ford in:

http://visualgo.net/sssp

Bellman-Ford distribuito

Una applicazione concreta dell'algoritmo di Bellman-Ford si trova nelle reti.

In una rete, diversi computer (i nodi) sono connessi tramite collegamenti (gli archi) che impiegano del tempo (i pesi degli archi) per consegnare determinati pacchetti.

Un **algoritmo di routing** serve ai nodi per capire verso che direzione inviare i pacchetti (o a volte, che percorso fargli fare) affinché arrivino a destinazione nel minor tempo possibile.

La direzione (cioè a che nodo inviare il pacchetto) ed una stima del tempo necessario sono mantenuti in **una tabella di routing** (che «riassume» d e π).

Il problema è che inizialmente **un singolo nodo della rete non conosce tutti gli altri nodi e le distanze da essi**, ma solo quelli ad esso adiacenti.

Bellman-Ford distribuito

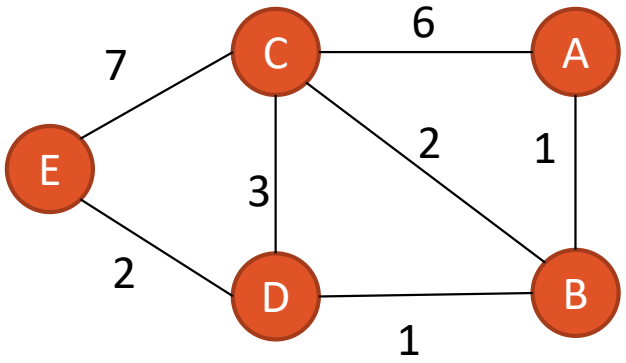
In questo contesto, sembra più efficiente (ed affidabile) calcolare in maniera **distribuita** le tabelle di routing.

Per farlo, si usa una variazione (distribuita) dell'algoritmo di Bellman-Ford.

A turno, ogni nodo scambia la sua tabella di routing con un suo vicino, e cerca di compiere dei **passi di rilassamento** sulla sua tabella usando quella del vicino.

Se non ci sono modifiche alla rete, l'algoritmo termina in **n-1** passi di esecuzione.

Bellman-Ford distribuito - passo



dest	Costo, next
A	0, A
B	1, B
C	6, C

dest	Costo, next
A	0, A
B	1, B
C	**3, B**

dest	Costo, next
A	0, A
B	1, B
C	3, B
D	**2, B**

dest	Costo, next
A	1, A
B	0, B
C	2, C
D	1, D

Alla prima iterazione, A riceve la tabella di B e cerca di compiere dei passi di rilassamento usandola.

$D(A,A) > D(A,B) + D^B(B,A)$? NO

$D(A,B) > D(A,B) + D^B(B,B)$? NO

$D(A,C) > D(A,B) + D^B(B,C)$? SI **6 > 2 + 1 = 3** allora aggiorno la tabella di A

$D(A,D) > D(A,B) + D^B(B,D)$? SI **∞ > 1 + 1 = 2** allora aggiorno la tabella di A

(l'apice ^{B} è utilizzato solo per indicare che la stima proviene da B)

Cosa devo aver capito fino ad ora

- Disuguaglianza triangolare
- Condizione di Bellman
- Tecnica del rilassamento di archi
- Sottostruttura ottima del problema dei cammini minimi
- Algoritmo di Bellman-Ford
  - Correttezza
  - Complessità
- Bellman-Ford e cicli negativi
- Bellman-Ford per grafi diretti aciclici
- Bellman-Ford distribuito (vedetelo più come un accenno a ciò che avete visto in maniera più approfondita nel corso di Reti 1)

…se non ho capito qualcosa

- Alzo la mano e chiedo
- Ripasso sul libro
- Chiedo aiuto sul forum
- Chiedo o mando una mail al docente