Progettazione di Algoritmi
Esercitazioni

Ivano Salvo

Corso di Laurea in Informatica

SAPIENZA
UNIVERSITÀ DI ROMA

Esercitazione **3D**, 8 maggio 2020

Esercizi 3D(a)

Affinità e Divergenze: Dijkstra e alberi minimi di copertura

Archi peso negativo e cost. additive

ESERCIZIO 1

Sia $G = (V, E)$ un qualsiasi grafo orientato con pesi sugli archi, pesi che possono essere anche negativi ma in cui non sono presenti cicli di peso negativo.

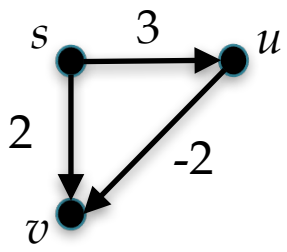
- Dimostrare che l'algoritmo di Dijkstra su grafi di questo tipo non calcola necessariamente i cammini di costo minimo tra la sorgente e gli altri nodi del grafo. [anche **Cormen, 24.4-2**]

- Per il calcolo dei cammini di costo minimo in G si suggerisce il seguente algoritmo:
  Sia M il costo minimo tra i costi degli archi di G. Modifichiamo i pesi degli archi di G sommando a ciascuno di questi l'intero $|M|$ abbastanza grande da renderli tutti positivi. Al grafo che si ottiene G' (che ha pesi positivi) applichiamo l'algoritmo di Dijkstra.
  I cammini minimi che vengono così calcolati sono anche cammini minimi per il grafo originale G? Motivare la risposta.

Archi di costo negativo: soluzione

1) Perché l'algoritmo di Dijkstra fallisce con archi di peso negativo? Sostanzialmente perché via via seleziona i nodi più vicini alla radice (un po' come una BFS, ma generalizzando la nozione di vicinanza con i pesi sugli archi).

Tuttavia, un nodo che "sembrava" stabilizzato alla sua distanza minima potrebbe essere raggiunto da un cammino di costo minore a causa di archi con pesi negativi. E quindi ecco un esempio minimale:

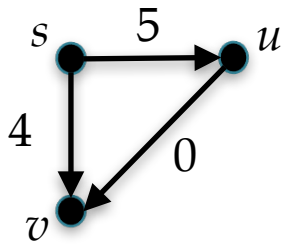


Al nodo *v* viene assegnata distanza 2, in quanto nodo più vicino a *s*, e viene ritenuto dall'algoritmo di Dijkstra "stabilizzato" al suo costo minimo. Infatti, con pesi positivi, nessun cammino che passa per *u* potrà essere migliore.

Costanti additive: soluzione

2) Purtroppo, aggiungere una costante additiva M ai pesi di tutti gli archi non permette generalizzare l'algoritmo di Dijkstra. Il problema è che la costante additiva M **influenza il peso dei cammini in modo diverso** a seconda della lunghezza del cammino: $w'(p)=w(p) + |p| \cdot M$.

Il controesempio può essere facilmente derivato dall'esempio precedente, aggiungendo 2 per rendere non negativo l'arco $u \rightarrow v$. Questa costante penalizza comunque il cammino $s \rightarrow u \rightarrow v$ più del cammino $s \rightarrow v$.



Ritorneremo su questo punto in un esercizio successivo.

Alberi minimi di copertura: proprietà

ESERCIZIO 11

Sia $G = (V, E)$ un grafo non orientato connesso e con pesi sugli archi. Sia T un minimo albero di copertura per G.

a. Provare che T è ancora un minimo albero di copertura anche per il grafo che si ottiene da G incrementando di una stessa costante c il peso degli archi.

b. Provare che T deve contenere un arco di peso minimo.

[anche **Cormen, 23.3-1**]

c. Provare che qualora i pesi di G siano tutti distinti, allora T è l'unico minimo albero di copertura. Mostrare che che la condizione sul fatto che i pesi siano tutti distinti non è necessaria per l'unicità del minimo albero di copertura.

Costanti additive e alberi: soluzione

Soluzione a) Questa si proprietà si può dimostrare in molti modi.

Forse il più semplice è il seguente: tutti gli alberi di copertura hanno esattamente n-1 archi. Una volta che considero una funzione peso $w'(e)=w(e)+c$, avrò che per ogni albero di copertura $w'(T) = w(T)+(n-1)c$. Da ciò discende subito che $w(T) < w(T'')$ implica che $w'(T')< w'(T'')$.

Questa situazione è abbastanza comune e spesso la **cardinalità delle soluzioni ammissibili** tutte uguali è una delle condizioni che rendono **applicabili algoritmi greedy** come quelli che trovano i minimi alberi di copertura.

Soluzione b) Questo è molto istruttivo nei ragionamenti con alberi minimi di supporto.

Supponiamo per assurdo esista un arco di peso minimo (u, v) in G, non scelto nell'albero **minimo** di copertura T di G.

Consideriamo un qualsiasi taglio $(S, V \setminus S)$ di G, con $u \in S$ e $v \notin S$. Siccome l'albero è connesso, c'è un arco (x, y) che attraversa il taglio e per ipotesi $w(x, y) > w(u, v)$. Rimuovendo (x, y) da T e aggiungendo (u, v) ottengo un albero di supporto T', tale che $w(T') < w(T)$.

Ovviamente potevo applicare il **Teorema dell'arco leggero**.

Pesi distinti, unicità dell'albero

Soluzione c) Possiamo applicare il ragionamento precedente iterandolo su tutti gli archi (e quindi i tagli "attraversati").

Supponiamo T non sia l'unico albero di copertura e sia (u, v) un arco in T ma non in un altro albero T'. Consideriamo un taglio $(S, V \setminus S)$ di G, con $u \in S$ e $v \notin S$ e sia (u', v') l'arco in T' che attraversa il taglio. Ma seguendo il ragionamento precedente $w(u, v)$ è minimo tra tutti i pesi degli archi che attraversano il taglio $(S, V \setminus S)$ e **non sono in T** (**attenzione!**) altrimenti costruirei un albero T'' tale che $w(T'') < w(T) = w(T')$.

Siccome gli archi hanno tutti pesi distinti, (u, v) **è l'unico arco che realizza il minimo** su ogni taglio che attraversa. Quindi tale arco **deve necessariamente** stare nell'albero di copertura minimo T.

Essendo questo ragionamento vero per ogni arco, T è necessariamente l'unico albero di copertura di G.

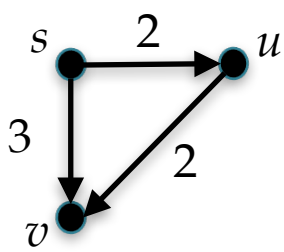
Cammini minimi e spanning trees

ESERCIZIO 12

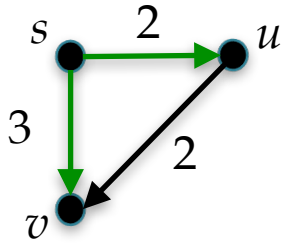
- Sia G un grafo non diretto, connesso e pesato. Dimostrare o confutare che un albero dei cammini minimi di G è anche un albero di copertura di peso minimo.
- Sia G un grafo non diretto, connesso e pesato dove i pesi sono tutti diversi tra loro. Sia T un minimo albero di copertura di G, sia s un nodo e T_s l'albero dei cammini minimi da s verso tutti gli altri nodi di G. Dimostrare oppure fornire un controesempio che T_s e T condividono almeno un arco.

Cammini minimi e spanning trees

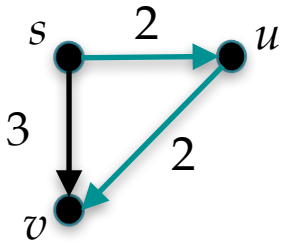
Soluzione 1) È facile vedere che questo **non è vero**. Ancora una volta basta un "triangolo" per trovare un controesempio.



grafo



cammini minimi



albero minimo di supporto

2) Viceversa il DAG dei cammini minimi ha sempre un arco in comune con l'albero minimo di copertura: è sufficiente applicare il **Teorema dell'arco leggero** sul taglio $(\{s\}, V \setminus \{s\})$ [in entrambi i casi si sceglie l'arco di peso minimo]. Oppure applicare **l'algoritmo di Prim** partendo dalla sorgente s: sceglie lo stesso arco di Dijkstra [e concludere **grazie all'unicità** dell'albero minimo nel caso di pesi diversi].

Esercizi 3D(b)

Algoritmo di Dijkstra

Verifica Soluzione cammini minimi

Esercizio 24.4-4 [Cormen]

Supponete di avere una soluzione dei cammini minimi da una sorgente s in un grafo con pesi positivi sugli archi, data come vettore di padri p e vettore dei costi c. Scrivere un algoritmo efficiente per determinare se la soluzione è effettivamente corretta.

Soluzione: Questo esercizio si risolve ricordando che per i cammini minimi vale la proprietà di **sottostruttura ottima**: i sottocammini di cammini minimi sono cammini minimi tra i nodi ai loro estremi.

Occorre quindi controllare che per tutti gli archi $u \to v$ del DAG dei cammini minimi $c(u) + w(u \to v) = c(v)$ e per ogni nodo t tale che $t \to v$, ho che $c(t) + w(t \to v) \geq c(v)$.

Occorre anche controllare che effettivamente s sia la sorgente (e quindi $p(s) = s$ (o NIL) e $c(s) = 0$). Vediamo un semplice algoritmo.

Verifica Soluzione cammini minimi

$\mathcal{O}(n)$

$\mathcal{O}(n+m)$

```
def verifyShortestPaths(G, w, s, p, d):

    forall v ∈ G.V do
        if (v=s) and (d(v)≠0 or p(v)≠NIL)) return FALSE;
        else if (p(v)=NIL and d(v)<∞) return FALSE;
        else if (d(v)≠d(p(v))+w(p(v)→v) return FALSE;

    forall u ∈ G.V do
        forall v ∈ G.adj(u) do
            if (v≠s and d(u)+w(u→v)<d(v))
                return FALSE;

    return TRUE;
```

Numero di Cammini Minimi

ESERCIZIO 2

Dato un grafo orientato G con pesi positivi sugli archi ed un nodo s di G, l'algoritmo di Dijkstra calcola l'albero dei cammini minimi da s ad un qualsiasi altro nodo di G che è raggiungibile da s. In generale, tra il nodo s e un altro nodo u di G può esserci più di un cammino minimo.

- Descrivere un algoritmo che calcoli per ogni nodo u il numero di tutti i possibili cammini di peso minimo da s a u.
- Discutere la complessità dell'algoritmo.

Numero di cammini minimi: soluzione

Soluzione: Abbiamo visto come calcolare efficientemente il numero di cammini in un DAG. Ovviamente, l'algoritmo di Dijkstra produce un albero in cui c'è sempre solo 1 cammino tra due nodi.

Ovviamente c'è più di un cammino minimo **quando ci sono due o più cammini che hanno lo stesso peso**. Questa cosa accade quando trovo un 'nuovo' cammino minimo per un nodo, nella procedura che nel Cormen è chiamata *relax*(u, v, w):

```
def relax(u, v, w):
    if (v.d > u.d + w(u→v))
        v.d = u.d + w(u→v)
        v.π = u
```

Piuttosto di memorizzare un albero, **possiamo memorizzare un DAG**, facendo in modo che l'attributo π non sia un singolo nodo, ma una **lista di nodi predecessori**.

Avendo questa struttura, possiamo ricostruire il numero di cammini.

Numero di cammini minimi: soluzione

```
def relax(u, v, w):
    if (v.d > u.d + w(u→v))
        v.d = u.d + w(u→v)
        v.π = ⟨v⟩
    else if (v.d = u.d + w(u→v))
        v.in = v.in + 1
        insert(u, v.π)
```

Avendo il un nodo u, il numero di cammini minimi da s a u sarà calcolato come nell'algoritmo che conta i cammini in un DAG (vedi slides Esercitazione **3**), in questo caso applicandolo su un grafo i cui archi sono orientati "al contrario", partendo da u.

Complessità $\mathcal{O}(m+n)$.

Il numero cammini in un DAG può anche essere calcolato **bottom-up** partendo dai nodi terminali e guardando gli archi a ritroso [**esercizio**].

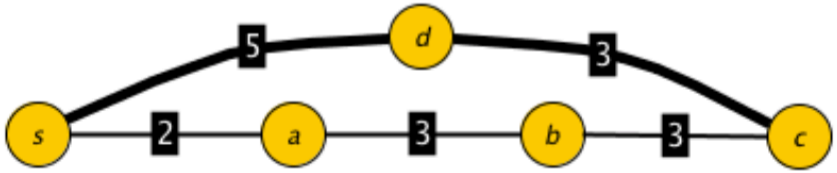
C'è una **soluzione di questo problema anche nelle dispense**, tuttavia la soluzione è meno modulare.

Cammino super-minimo

ESERCIZIO 5

Un cammino da un nodo u a un nodo v si dice super-minimo se ha peso minimo tra tutti i cammini da u a v e inoltre tra tutti i cammini di peso minimo da u a v ha il minimo numero di archi. Dato un grafo pesato G tale che i pesi sono interi positivi, si vogliono trovare i cammini super-minimi da un nodo s.

Ad esempio, nel grafo qui sotto vogliamo il cammino (s, d, c) e non quello di pari peso ma più lungo (s, a, b, c).



Mostrare come modificare i pesi del grafo G in modo tale che applicando Dijkstra al grafo coi nuovi pesi si ottengono i cammini super minimi di G.

Cammino super-minimo: soluzione

Soluzione: Occorre perturbare il valore degli archi in modo da favorire i cammini più corti: abbiamo già visto che questo si può ottenere aggiungendo una costante additiva al peso di tutti gli archi.

Tuttavia, occorre scegliere tale costante in modo che non faccia diventare sfavorevole un cammino strettamente migliore coi pesi originali.

Avendo gli archi pesi interi, **due cammini di peso diverso hanno come minima differenza 1**. D'altra parte, avendo n nodi, i cammini semplici hanno al più n-1 archi. Quindi, definendo una nuova funzione di costo $w'(e) = w(e) + 1/n$, abbiamo che ogni cammino p avrà un costo $w'(p)=w(p)+|p|/n < w(p)+1$. Ergo, il peso dei cammini cambia meno di 1.

Conoscendo la precisione del peso degli archi, possiamo applicare questo trucco anche se i pesi non sono interi: se λ è la minima differenza tra due cammini allora la costante da aggiungere è λ/n.

Esercizi 3D(c)

Proprietà degli alberi minimi di copertura

Archi leggeri e sicuri

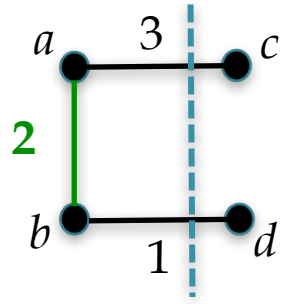
Definizione: *Sia $A \subseteq E$ un insieme di archi contenuto in un albero minimo di copertura. Un arco e è sicuro per A, se $A \cup \{e\}$ è ancora contenuto in un albero minimo di copertura.*

Teorema: *Sia $A \subseteq E$ un insieme di archi contenuto in un albero minimo di copertura e sia $(S, V \setminus S)$ un qualsiasi taglio che rispetta A. Un arco leggero che attraversa il taglio è sicuro per A.*

Esercizio 23.1-2 [Cormen]

Vale il seguente converso del Teorema dell'arco leggero?
Sia $A \subseteq E$ un insieme di archi contenuto in un albero minimo di copertura e sia $(S, V \setminus S)$ un qualsiasi taglio che rispetta A. Un arco sicuro per A che attraversa il taglio è necessariamente leggero.

Soluzione: Questo teorema è falso. Consideriamo il seguente grafo:



Su questo taglio, entrambi gli archi sono sicuri (il grafo è peraltro già un albero), ma chiaramente l'arco (a, c) non è leggero. Ovviamente l'essere sicuro non è relato alla possibilità di scelta in un algoritmo goloso per MST.

Archi leggeri e sicuri

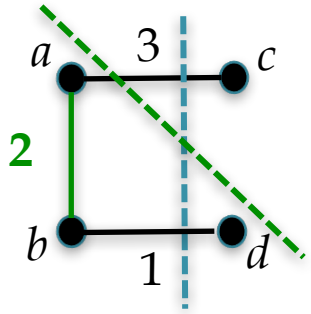
Esercizio 23.1-3 [Cormen]

Sia (u, v) un arco che appartiene a un qualche un albero di copertura T di un grafo $G=(V, E)$. Allora (u, v) è leggero per un qualche taglio $(S, V \setminus S)$ di G.

Soluzione: Sia T un albero di copertura a cui appartiene (u, v). Se rimuoviamo l'arco (u, v) da T dividiamo T in due componenti connesse chiamiamole T_u e T_v. I nodi di questi due alberi formano un taglio $(T_u(V), T_v(V))$: se ci fosse un arco (x, y) su questo taglio tale che $w(x, y) < w(u, v)$, potrei aggiungere l'arco (x, y) a T_u e T_v ottenendo un albero di copertura T' per cui $w(T') < w(T)$.

Riassumendo: ***un arco sicuro non necessariamente è leggero per un taglio arbitrario, ma esiste sempre un taglio per cui esso è leggero.***

Ricordiamo l'arco (a, c) nell'esempio precedente.



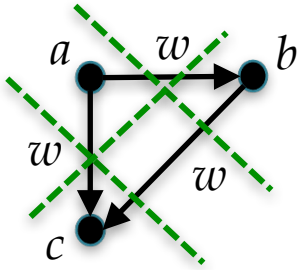
Archi leggeri e sicuri

Esercizio 23.1-4 [Cormen]

Trovare un esempio per cui l'insieme di archi: $\{(u, v) \mid \exists S \subseteq V\ (u, v)$ è leggero per $(S, V \setminus S)\}$ ***non è*** *un albero di supporto.*

Soluzione: Ancora una volta una proprietà che sembra in contraddizione con le precedenti.

Tuttavia, nel caso in cui ci siano archi di peso uguale, può succedere che tutti questi archi formino un grafo ciclico, quindi non un albero.
Vediamo un esempio banale:



Tutti questi archi sono leggeri per 2 tagli, ma ovviamente la loro unione non fa un albero!

Archi pesanti

Esercizio 23.1-5 [Cormen]

Sia $G=(V, E)$ un grafo ed $e \in E$ un arco di peso massimo in un qualche ciclo di G. Far vedere che esiste un albero di copertura minimo di $G'=(V, E \setminus \{e\})$ che è anche albero di copertura di G.

Soluzione: Siccome e appartiene a un ciclo, G' è connesso. Ha quindi un albero di copertura minimo T. Ovviamente T copre tutto V e non è conveniente rimpiazzare nessun arco di T con e in quanto non sarà mai leggero per nessun taglio.

Osservazione: se e non è in un ciclo potrebbe essere necessario per mantenere connesso l'albero di copertura.

Corollario: Ogni arco che non appartiene a un ciclo è necessariamente nell'albero di copertura.

Dim: Sia $e=(u, v)$ arco che non appartiene a nessun ciclo. Tolto e, non c'è nessun cammino tra u e v.

Ancora sull'unicità del MST

Esercizio 23.1-6 [Cormen]

Dimostrare che se per ogni taglio $(S, V \setminus S)$ di G esiste un unico arco leggero, allora esiste un unico albero minimo di copertura per G.

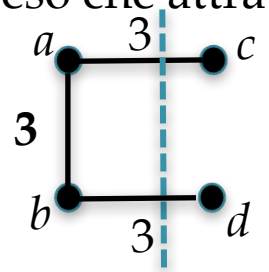
Viceversa, non è detto che l'unicità del MST implichi l'unicità degli archi leggeri sui tagli.

Soluzione: Supponiamo esistano due MST distinti T e T'. Sia $(u, v) \in T$ e $(u, v) \notin T'$. Consideriamo il taglio formato dai nodi connessi a u in T e da quelli connessi a v, una volta rimosso (u, v) da T. Sia (u', v') l'arco di T' che attraversa questo taglio. Per ipotesi $w(u', v') > w(u, v)$. Quindi otterrei un MST T'' tale che $w(T'') < w(T') = w(T)$. Assurdo.

Osservazione: il fatto che archi tutti di peso diverso implichino l'unicità dell'MST è un facile corollario di questa proprietà.

Il viceversa è falso perché potrei avere che l'unicità dipende dalla necessità di inserire tutti gli archi con lo stesso peso che attraversano un taglio, per mantenere la connessione.

Per esempio quando G è già un albero.



Sottoinsiemi di archi di peso minimo

Esercizio 23.1-7 [Cormen]

Dimostrare che se tutti gli archi hanno peso strettamente positivo in $G = (V, E)$*, allora qualsiasi sottoinsieme di archi che rende connessi i nodi di* V *di peso minimo è un MST. Non è vero se ci sono archi di peso negativo.*

Soluzione: Supponiamo per assurdo che esista un insieme di archi F di peso minimo che connette V, ma che non è un albero. Allora necessariamente contiene un ciclo. Posso eliminare un arco e da F, mantenedo $(V, F \setminus \{e\})$ connesso. $w(F \setminus \{e\}) = w(F) - w(e) < w(F)$ contro la minimalità di F.

Ovviamente, con archi di peso negativo, posso aggiungere archi e diminuire il peso dell'insieme di archi. Con archi di peso nullo potrei avere sovrainsiemi di archi con lo stesso peso di sottoinsiemi.

Esercizi 3D(d)

Il caso dei costi negativi nel calcolo dei cammini minimi

Capire Bellman-Ford [1]

Esercizio 24.2-3 [Cormen]

Dato un grafo orientato pesato G=(V, E, w), senza cicli di peso negativo, sia m il massimo del numero minimo di archi di un cammino minimo da s a v, per ogni nodo v ∈V. Modificate l'algoritmo di Bellman-Ford in modo che termini in m+1 passi, anche se m non è noto a priori.

Soluzione: Questo esercizio ci dà l'opportunità di rivedere l'algoritmo di Bellman-Ford.

```
def bellmanFord(G, w, s):
    initializeSingleSource(G, s)
    for i=1 to n-1 do
        forall (u, v) ∈ G.E do relax(u, v, w)
    forall (u, v) ∈ G.E do
        if v.d > u.d + w(u, v) return FALSE
    return TRUE
```

```
def initialiseSingleSource(u, v, w):
    forall v ∈ G.V do
        v.d = ∞
        v.π = NIL
    s.d = 0
```

```
def relax(u, v, w):
    if (v.d > u.d + w(u→v))
        v.d = u.d + w(u→v)
        v.π = u
```

Capire Bellman-Ford [2]

Proposizione: *Alla k-esima iterazione del ciclo principale di Bellman-Ford ha stabilizzato al suo valore finale l'attributo v.d per tutti i nodi tali che il cammino minimo $s \rightsquigarrow v$ ha al più k archi.*

Dimostrazione: Induzione su k. Per k=0, la proposizione è banalmente vera, perché la procedura *initializeSingleSource* inizializza a 0 l'attributo $s.d$ ed s è l'unico nodo a distanza 0 (sia in peso che in numero di archi).

Se k>0, possiamo assumere per ipotesi induttiva che l'algoritmo abbia stabilizzato tutti i nodi il cui cammino minimo è lungo k-1 o meno passi.

Se v è un nodo tale che il cammino minimo $s \rightsquigarrow v$ è lungo k, significa che tale cammino ha la forma $s \rightsquigarrow u \rightarrow v$ minimo e $s \rightsquigarrow u$ minimo lungo k-1 passi (i cammini minimi godono della proprietà di sottostruttura ottima).

Quindi v viene stabilizzato alla k-esima iterazione da *relax*(u, v, w). □

Capire Bellman-Ford [3]

Proposizione: *Se un'iterazione di relax non modifica nessun attributo u.d, allora u.d è il valore del cammino minimo* $s \rightsquigarrow u$.

Dimostrazione: La funzione *relax* (invocata su tutti gli archi ad ogni iterazione) fa esattamente gli stessi calcoli se gli attributi *u.d* non cambiano. Ho raggiunto un **punto fisso**.□

```
def relax(u, v, w):
    fix = TRUE
    if (v.d > u.d + w(u→v))
        v.d = u.d + w(u→v)
        v.π = u
        fix = FALSE
    return fix
```

```
def bellmanFord(G, w, s):
    initializeSingleSource(G, s)
    for i=1 to n do
        fix = TRUE
        forall (u, v) ∈ G.E do
            fix = fix and relax(u, v, w)
        if fix return TRUE
    if !fix return FALSE
```

Se non si è stabilizzata a un punto fisso dopo n giri sono sicuro che c'è un ciclo negativo

Capire Bellman-Ford [4]

Esercizio 24.2-4 [Cormen]

Modificate l'algoritmo di Bellman-Ford in modo v.d sia posto a -∞ su tutti i vertici v raggiungibili da s attraverso un cammino che comprende un ciclo di peso negativo.

Soluzione: La seconda passata dell'algoritmo di Bellman-Ford 'scopre' l'esistenza di cicli negativi, perché ci sono nodi la cui distanza non si è stabilizzata dopo n-1 cicli. In realtà, in questo caso, il **loro costo non si stabilizzerebbe mai**.

Siccome anche tutti i cicli semplici sono al più lunghi n-1, dopo n-1 iterazioni del **for** che verifica del check di Bellman-Ford, sono sicuro di aver marcato tutti questi nodi modificandolo come segue:

```
def bellmanFord(G, w, s):
    …
    for i=1 to n-1 do
        forall (u, v) ∈ G.E do
            if v.d > u.d + w(u, v) then v.d = -∞
```

Ovviamente per tutto il primo ciclo potete avere il dubbio che i costi decrescano per effetto di cammini semplici

Capire Bellman-Ford [5]

Personalmente non sono convinto che questa soluzione del Cormen sia corretta. Voi? Dimostratene la correttezza o trovate un controesempio. Se funziona, confrontate la complessità con il mio.

> Modify the Bellman-Ford algorithm so that it sets $v.d$ to $-\infty$ for all vertices v for which there is a negative-weight cycle on some path from the source to v.

```
1  BELLMAN-FORD'(G, w, s)
2      INITIALIZE-SINGLE-SOURCE(G, s)
3      for i = 1 to |G.V| - 1
4          for each edge (u, v) ∈ G.E
5              RELAX(u, v, w)
6      for each edge(u, v) ∈ G.E
7          if v.d > u.d + w(u, v)
8              mark v
9      for each vertex v ∈ marked vertices
10         FOLLOW-AND-MARK-PRED(v)
```

```
1  FOLLOW-AND-MARK-PRED(v)
2      if v != NIL and v.d != -∞
3          v.d = -∞
4          FOLLOW-AND-MARK-PRED(v.π)
5      else
6          return
```

After running $\text{BELLMAN-FORD}'$, run DFS with all vertices on negative-weight cycles as source vertices. All the vertices that can be reached from these vertices should have their d attributes set to $-\infty$.

Capire Bellman-Ford [1]

Esercizio 24.2-6 [Cormen]

Dare un algoritmo che elenca i vertici in un ciclo di peso negativo se c'è.

Soluzione: L'algoritmo precedente marca tutti i vertici che stanno su un ciclo di costo negativo. Partendo da uno di essi ed eseguendo una DFS si trova facilmente un ciclo (**percorrendo solo nodi marcati -∞**).

I nodi marcati -∞ potrebbero essere in un ciclo di peso negativo oppure a valle di un ciclo negativo e quindi occorre fare la DFS sul grafo trasposto (con archi invertiti) oppure semplicemente sul vettore dei padri prodotto dall'algoritmo di Bellman-Ford modificato.

Lezione 3D

That's all Folks…

…Domande?