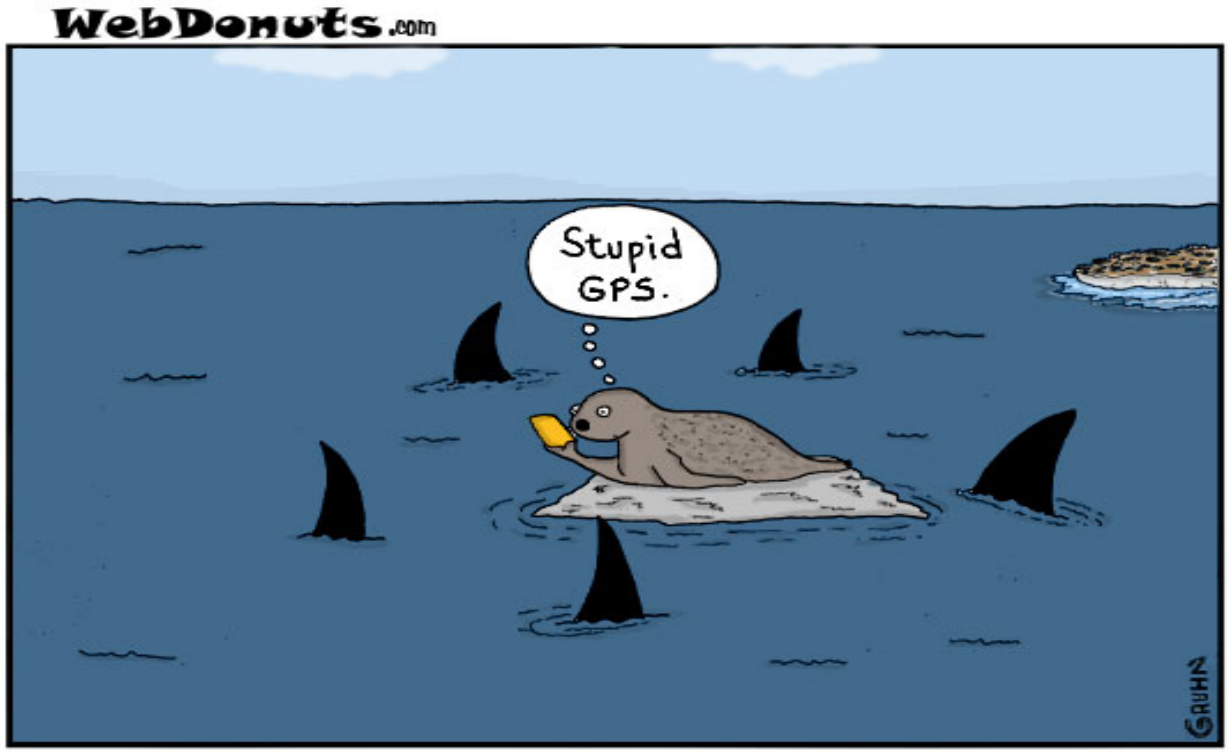
Algoritmi e strutture dati

Grafi e percorsi minimi con sorgente singola



Menú di questa lezione

In questa lezione introduciamo il concetto di percorso minimo su un grafo pesato e diretto, e vediamo alcune soluzioni al problema di calcolarlo, partendo da una singola sorgente.

Percorsi minimi con sorgente singola

Ci concentriamo adesso su grafi diretti, pesati, e connessi. Dato $G = (V, E, W)$, e dato un vertice s, ci proponiamo di trovare per ogni $v \in V$ il percorso di peso minimo che porta da s a v. Questo problema, chiamato **percorsi minimi con sorgente singola** è da considerarsi la generalizzazione sia di *BreadthFirstSearch* che degli algoritmi per la ricerca dell'albero di copertura minimo. Nel primo caso da grafi non pesati a pesati e nel secondo da grafi indiretti a diretti.

Percorsi minimi con sorgente singola

Come abbiamo fatto nel caso dei grafi non pesati, anche nel caso dei grafi pesati possiamo definire il concetto di distanza minima tra due vertici. Immaginiamo di individuare un vertice speciale s (la sorgente, come nella visita in ampiezza), e diciamo che $\delta(s, v)$ è la **distanza più corta** tra s e v, e la definiamo come segue. Prima diciamo che dato un certo percorso $p = v_0, v_1, \ldots, v_k$ (nella nostra notazione: $v_0 \rightsquigarrow v_k$), il suo **peso** è $w(p) = \Sigma_1^k W(v_{i-1}, v_i)$, e poi diciamo che:

$$\delta(s, v) = \begin{cases} min\ w(p) & p = s, \ldots, v \\ \infty & \text{se } v \text{ non si raggiunge da } s \end{cases}$$

La distanza piu corta, definita cosi, gode delle stesse proprietà di quella definita nel caso non pesato, e vale, inoltre, la sua sottostruttura ottima, come vediamo subito.

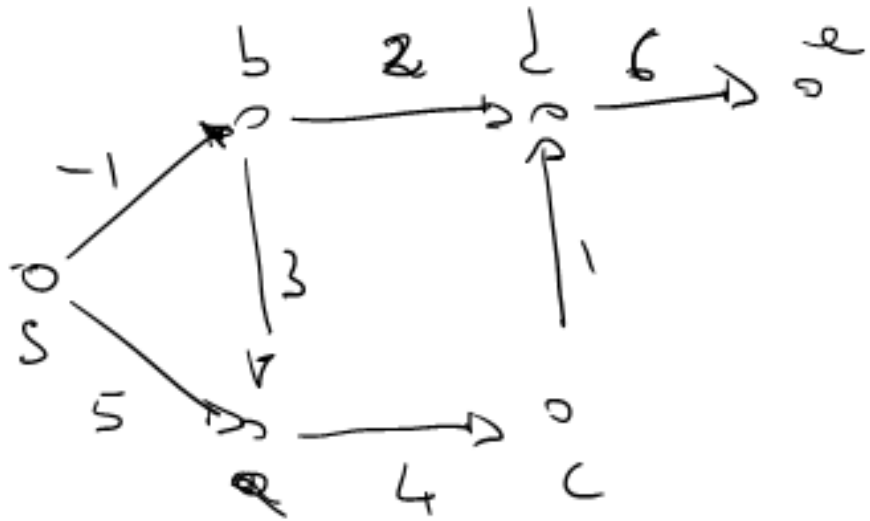
Percorsi minimi con sorgente singola

Un percorso di peso minimo da s a v, denotato con $s = v_0, v_1, \ldots, v_k = v$ ha, come abbiamo detto, una **sotto struttura ottima**. E' facile vedere che per ogni i, j, anche $v_i, \ldots, v_j$ è un percorso di peso minimo da v_i a v_j (per vederlo, si ragiona per contraddizione sull'ipotesi che $v_0, \ldots, v_k$ fosse di peso minimo. Cosa accade quando esistono pesi negativi? Osserviamo prima di tutto che dobbiamo distinguere la situazione in cui i pesi negativi rendono la definizione di percorso minimo priva di senso, e quelle in cui il percorso minimo esiste ancora ma un particolare algoritmo non è in grado di gestirli. Se G ha pesi negativi, ma non ci sono **cicli negativi** raggiungibili da s, allora i percorsi minimi da s sono ancora ben definiti. D'altra parte, se da s c'é un ciclo negativo raggiungibile, allora non ha senso definire un percorso di peso minimo da s: ogni nuovo passaggio attraverso quel ciclo diminuirebbe il peso. In questo caso diciamo che il peso del percorso da s a v, $\delta(s, v)$ è $-\infty$. In maniera simile, se v semplicemente non è raggiungibile da s, diremo che il peso del percorso minimo è ∞.

Percorsi minimi con sorgente singola

Ma che ruolo hanno i cicli (negativi o no) sui percorsi minimi? Se $s \rightsquigarrow v$ è un percorso minimo da s a v che contiene un ciclo positivo, allora eliminando il ciclo si ottiene un nuovo percorso minimo; pertanto $s \rightsquigarrow v$ non contiene un ciclo positivo. D'altra parte, se il ciclo fosse negativo, abbiamo giá osservato che il peso del percorso minimo sarebbe $-\infty$. Se, infine, il ciclo ha peso 0, allora eliminandolo si ottiene un nuovo percorso minimo dello stesso peso. Pertanto, indipendentemente dalla presenza di pesi negativi, possiamo sempre dire che un percorso minimo è aciclico, e quindi ci limitiamo a studiare percorsi minimi di lunghezza $|V| - 1$ al massimo.

ESEMPI DI GRAFI DIRETTI PESATI

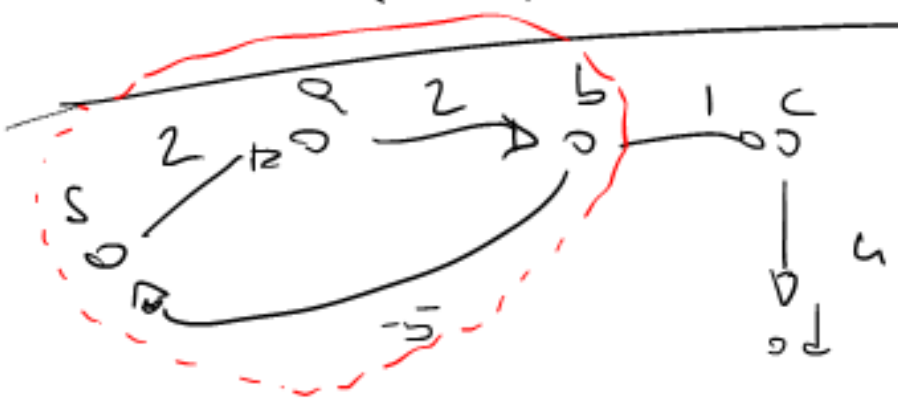


$f(s,b) = -1$

$f(s,a) = 2$

$f(s,c) = 6$

$f(c,a) = \infty$



CICLO NEGATIVO

$f(s,x) = -\infty \quad \forall x$

CICLO POSITIVO

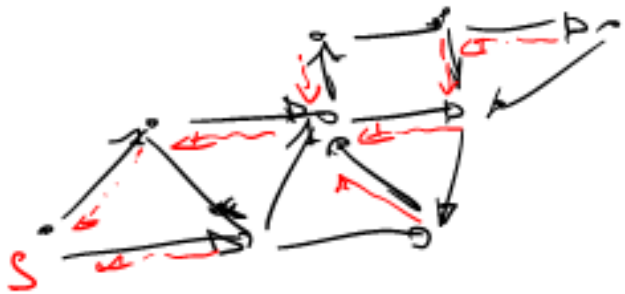
$f(s,c) = 4$

$f(s,b) = 5$

$f(s,e) = 1$

Percorsi minimi con sorgente singola

Come nel caso delle visite, interpretiamo l'output di un qualsiasi algoritmo che dati G (diretto, pesato, che non contiene cicli negativi), ed $s \in G.V$, come un assegnamento del valore $v.\pi$ per ogni $v \in V \setminus \{s\}$. Cerchiamo, quindi, l'**albero dei cammini di peso minimo** in G (questa volta si tratta di un albero radicato diretto), che non è necessariamente unico, e tale che ogni (prefisso di ogni) ramo rappresenta un cammino di peso minimo da s al vertice in questione. In generale gli algoritmi per il calcolo di questo albero utilizzano una tecnica chiamata **rilassamento**, che funziona sulla base tanto di $v.\pi$ come di $v.d$, rappresentando, quest'ultimo, il peso del cammino minimo **che si conosce fino ad un determinato momento** da s a v.



Percorsi minimi con sorgente singola

Il valore $v.d$ deve essere visto pertanto come **una stima** del peso del cammino minimo da s a v. Come tale, deve essere inizializzata.

proc ***InizializeSingleSource*** (G, s)

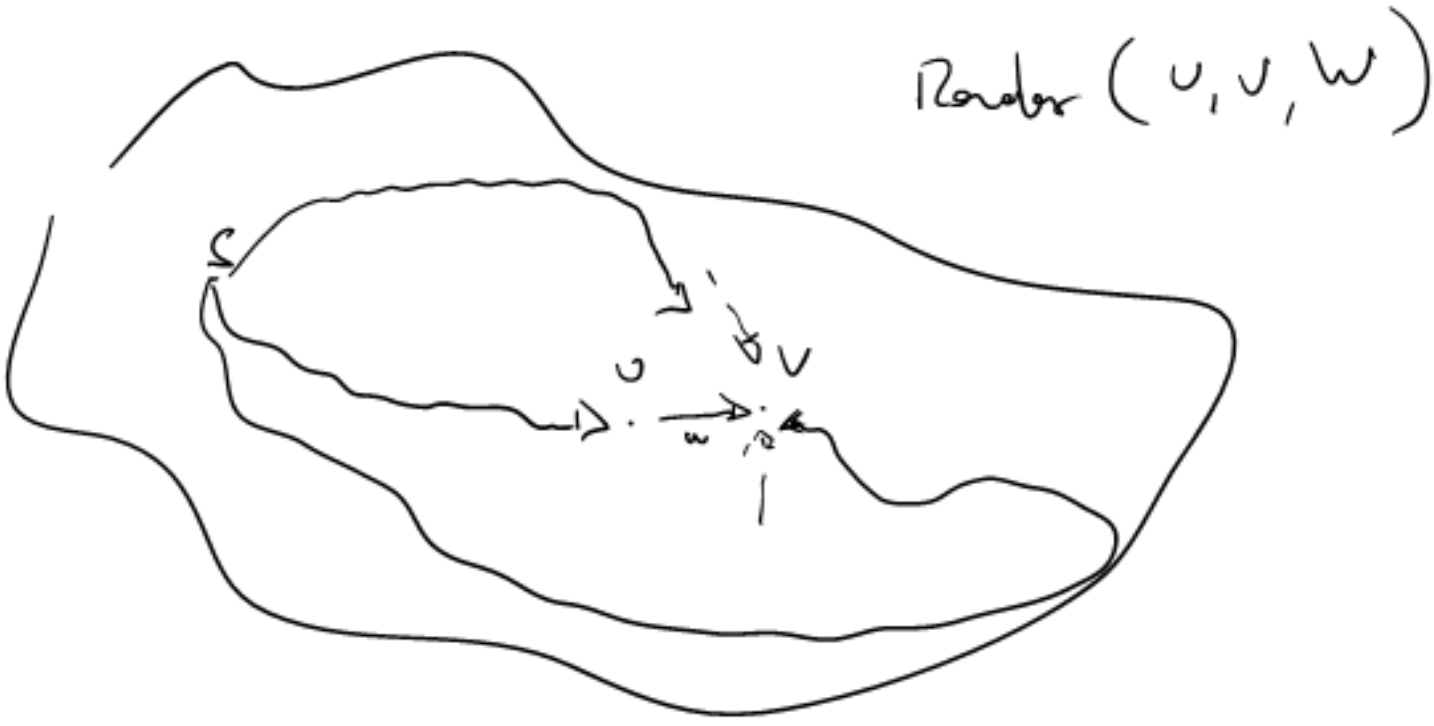
- **for** $(v \in G.V)$
  - $v.d = \infty$
  - $v.\pi = Nil$
- $s.d = 0$

Percorsi minimi con sorgente singola

Rilassare un arco (u, v) consiste nel testare se la stima su $v.d$ puó essere migliorata grazie all'arco (u, v). In questo caso si aggiorna tanto $v.d$ come $v.\pi$. Assumiamo quindi che (u, v) esista e vediamo il codice di *Relax*. Osserviamo che ogni algoritmo per il calcolo dell'albero dei cammini minimi che vediamo si basa sull'esecuzione come passo previo di *InizializeSingleSource*, e sul fatto che ogni modifica ai pesi dei cammini e la loro struttura è fatta esclusivamente attraverso *Relax*.

```
proc Relax (u, v, W)
  if (v.d > u.d + W(u, v))
    then
      v.d = u.d + W(u, v)
      v.π = u
```

Esempio di Rilassamento



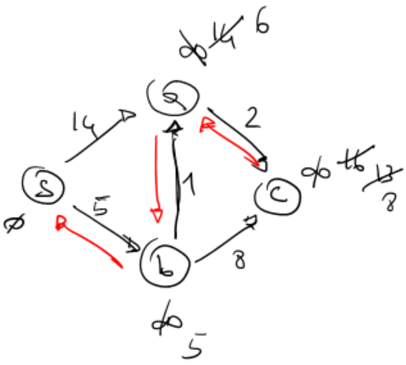
Algoritmo di Bellman-Ford

Il primo algoritmo che vediamo, il piú generale, noto come algoritmo di Bellman-Ford, lavora con pesi negativi e con cicli negativi: restituisce i cammini minimi ed i loro pesi se non ci sono cicli negativi raggiungibili dalla sorgente, e **false** altrimenti.

Algoritmo di Bellman-Ford

```
proc Bellman-Ford (G, s)
  InizializeSingleSource(G, s)
  for i = 1 to |G.V| − 1
    {for ((u, v) ∈ G.E) Relax(u, v, W)
  for ((u, v) ∈ G.E)
    {if (v.d > u.d + W(u, v))
       then return false
  return true
```

ESSMA 2b 1138



(s,a)
(s,b)
(a,c)
(b,a)
(b,c)

13

I II III

Correttezza e complessità di Bellman-Ford

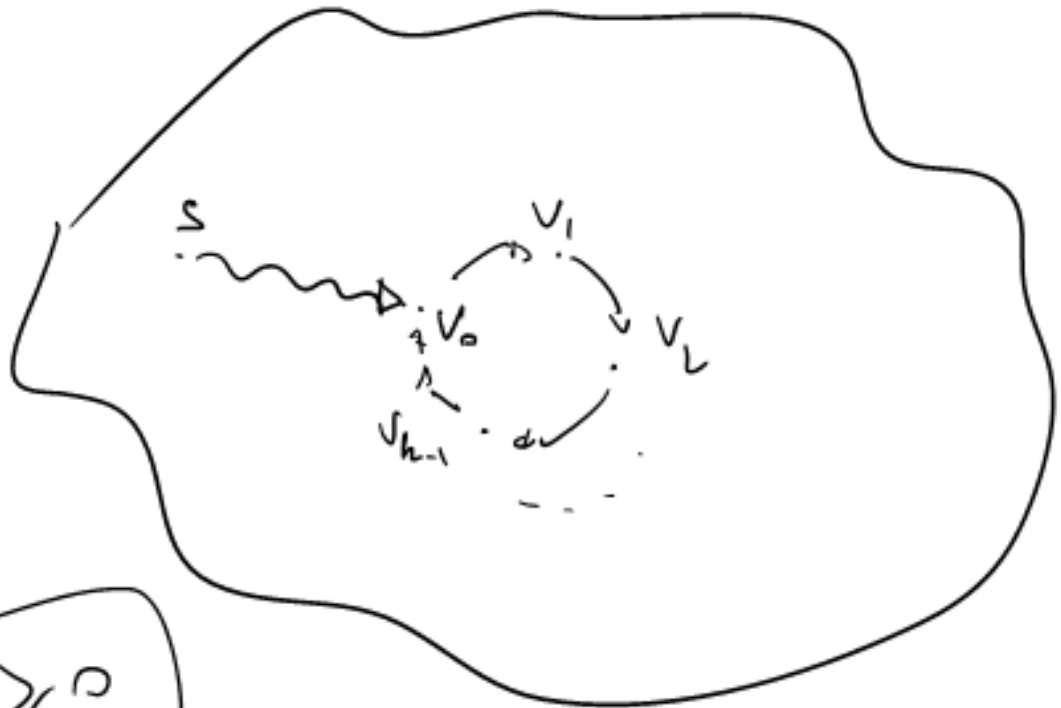
La **complessitá** di *Bellman-Ford* è, evidentemente, $\Theta(|V| \cdot |E|)$, cioè $O(|V|^3)$ ($\Theta|V|^3$ nel caso di grafi densi), e la **terminazione** non presenta particolari difficoltà. Per dimostrare la **correttezza**, invece, dobbiamo fare qualche passo in piú. Sia G un grafo diretto pesato, s uno dei suoi vertici, e assumiamo che *Bellman-Ford* sia stato eseguito su G con sorgente s. Vogliamo mostrare che se G non contiene cicli negativi, allora per ogni v raggiungibile da s si ha che $v.d = \delta(s, v)$, il grafo G_π (che corrisponde ad un albero a radice s) è precisamente l'albero dei cammini minimi, e l'algoritmo ritorna **true**; inoltre, se G contiene un ciclo negativo raggiungibile da s allora l'algoritmo ritorna **false**.

Correttezza e complessità di Bellman-Ford

Procedamo per casi, e assumiamo prima che G non contenga cicli negativi raggiungibili dalla sorgente. Pertanto, per un determinato vertice v, succede che il cammmino minimo $s = v_0, v_1, v_2, \ldots, v_k = v$ è definito. Vogliamo mostrare che, dopo aver eseguito l'algoritmo, $v.d = \delta(s, v)$. Mostriamo la seguente **invariante**: dopo i esecuzioni del ciclo piú esterno, $v_i.d = \delta(s, v_i)$.

- Il **caso base** è ovvio: dopo 0 iterazioni, $v_0 = s$ è tale che $v_0.d = 0 = \delta(s, v_0)$ (conseguenza di *InizializeSingleSource*).
- Supponiamo adesso (**caso induttivo**) che dopo $i - 1$ iterazioni si abbia $v_{i-1}.d = \delta(s, v_{i-1})$ (ipotesi induttiva). Poichè nell'iterazione i-esima si rilassano **tutti gli archi**, tra questi si rilasserà anche l'arco (v_{i-1}, v_i); dopo questo rilassamento, chiaramente, $v_i.d = \delta(s, v_i)$ (perchè l'arco (v_{i-1}, v_i) è parte di un percorso minimo). Inoltre, in ogni rilassamento successivo di qualunque arco, compreso (v_{i-1}, v_i), il valore di $v_i.d$ non puó cambiare perchè è già minimo.

Bellman-Ford con cicli negativi



v_0
=
$v_0, v_1, v_2, \ldots, v_{k-1}, v_k$

ciclo negativo

Se s raggiunge v_0 e v_0 raggiunge v_0 allora

T_1	T_2	
$v_0.d$	$v_1.d$	$+ w(v_0, v_1)$
$v_1.d$	$v_2.d$	$+ w(v_1, v_2)$
$v_2.d$	$v_3.d$	$+ w(v_2, v_3)$
	⋮	
$v_{k-1}.d$	$v_k.d$	$+ w(v_{k-1}, v_k)$

$T_1 = T_2$

T

$T' \leq T +$ peso del ciclo

$c < 0$ peso

Algoritmo di Bellman-Ford

Supponiamo adesso che G contenga un ciclo negativo raggiungibile dalla sorgente: $v_0, v_1, \ldots, v_k = v_0$, tale che:

$$\Sigma_{i=1}^{k} W(v_{i-1}, v_i) < 0.$$

Se, per assurdo, *Bellman-Ford* restituisse **true**, allora per $i = 1, 2, \ldots k$ succederebbe che $v_i.d \leq v_{i-1}.d + W(v_{i-1}, v_i)$. Sommando:

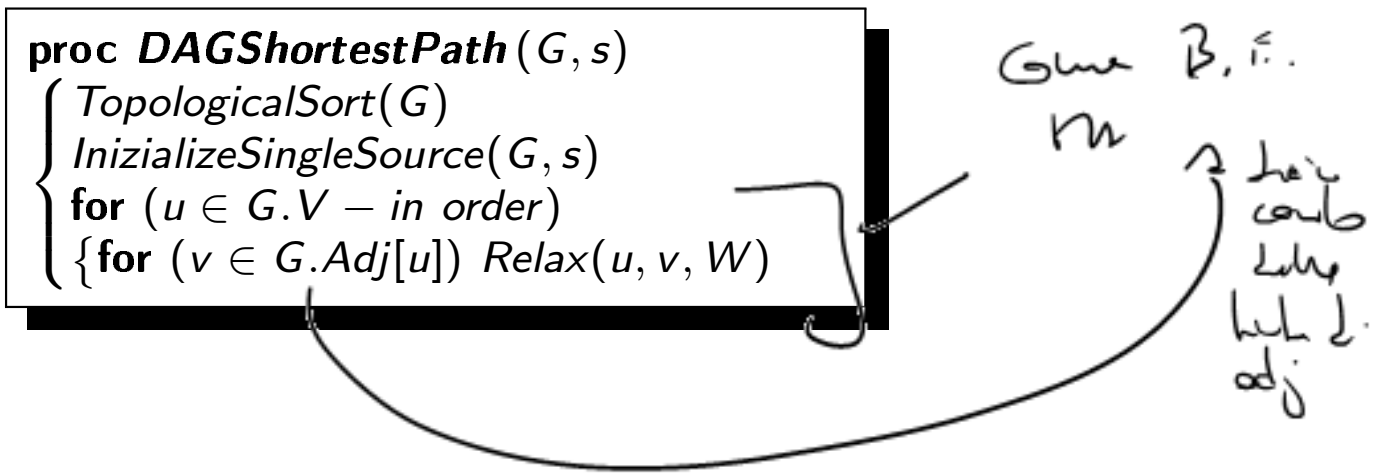
$$\begin{array}{lll} \Sigma_{i=1}^{k} v_i.d & \leq \Sigma_{i=1}^{k} (v_{i-1}.d + W(v_{i-1}, v_i)) & \text{somma nel ciclo} \\ & = \Sigma_{i=1}^{k} v_{i-1}.d + \Sigma_{i=1}^{k} W(v_{i-1}, v_i) & \text{proprietá di } \Sigma \\ 0 & \leq \Sigma_{i=1}^{k} W(v_{i-1}, v_i) & \end{array}$$

perché $\Sigma_{i=1}^{k} v_{i-1}.d = \Sigma_{i=1}^{k} v_i.d$. Questo contraddice l'ipotesi assurda. Quindi, *Bellman-Ford* restituisce **false**.

Percorsi minimi in grafi aciclici

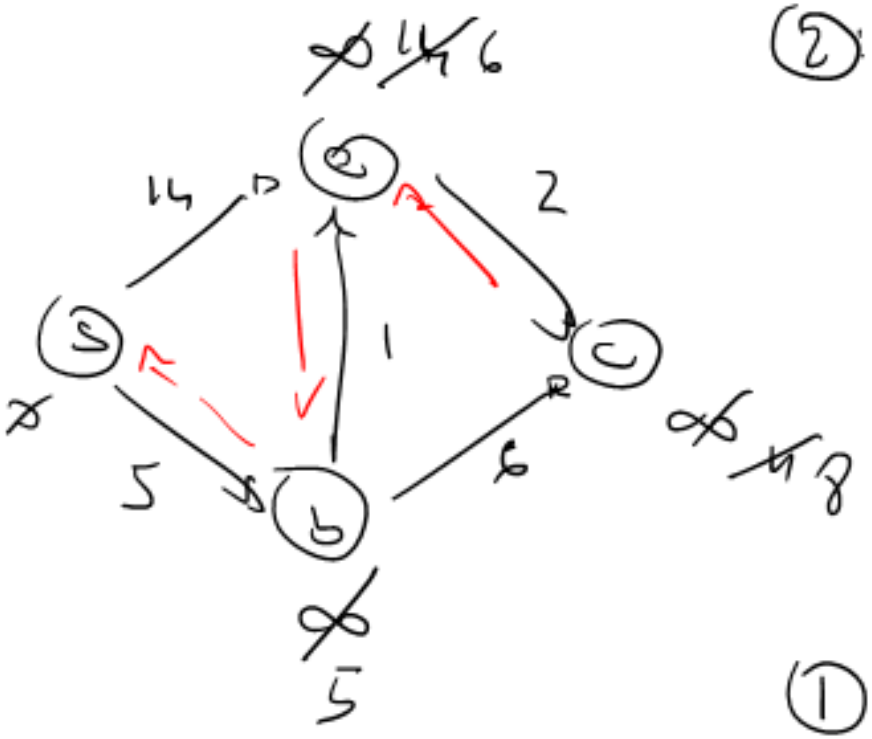
Nel caso particolare in cui possiamo assumere che G sia aciclico, trovare i percorsi minimi da una sorgente s, anche in presenza di pesi negativi, è piú semplice. Infatti un grafo aciclico puó essere ordinato topologicamente: osservando che se u precede v in un percorso minimo, allora u precede v nell'ordine topologico. Quindi possiamo operare un ordinamento topologico prima, e poi una serie di rilassamenti in ordine, risparmiando quindi molti passi di rilassamento. L'algoritmo, che noi chiamiamo *DAGShortestPath*, è in realtà *Bellman-Ford* ottimizzato per il caso aciclico.

Percorsi minimi in grafi aciclici



Esercizio 135

C



12 cammini

② Dag Sh Path:

O.T: s, b, e, c

(5 iterazioni)

		I	II	III
①	(b, v) s.t	∞	∞	
	(e, v) e.t	14	6	
	(b, e) b.t	5	5	
	(s, e) c.t	∞	16	8
	(c, b)			

Correttezza e complessità di *DAGShortestPath*

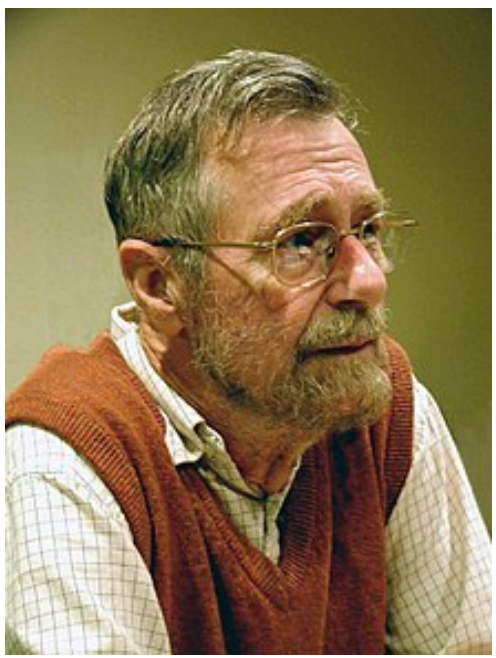
Con analisi aggragata, vediamo facilmente che la **complessitá** di *DAGShortestPath* è $\Theta(|V| + |E|)$, e la sua **terminazione** è ovvia. Dimostriamo adesso la sua **correttezza**. L'**invariante** che usiamo è: assumendo che v sia raggiungibile da s, per un percorso minimo qualsiasi $s = v_0, v_1, \ldots, v_k = v$, dopo l'$i$-esima esecuzione del ciclo, si ha che $v_i.d = \delta(s, v_i)$.

- Nel **caso base**, $s = v$, $v.d = 0$ (*InizializeSingleSource*), e poiché s non puó essere raggiungibile da se stesso, nessun rilassamento modifica $s.d$, che rimane 0.
- Consideriamo quindi un percorso di lunghezza superiore a 0 da s a v (**caso induttivo**): $s = v_0, v_1, \ldots, v_k = v$. Giacché i vertici vengono considerati in ordine topologico, gli archi $(v_0, v_1), (v_1, v_2), \ldots$ vengono considerati precisamente in questo ordine. Quindi se è vero per ipotesi induttiva che $v_j.d = \delta(s, v_j)$, quando viene rilassato l'arco (v_j, v_{j+1}) succede che $v_{j+1}.d$ viene aggiornato al valore corretto $\delta(s, v_{j+1})$.

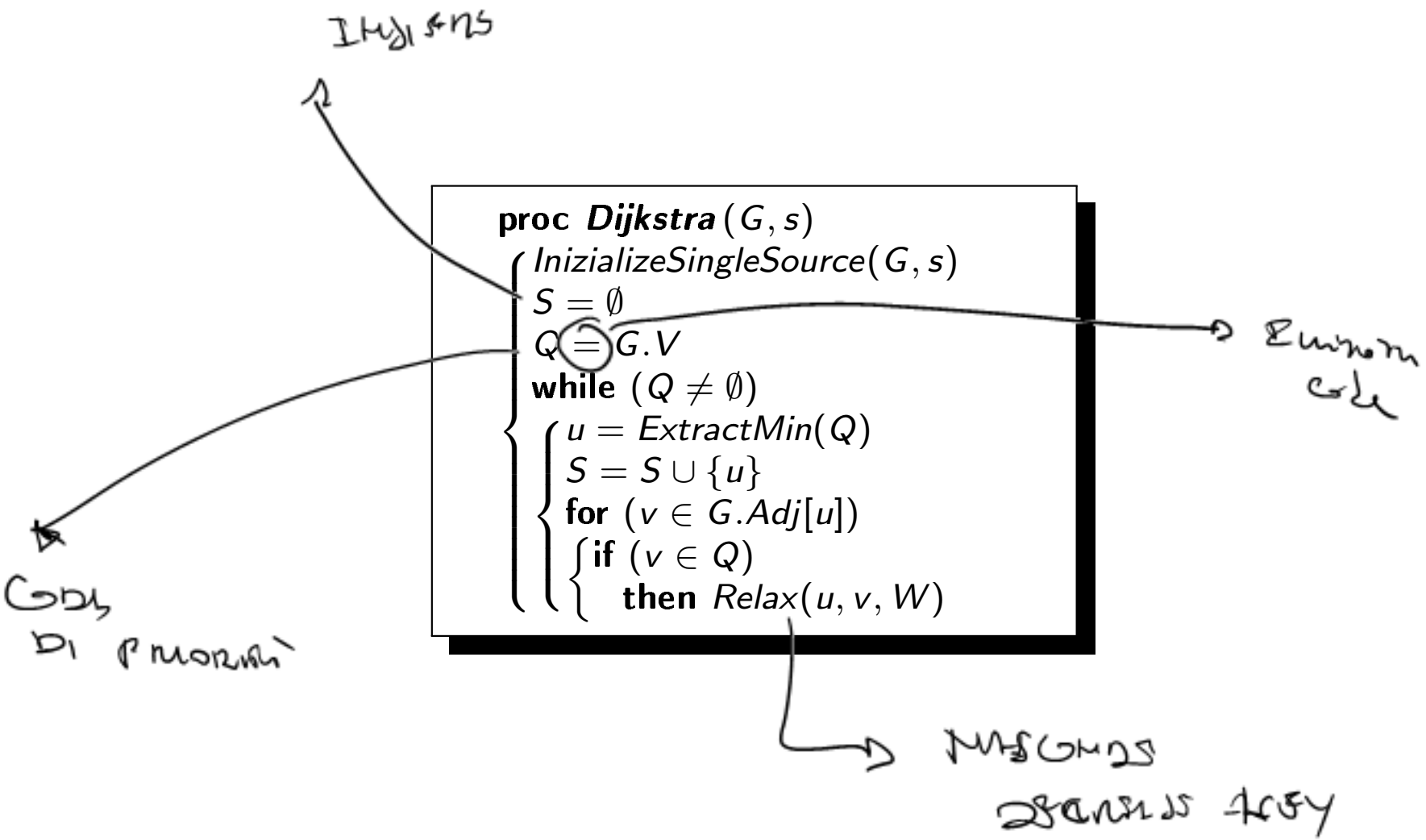
Che G_π sia, anche in questo caso, l'albero dei percorsi minimi a fine esecuzione, diventa ovvio.

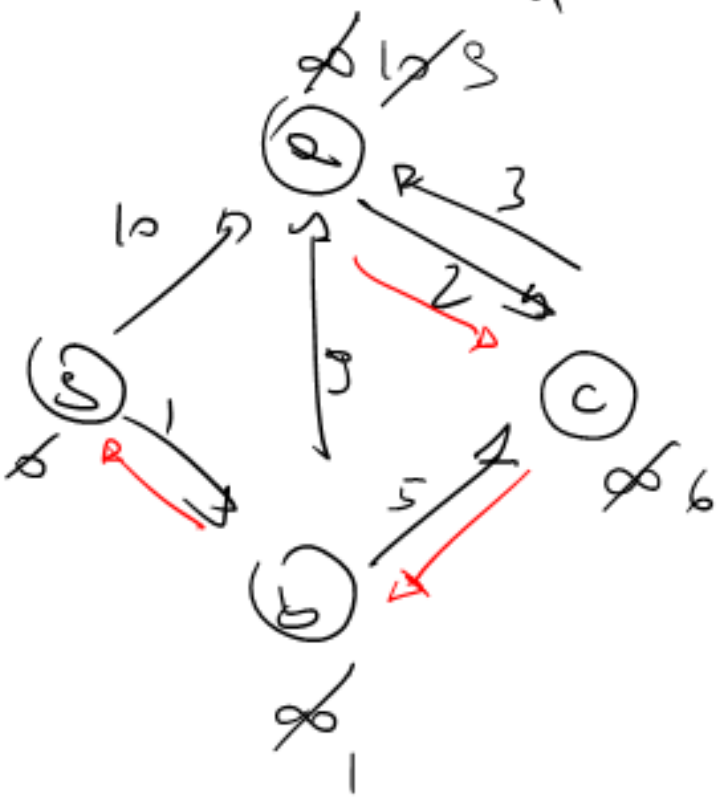
Algoritmo di Dijkstra

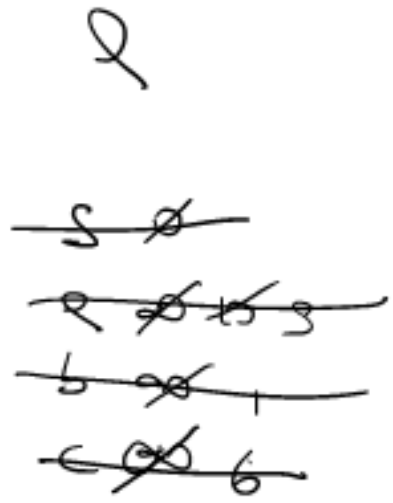
Possiamo migliorare le prestazioni di *Bellman-Ford* se aggiungiamo l'ipotesi che tutti gli archi hanno peso positivo o zero? L'algoritmo di Dijkstra risponde positivamente a questa domanda, utilizzando una tecnica che ricorda molto l'algoritmo di Prim (erditando pertanto le sue qualità e i suoi difetti). Come in *MST-Prim*, abbiamo bisogno di una coda di prioritá, per la quale conosciamo diverse implementazioni. La chiave sará il valore $v.d$ di ogni vertice. Il famosissimo Edsger Wybe Dijkstra pubblico il suo algoritmo nel 1959.



Algoritmo di Dijkstra







B

Algoritmo di Dijkstra

Come per *MST-Prim*, l'operazione di cambio del valore di una chiave contenuta in *Relax* contiene, nascosta, una operazione di decremento di una chiave nella coda di priorità. Pertanto, si applica solo ai vertici ancora nella coda di prioritá. Dopo *InizializeSingleSource* il valore $s.d = 0$; pertanto, s si trova in cima alla coda di prioritá e sará estratto per primo (tutti gli altri hanno valore ∞). Inoltre, nessuno inserisce elementi in Q, e pertanto il ciclo **while** viene eseguito precisamente $|V|$ volte.

Connected by Digraph

Correttezza e complessità dell' algoritmo di Dijkstra

Dimostriamo che l'algoritmo è **corretto**. L'**invariante** è: all'inizio di ogni iterazione del ciclo **while**, $v.d = \delta(s, v)$ per ogni $v \in S$:

- **Caso base**: quando $S = \emptyset$, l'inviariante è trivialmente vera.
- In quanto al **caso induttivo**, ragioniamo per contraddizione. Sia v il primo vertice in S tale che $v.d \neq \delta(s, v)$. Chiaramente $v \neq s$ e $\delta(s, v) \neq \infty$. Sia x il primo vertice sul percorso minimo tra s e v che è già in S, e y il primo in $V \setminus S$. Per ipotesi, $x.d = \delta(s, x)$, e questo implica che $y.d = \delta(s, y)$. Siccome y è prima di v sul percorso minimo tra s e v, $\delta(s, y) \leq \delta(s, v)$ (perchè tutti gli archi sono non negativi), quindi $y.d = \delta(s, y) \leq \delta(s, v) \leq v.d$. Ma v è scelto prima di y, dunque $v.d \leq y.d$. Allora, $y.d = \delta(s, y) = \delta(s, v) = v.d$, che genera una contraddizione.

Correttezza e complessità dell' algoritmo di Dijkstra

In quanto alla **complessità**, esattamente come per *MST-Prim*, possiamo implementare la coda di priorità con un array senza ordine oppure con una heap binaria, e dobbiamo considerare separatamente il caso dei grafi densi e di quelli sparsi. Se il grafo è denso, usiamo una coda senza struttura, per ottenere: $\Theta(|V|)$ (inizializzazione), $\Theta(|V|)$ (costruzione della coda), $\Theta(|V|^2)$ (per le estrazioni del minimo, analisi aggregata), e $\Theta(|E|) = \Theta(|V|^2)$ (per i decrementi, analisi aggregata), per un totale di $\Theta(|V|^2)$. Se, invece, il grafo è sparso usiamo una heap binaria, e avremo: $\Theta(|V|)$ (inizializzazione), $\Theta(|V|)$ (costruzione della coda), $\Theta(|V| \cdot log(|V|))$ (per le estrazioni del minimo, analisi aggregata), e $\Theta(|E| \cdot log(|V|))$ (per i decrementi, analisi aggregata), per un totale di $\Theta(|E| \cdot log(|V|))$.

Conclusione

Questo conclude il nostro trattamento del problema dei percorsi minimi con una sola sorgente. Adesso ci domanderemo cosa succede quando vogliamo trattare lo stesso problema però tra tutte le coppie di vertici, nell'ultimo blocco di questo corso.