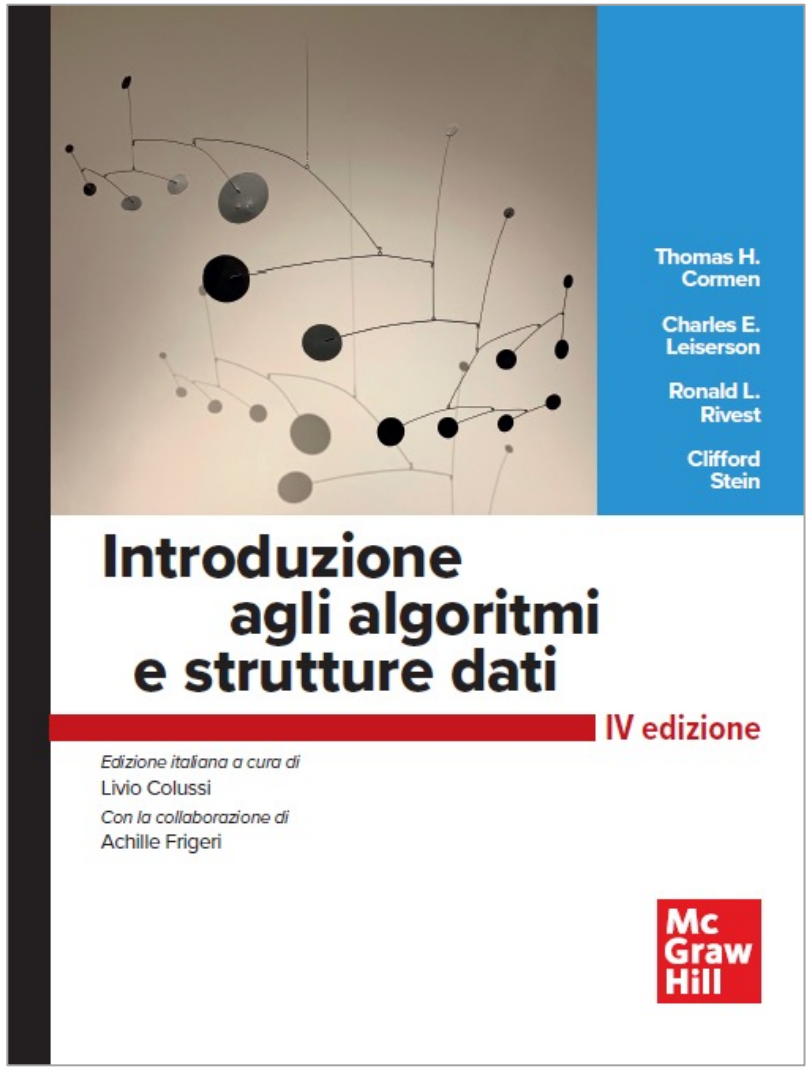


Capitolo 22

Cammini minimi da sorgente unica

Cammini minimi da sorgente unica

Sia $G = (V,E)$ un grafo orientato ai cui archi è associato un costo $W(u,v)$.

Il *costo di un cammino* $p = (v_0, v_1, \ldots, v_k)$ è la somma dei costi degli archi che lo costituiscono.

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

Il *costo di cammino minimo* da un vertice ***u*** ad un vertice ***v*** è definito nel seguente modo:

$$\delta(u,v) = \begin{cases} \min\{W(p)\} & \textbf{se esistono cammini } p \textbf{ da } u \textbf{ a } v \\ \infty & \textbf{altrimenti} \end{cases}$$

Un *cammino minimo* da ***u*** a ***v*** è un cammino ***p*** da ***u*** a ***v*** di costo $W(p) = \delta(u,v)$.

Nel *problema dei cammini minimi* viene appunto richiesto di calcolare i cammini minimi.

Ci sono quattro versioni del problema.

1) Cammini minimi da un'*unica sorgente* a tutti gli altri vertici.

2) Cammini minimi da ogni vertice ad un'*unica destinazione*.

3) Cammini minimi da un'*unica sorgente* ad un'*unica destinazione*.

4) Cammini minimi da ogni vertice ad ogni altro vertice.

Analizzeremo la prima istanza.

La seconda istanza si risolve simmetricamente.

La terza si può risolvere usando la soluzione della prima (non si conosce a tuttora alcun algoritmo che risolva la terza istanza in tempo asintoticamente migliore della prima).

La quarta si può risolvere usando la soluzione della prima per ogni vertice del grafo, ma in genere si può fare di meglio.

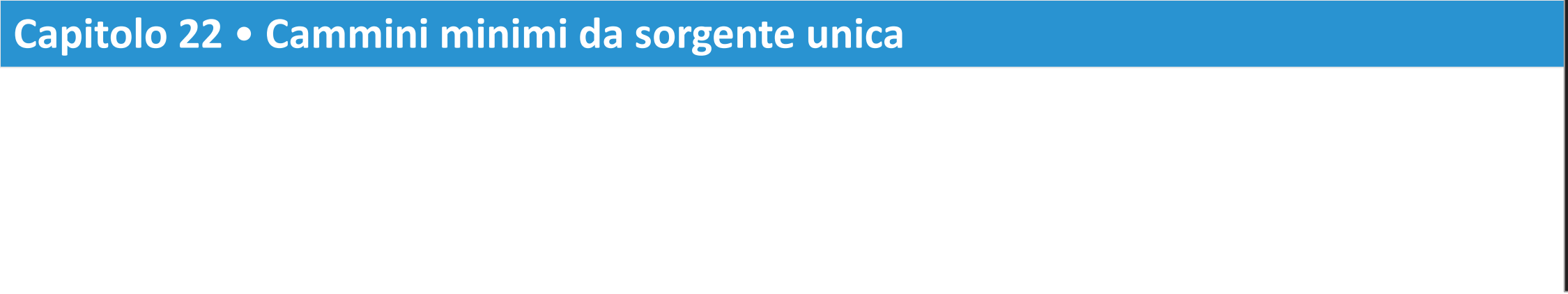
Cammini minimi da sorgente unica: archi di costo negativo

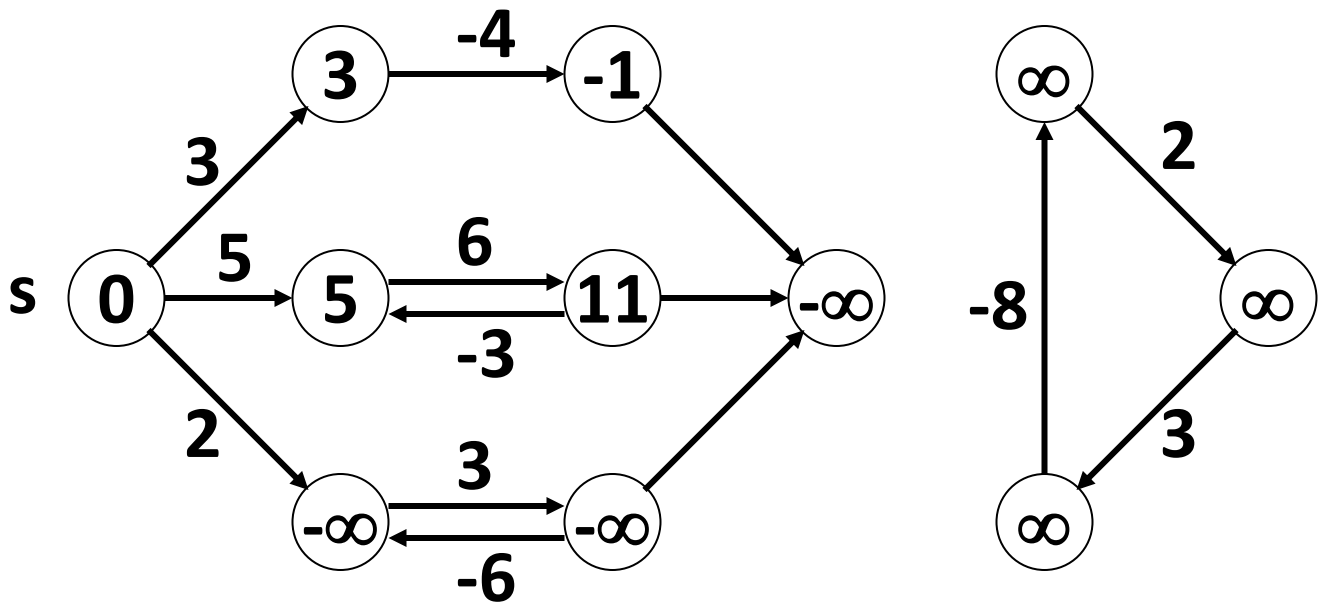
In alcuni casi il costo degli archi può anche essere negativo.

Questo non crea problemi nella ricerca dei cammini minimi da una sorgente ***s***, a meno che ci siano cicli di costo negativo raggiungibili da ***s***.

Se ***u*** è un vertice raggiungibile da ***s*** con un cammino ***p*** passante per un vertice ***v*** di un ciclo negativo, allora esistono dei cammini da ***s*** a ***u*** di costo sempre minore e il costo del cammino minimo $\delta(s,u)$ non è definito.

In questo caso poniamo $\delta(s,u) = -\infty$.





Cammini minimi: proprietà

Sottostruttura ottima dei cammini minimi. Se il cammino $\boldsymbol{p} = (\boldsymbol{v_0}, \boldsymbol{v_1}, \ldots, \boldsymbol{v_k})$ è minimo, allora sono minimi anche tutti i sottocammini $\boldsymbol{p_{ij}} = (\boldsymbol{v_i}, \ldots, \boldsymbol{v_j})$.

Dimostrazione. Se esistesse un cammino $\boldsymbol{q}$ da $\boldsymbol{v_i}$ a $\boldsymbol{v_j}$ di costo minore di $\boldsymbol{p_{ij}}$, allora sostituendo nel cammino $\boldsymbol{p}$ il sottocammino $\boldsymbol{p_{ij}}$ con il cammino $\boldsymbol{q}$ si otterrebbe un cammino da $\boldsymbol{v_0}$ a $\boldsymbol{v_k}$ di costo minore di $\boldsymbol{p}$. Impossibile se $\boldsymbol{p}$ è minimo.

Cammini minimi: rappresentazione

In genere ci interessa calcolare non solo i costi dei cammini minimi dalla sorgente ***s*** ad ogni vertice del grafo, ma anche i cammini minimi stessi.

Poiché i cammini minimi hanno sottostruttura ottima, possiamo rappresentarli arricchendo ogni vertice con un puntatore ***v.p*** che punta al vertice precedente in un cammino minimo da ***s*** a ***v***.

Cammini minimi: scomposizione dei costi

Scomposizione dei costi di cammino minimo. Se $\boldsymbol{p}$ è un cammino minimo da $\boldsymbol{s}$ ad un vertice $\boldsymbol{v}$ diverso da $\boldsymbol{s}$ ed $\boldsymbol{u}$ è il vertice che precede $\boldsymbol{v}$ nel cammino, allora

$$\delta(\boldsymbol{s},\boldsymbol{v}) = \delta(\boldsymbol{s},\boldsymbol{u}) + \boldsymbol{W}(\boldsymbol{u},\boldsymbol{v}).$$

Dimostrazione. Conseguenza della sottostruttura ottima: $\delta(\boldsymbol{s},\boldsymbol{v}) = \boldsymbol{W}(\boldsymbol{p}) = \delta(\boldsymbol{s},\boldsymbol{u}) + \boldsymbol{W}(\boldsymbol{u},\boldsymbol{v})$.

Cammini minimi: limite superiore

Limite superiore per i costi di cammino minimo. Per ogni arco uv vale la diseguaglianza:

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

Dimostrazione.

Se u non è raggiungibile da s allora $\delta(s,u) = \infty$ e $\delta(s,v) \leq \infty + W(u,v)$ banalmente.

Se u è raggiungibile da s allora $\delta(s,u) + W(u,v)$ è il costo di un cammino da s a v ed è quindi maggiore o uguale di $\delta(s,v)$.

Cammini minimi: rilassamento

Tecnica del rilassamento.

Gli algoritmi che studieremo per il problema dei cammini minimi usano la tecnica del rilassamento.

Aggiungiamo ad ogni vertice v del grafo un attributo $v.d$ che durante tutta l'esecuzione dell'algoritmo è un limite superiore per $\delta(s,v)$, mentre alla fine è proprio uguale a $\delta(s,v)$.

L'inizializzazione dei campi ***v.p*** e ***v.d*** è la stessa per tutti gli algoritmi.

Inizialize-Single-Source(*G*, *s*) *G* grafo pesato sugli archi
  for "ogni $v \in G.V$"
    $v.d = \infty$
    $v.p$ = nil
  $s.d = 0$

Il rilassamento rispetto ad un arco ***uv*** consiste nel controllare se è possibile migliorare il cammino finora trovato per ***v*** allungando il cammino trovato per ***u*** con l'arco ***uv***.

```
Relax(u, v, W)     // W funzione di peso
   if v.d > u.d + W(u,v)
      v.d = u.d + W(u,v)
      v.p = u
```

Rilassamento: proprietà

Effetto del rilassamento.

Dopo aver eseguito ***Relax*(*u*,*v*,*w*)**, vale la diseguaglianza

$$v.d \leq u.d + W(u,v)$$

Dimostrazione.

Se $v.d > u.d + W(u,v)$ prima del rilassamento viene posto $v.d = u.d + W(u,v)$.

Se $v.d \leq u.d + W(u,v)$ prima del rilassamento non viene fatto nulla e quindi è vero anche dopo.

Rilassamento: proprietà

Invariante del rilassamento.

Dopo l'inizializzazione, per ogni vertice v vale la disuguaglianza $v.d \geq \delta(s,v)$ che rimane vera anche dopo un numero qualsiasi di rilassamenti.

Inoltre, se a un certo punto $v.d = \delta(s,v)$, allora il suo valore non può più cambiare.

Dimostrazione.

Dopo l'inizializzazione si ha $s.d = 0 \geq \delta(s,s)$ e per ogni altro vertice $v.d = \infty \geq \delta(s,v)$.

Supponiamo che la disuguaglianza sia vera per ogni vertice prima di eseguire ***Relax*(*u*,*v*,*W*)**.

Se ***v.d*** non viene modificata, la disuguaglianza resta ovviamente vera.

Se ***v.d*** viene modificata, allora dopo aver eseguito il rilassamento si ha $v.d = u.d + W(u,v)$.

Poiché ***u.d*** non è stata modificata, vale la disuguaglianza $u.d \geq \delta(s,u)$, e quindi

$$v.d \geq \delta(s,u) + W(u,v) \geq \delta(s,v)$$

per il limite superiore dei costi di cammino minimo.

Infine, poiché il valore di $v.d$ può soltanto diminuire e $v.d \geq \delta(s,v)$ se $d[v] = \delta(s,v)$, allora il suo valore non può più cambiare.

*Correttezza di **v.d** per vertici non raggiungibili*.

Dopo l'inizializzazione per ogni vertice v non raggiungibile da s vale $v.d = \delta(s,v)$, e tale uguaglianza rimane vera anche dopo un numero qualsiasi di rilassamenti.

Dimostrazione. Dopo l'inizializzazione, $v.d = \infty$ per ogni vertice diverso da s.

Se v non è raggiungibile da s allora $\delta(s,v) = \infty = v.d$ e, per l'invariante del rilassamento, $v.d$ non può più cambiare.

Estensione della correttezza di ***v.d*** *per vertici raggiungibili*. Se ***uv*** è l'ultimo arco di un cammino minimo da ***s*** a ***v*** e $u.d = \delta(s,u)$ prima di eseguire il rilassamento dell'arco ***uv***, allora dopo il rilassamento $v.d = \delta(s,v)$.

Dimostrazione.

Dopo il rilassamento, $v.d \leq \delta(s,u) + w(u,v)$. Poiché ***uv*** è l'ultimo arco di un cammino minimo,

$\delta(s,v) = \delta(s,u) + w(u,v)$ e quindi $v.d \leq \delta(s,v)$.

Per l'invariante del rilassamento, $v.d = \delta(s,v)$.

Il *grafo dei predecessori* $G_{pred} = (V_{pred}, E_{pred})$ è il sottografo di $G = (V,E)$ con vertici

$$V_{pred} = \{s\} \cup \{v \in V : p[v] \neq \text{nil}\}$$

e archi

$$E_{pred} = \{ uv \in E : u = p[v]\}$$

Il grafo dei predecessori è un albero.

Se il grafo ***G*** non contiene cicli negativi raggiungibili dalla sorgente ***s***, allora dopo l'inizializzazione ed un numero arbitrario di rilassamenti, il grafo dei predecessori è un albero radicato in ***s***.

Dimostrazione. Dopo l'inizializzazione, il grafo dei predecessori contiene soltanto la radice ***s*** e quindi la cosa è vera.

Dobbiamo dimostrare che dopo un numero arbitrario di rilassamenti:

a) il grafo dei predecessori G_{pred} è aciclico;

b) in $G_{pred} = (V_{pred}, E_{pred})$ c'è uno ed un solo cammino da s ad ogni altro vertice $v \in V_{pred}$.

Supponiamo, per assurdo, che ad un certo punto il grafo G_{pred} contenga un ciclo $(v_0, v_1, \ldots, v_k = v_0)$.
Per ogni $i = 1, \ldots, k$, quando è stata effettuata l'ultima assegnazione $v_i.p = v_{i-1}$ è stato anche posto

$$v_i.d = dv_{i-1}.d + W(v_i, v_{i-1}).$$

Da allora, $v_i.d$ non è più cambiato mentre $v_{i-1}.d$ può soltanto essere diminuito.
Quindi, da allora in poi $v_i.d \geq v_{i-1}.d + W(v_i, v_{i-1})$.

Possiamo assumere che il ciclo si sia formato rilassando l'arco $v_{k-1}v_k$.
Prima di tale rilassamento $v_k.d > v_{k-1}.d + W(v_k, v_{k-1})$ per cui:

$$\sum_{i=1}^{k} v_i.d > \sum_{i=1}^{k} v_{i-1}.d + \sum_{i=1}^{k} W(v_{i-1}, v_i)$$

e quindi $0 > \sum_{i=1}^{k} W(v_{i-1}, v_i)$, assurdo perché non ci sono cicli negativi.

Supponiamo che ci sia uno ed un solo cammino da s ad ogni altro vertice $v \in V_{pred}$ prima di rilassare l'arco uv.

Se $v.p$ non viene modificato rimane vero anche dopo.

Se $v.p$ viene modificato allora prima del rilassamento $v.d > u.d + W(u,v)$.

Quindi $u.d < \infty$ e o $u = s$ oppure $u.p \neq$ **nil** e dunque esiste un unico cammino P da s a u.

Il rilassamento pone $v.p = u$ e quindi, dopo il rilassamento, uv è l'unico arco in E_{pred} entrante in v.

Poiché non si formano cicli, nessun cammino da s a u può contenere il vertice v.

Quindi anche dopo il rilassamento di uv, esiste l'unico cammino P da s a u.

Aggiungendo l'arco uv a P otteniamo quindi un unico cammino da s a v.

Cammini minimi. Se il grafo G non contiene cicli negativi raggiungibili dalla sorgente s e dopo aver eseguito l'inizializzazione ed un certo numero di rilassamenti vale $v.d = \delta(s,v) < \infty$ per un vertice v, allora $v \in V_{pred}$ e il cammino P da s a v in G_{pred} è un cammino minimo.

Dimostrazione.

Quando è stato assegnato un valore minore di ∞ a $v.d$ è stato anche assegnato un valore diverso da **nil** a $v.p$, e quindi $v \in V_{pred}$.

Sia $(s = u_0, u_1, \ldots, u_k = v)$ l'unico cammino da s a v in G_{pred}.

Allora $u_i.d \geq u_{i-1}.d + W(u_i, u_{i-1})$ per ogni i e

$$\sum_{i=1}^{k} u_i.d \geq \sum_{i=1}^{k} u_{i-1}.d + \sum_{i=1}^{k} W(u_{i-1}, u_i)$$

da cui

$$\sum_{i=1}^{k} W(u_{i-1}, u_i) \leq u_k.d - u_0.d = v.d - s.d = \delta(s, v)$$

Possiamo concludere che ogni algoritmo che esegua l'inizializzazione ed una sequenza di rilassamenti per cui alla fine $v.d = \delta(s,v)$ per ogni vertice v, calcola correttamente i cammini minimi.

Ci sono due algoritmi classici di questo tipo, uno dovuto a *Dijkstra* ed uno dovuto a *Bellman* e *Ford*.

L'algoritmo di Dijkstra richiede che i pesi degli archi non siano negativi, mentre quello di Bellman-Ford funziona anche nel caso generale.

Cammini minimi da sorgente unica: algoritmo di Bellman-Ford

L'algoritmo di Bellman-Ford risolve il problema dei cammini minimi da sorgente unica nel caso generale in cui i pesi degli archi possono essere negativi.

Dato un grafo orientato pesato ***G=(V,E)*** con vertice sorgente ***s*** e funzione peso ***W***, restituisce *false* se esiste un ciclo di peso negativo che è raggiungibile da ***s***.

Se un tale ciclo non esiste, l'algoritmo fornisce i cammini minimi e i loro pesi.

Bellman-Ford rilassa gli archi, riducendo progressivamente il valore stimato ***v.d*** per il peso di un cammino minimo dalla sorgente ***s*** a ciascun vertice ***v*** di ***V***, fino a raggiungere il peso effettivo $\delta(s,v)$ di un cammino minimo.

L'algoritmo restituisce *true* se e solo se il grafo non contiene cicli di peso negativo che sono raggiungibili dalla sorgente.


```
Bellman-Ford(G, W, s)
    Initialize-Single-Source(G, S)
    for i = 1 to |G.V| − 1
        for "ogni arco uv ∈ G.E"
            Relax(u, v, W)
for "ogni arco uv ∈ G.E"
    if v.d > u.d + W(u,v)
        return false
return true
```

Algoritmo di Bellman-Ford: complessità

Bellman-Ford richiede un tempo $O(V^2+VE)$ quando il grafo è rappresentato con liste delle adiacenze, perché l'inizializzazione richiede un tempo $\Theta(V)$, ciascuno dei $|V|-1$ passaggi sugli archi richiede un tempo $\Theta(E)$ (dovendo esaminare $|V|$ liste per trovare $|E|$ archi) e il primo ciclo for richiede un tempo $O(V+E)$.

Cammini minimi da sorgente unica: algoritmo di Dijkstra

L'algoritmo di Dijkstra risolve il problema dei cammini minimi da sorgente unica, ma richiede che tutti gli archi abbiano pesi non negativi.

Con una buona implementazione, il tempo di esecuzione di *Dijkstra* è minore di quello di *Bellman-Ford*.

Dijkstra generalizza la visita in ampiezza ai grafi pesati.
Un'onda si propaga dalla sorgente e non appena l'onda raggiunge un vertice, una nuova onda parte da tale vertice.
Mentre la visita in ampiezza opera come se ogni onda attraversasse un arco in un tempo unitario, in un grafo pesato il tempo richiesto dall'onda per attraversare un arco è dato dal peso dell'arco.
Poiché un cammino minimo in un grafo pesato può non avere il minor numero di archi, una semplice coda FIFO può non bastare per scegliere il nuovo vertice da cui far partire una nuova onda.

Dijkstra mantiene un insieme ***S*** di vertici i cui pesi finali dei cammini minimi dalla sorgente ***s*** sono stati già determinati.

L'algoritmo seleziona ripetutamente il vertice ***u*** di ***V-S*** con la stima minima del cammino minimo, aggiunge ***u*** a ***S*** e rilassa tutti gli archi che escono da ***u***.

Dijkstra sostituisce la coda FIFO della visita in ampiezza con una coda di min-priorità ***Q*** di vertici, utilizzando come chiavi i loro valori ***d***.


```
Dijkstra(G, W, s)
    Initialize-Single-Source(G, S)
    S = ∅, Q = ∅
    for "ogni vertice u ∈ G.V"
        Insert(Q, u)
        while Q ≠ ∅
            u = Extract-Min(Q)
            S = S ∪ {u}
            for "ogni vertice v ∈ G.Adj[u]"
                Relax(u, v, W)
                if la chiamata a Relax diminuisce v.d
                    Decrease-Key(Q, v, v.d)
```

Algoritmo di Dijkstra: complessità

Dijkstra mantiene la coda di min-priorità ***Q*** chiamando le tre operazioni *Insert*, *Extract-Min* e *Decrease-Key*.
Insert viene chiamata una sola volta per ogni vertice, come *Extract-Min*.
Ogni vertice viene aggiunto a ***S*** esattamente una volta, quindi ogni arco nella lista di adiacenza di ***u*** viene esaminato nel secondo ciclo for esattamente una volta durante l'esecuzione dell'algoritmo.

Poiché il numero totale di archi in tutte le liste di adiacenza è $|E|$, ci sono in totale $|E|$ iterazioni di questo ciclo for, e quindi in totale l'algoritmo chiama *Decrease-Key* al più $|E|$ volte.

Come per l'algoritmo di Prim, il tempo di esecuzione di *Dijkstra* dipende dalla implementazione della coda di min-priorità.

Sfruttando il fatto che i vertici sono numerati da ***1*** a $|V|$, memorizziamo ***v.d*** nell'elemento *v*-esimo di un array.

Insert e *Decrease-Key* richiedono un tempo (ammortizzato) $O(1)$ e *Extract-Min* un tempo $O(V)$ (perché deve ispezionare l'intero array), per un tempo totale $O(V^2 + E) = O(V^2)$.

Cammini minimi in grafi aciclici

Rilassando gli archi di un grafo aciclico in ordine topologico possiamo calcolare i cammini minimi in tempo $O(|V| + |E|)$.

DAG-Shortest-Path(*G, W, s*) \\ *G* grafo orientato aciclico

ordina in ordine topologico i vertici di G

Inizialize-Single-Source(*G*, *s*)

for "ogni vertice $u \in G.V$ preso in ordine topologico"

for "ogni vertice $v \in G.Adj[u]$"

Relax(*u, v, W*)

Complessità.

La complessità dell'algoritmo è $O(|V|+|E|)$.

Correttezza.

Sia v un vertice raggiungibile da s e sia

$$P = (s = u_0, u_1, \ldots, u_k = v)$$

un cammino minimo da s a v.

L'algoritmo rilassa gli archi del cammino nell'ordine $u_0u_1, u_1u_2, \ldots, u_{k-1}u_k$ e quindi alla fine $v.d = \delta(s,v)$.