

# Cammini minimi fra tutte le coppie

Alberto Montresor

17/04/2009

## 1 Introduzione

Problema dei cammini minimi fra tutte le coppie: ad esempio, quando si vuole creare la tabella con le distanze fra tutte le città di un atlante stradale.

Per risolvere questo problema, possiamo utilizzare le soluzioni per i cammini minimi da singola sorgente per ogni vertice del grafo.

- In caso di archi non negativi, possiamo utilizzare Dijkstra con un costo  $O((V^2 + VE) \log V)$  (con min-heap binario, il limite può essere semplificato come  $O(VE \log V)$ ) oppure con costo  $O(VE + V^2 \log V)$  utilizzando uno Heap di Fibonacci.
- In caso di archi negativi, possiamo utilizzare Bellman-Ford con un costo  $O(V^2E)$ , che corrisponde a  $V^4$  nel caso peggiore (grafi non sparsi).

E' possibile fare di meglio:

- L'algoritmo di Floyd e Warshall permette di risolvere il problema in tempo  $O(V^3)$  con archi negativi (ovviamente senza cicli negativi). Nota: nel Cormen si specifica che l'algoritmo funziona con matrice di adiacenza; sembra quasi che sia "necessaria"; l'idea è invece che alla fine si produrrà una matrice delle distanze che inizialmente è una matrice dei pesi degli archi. Ma il costo per trasformare una rappresentazione a lista in una rappresentazione a matrice è  $O(V^2)$
- Utilizzando una rappresentazione a liste di adiacenza, l'algoritmo di Johnson permette di risolvere il problema in  $O(VE + V^2 \log V)$ , che nel caso di grafi sparsi è migliore di  $O(V^3)$ .

## 2 Algoritmo di Floyd e Warshall

Questo algoritmo è basato su una tecnica di programmazione dinamica.

Tutte le definizioni e le discussioni che seguono assumono la presenza di un grafo  $G = (V, E)$  orientato con una funzione di costo  $w : E \rightarrow \mathbf{R}$ . Assumeremo inoltre che i vertici siano ordinati in qualche modo e li denoteremo  $\{v_1, v_2, \dots, v_n\}$ .

**Definizione 1.** Sia  $k$  un valore in  $\{0, \dots, n\}$ . Diciamo che un cammino  $p_{xy}^k$  è un cammino minimo  $k$ -vincolato fra  $x$  e  $y$  se esso ha il costo minimo fra tutti i cammini fra  $x$  e  $y$  che non passano per nessun vertice in  $v_{k+1}, \dots, v_n$  ( $x$  e  $y$  sono esclusi dal vincolo).

**Definizione 2.** Se esiste un cammino minimo  $k$ -vincolato  $p_{xy}^k$  fra  $x$  e  $y$ , definiamo  $\delta^k(x, y) = w(p_{xy}^k)$ ; nel caso tale cammino non esista,  $\delta^k(x, y) = +\infty$ .

E' immediato osservare che  $\delta^0(x, y) = w(x, y)$  se  $(x, y) \in E$ , altrimenti  $\delta^0(x, y) = +\infty$ .

**Teorema 1** (Sottostruttura ottima). Ogni sottocammino di un cammino minimo  $k$ -vincolato in  $G$  è esso stesso un cammino minimo  $k$ -vincolato in  $G$ .

*Dimostrazione.* Sia  $G_k$  il sottografo ottenuto da  $G$  rimuovendo tutti i vertici  $v_{k+1}, \dots, v_n$  diversi da  $x$  e  $y$ . Chiaramente, un cammino in  $G$  è minimo  $k$ -vincolato se e solo se è minimo in  $G_k$ . Il teorema segue quindi dalla sottostruttura ottima dei cammini minimi.  $\square$

Floyd, basandosi su un teorema precedente di Warshall, osservò che è possibile costruire  $\delta_{xy}^k$  in questo modo:

**Lemma 1** (Relazione tra distanze vincolate). *Per ogni  $k \in \{1, \dots, n\}$  e per ogni  $x, y \in V$ , si ha:*

$$\delta^k(x, y) = \min\{\delta^{k-1}(x, y), \delta^{k-1}(x, v_k) + \delta^{k-1}(v_k, y)\}$$

*Dimostrazione.* Sia  $p_{xy}^k$  un cammino minimo  $k$ -vincolato fra  $x$  e  $y$ . Ci sono due possibilità:

- se  $v_k$  non è un vertice interno di  $p_{xy}^k$ , allora questo cammino è anche  $k-1$ -vincolato; quindi

$$\delta^k(x, y) = \delta^{k-1}(x, y)$$

- se  $v_k$  è un vertice interno, allora il cammino può essere spezzato in due sottocammini  $p_{xv_k}^k$  e  $p_{v_k y}^k$ ; per la sottostruttura dei cammini minimi  $k$ -vincolati, sono anch'essi cammini minimi  $k$ -vincolati; inoltre, poiché entrambi i cammini non contengono  $v_k$  come vertice interno, sono anche cammini  $k-1$ -vincolati.

$$\delta^k(x, y) = \delta^{k-1}(x, v_k) + \delta^{k-1}(v_k, y)$$

Ovviamente, il valore inferiore fra i due è quello che corrisponde al cammino minimo.  $\square$

Grazie a questa relazione, possiamo utilizzare la programmazione dinamica per risolvere il problema. Notate che memoization non va bene in questo caso, perchè vogliamo calcolare la lunghezza di tutti i cammini.

## 2.1 Algoritmo

Per realizzare il cammino, abbiamo bisogno di una tabella  $d$  di dimensione  $|V| \times |V|$  per ogni coppia di vertici:  $d[x, y]$  restituisce il costo del cammino fra  $x$  e  $y$ .

---

**Algorithm 1** FLOYDWARSHALL( $G$ )

---

```

1: Trasforma una rappresentazione a matrice in una rappresentazione a tabella.
2: for all  $x, y \in G.V$  do
3:    $d[x, y] \leftarrow +\infty$ 
4: for all  $x \in G.V$  do
5:   for all  $y \in x.adj$  do
6:      $d[x, y] \leftarrow w(x, y)$ 
7: for  $k := 1$  to  $n$  do
8:   for all  $x \in G.V$  do
9:     for all  $y \in G.V$  do
10:      if  $d[x, v_k] + d[v_k, y] < d[x, y]$  then
11:         $d[x, y] \leftarrow d[x, v_k] + d[v_k, y]$ 

```

---

**Complessità:** Ovviamente, la complessità è  $O(V^3)$ .

## 2.2 Costruzione dei cammini

Al solito, abbiamo ottenuto i pesi ma non ci siamo preoccupati molto dei cammini. Nelle versioni “single-source”, ogni nodo conosceva un singolo predecessore dell’albero dei cammini minimi; e c’era un unico albero di cammini minimi. Qui, ogni nodo conosce  $|V|$  diversi predecessori, uno per ogni sorgente. Parliamo quindi di matrice dei predecessori  $p$  di dimensione  $|V| \times |V|$ , dove  $p[x, y] = \text{nil}$  se  $x = y$  o non esiste cammino da  $x$  a  $y$ , altrimenti rappresenta il predecessore di  $y$  nel cammino minimo con sorgente  $x$ . Per ogni vertice  $x \in V$ , definiamo il sottografo dei predecessori per  $x$ :  $G_{p,x} = (V_{p,x}, E_{p,x})$  in questo modo:

$$\begin{aligned} V_{p,x} &= \{ y \in V : p[x, y] \neq \text{nil} \} \cup \{x\} \\ E_{p,x} &= \{ (p[x, y], y) : y \in V_{p,x} - \{x\} \} \end{aligned}$$

E’ possibile calcolare i valori della tabella  $p$  online, in questo modo:

---

**Algorithm 2** FLOYDWARSHALL( $G$ )

---

```
1: Trasforma una rappresentazione a matrice in una rappresentazione a tabella.
2: for all  $x, y \in G.V$  do
3:    $d[x, y] \leftarrow +\infty$ 
4:    $p[x, y] \leftarrow \text{nil}$ 
5: for all  $x \in G.V$  do
6:   for all  $y \in x.adj$  do
7:      $d[x, y] \leftarrow w(x, y)$ 
8:      $p[x, y] \leftarrow x$ 
9: for  $k := 1$  to  $n$  do
10:  for all  $x \in G.V$  do
11:    for all  $y \in G.V$  do
12:      if  $d[x, v_k] + d[v_k, y] < d[x, y]$  then
13:         $d[x, y] \leftarrow d[x, v_k] + d[v_k, y]$ 
14:         $p[x, y] \leftarrow p[v_k, y]$ 
```

---

In altre parole, se decidiamo di passare per  $v_k$ , allora prendiamo il suo “suggerimento” per il cammino minimo.

## 3 Chiusura transitiva di un grafo orientato

Dato un grafo orientato  $G = (V, E)$ , la chiusura transitiva  $G^* = (V, E^*)$  è definita nel modo seguente:

$$E^* = \{ (x, y) \mid \text{esiste un cammino da } x \text{ a } y \text{ in } G \}$$

Esempio di utilizzo della chiusura transitiva? Considerate uno foglio di calcolo; esiste un grafo sottostante al foglio, dato dalle relazioni fra celle. Quando una cella viene modificata, bisogna aggiornare tutte le celle che dipendono da quella cella. Avere la chiusura transitiva permette di individuare rapidamente quali celle modificare.

Come ottenere la chiusura transitiva di un grafo? Assegnando un peso uno a tutti gli archi, possiamo utilizzare Floyd-Warshall; esiste un cammino fra  $x, y$  se  $d[x, y]$  è un numero finito.

Altrimenti, è possibile utilizzare direttamente la matrice binaria di adiacenza, risparmiando spazio. Se  $m$  è la matrice, abbiamo che

$$\begin{aligned} m^0[x, y] &= m[x, y] \\ m^k[x, y] &= m^{k-1}[x, y] \vee (m^{k-1}[x, v_k] \wedge m^{k-1}[v_k, y]) \end{aligned}$$



## 4 Algoritmo di Johnson

Questo algoritmo trova i cammini minimi fra tutte le coppie nel tempo  $O(V^2 \log V + VE)$ , che nel caso di grafi sparsi è asintoticamente migliore di  $O(V^3)$ . Vi sono molti dettagli nel funzionamento dell'algoritmo, quindi lo vedremo a grandi linee.

Se il grafo contiene solo pesi positivi, ripetendo  $V$  volte l'algoritmo di Dijkstra e utilizzando Heap di Fibonacci si ottiene la complessità prevista.

Se il grafo contiene pesi negativi, si utilizza una tecnica detta **ricalcolo dei pesi** per rimuovere i pesi negativi dai grafi. L'idea è utilizzare una funzione peso  $\hat{w}$  tale per cui esiste un cammino minimo  $p$  in  $G$  con  $\hat{w}$  se e solo se  $p$  è un cammino minimo anche per  $G$  con  $w$ . Costruiremo  $\hat{w}$  in modo da avere solo pesi positivi.

Possiamo semplicemente prendere il valore minimo dei pesi sugli archi e sommarlo a tutti gli archi? Si veda la figura 4. Qual è il cammino minimo da  $A$  a  $B$  nella parte a)? Qual è il cammino minimo da  $A$  a  $B$  nella parte b)? Ovviamente non funziona - bisogna ripensare l'approccio.

Sia  $w : E \rightarrow \mathbf{R}$  una funzione di peso sugli archi; sia  $h : V \rightarrow \mathbf{R}$  una funzione di peso sui nodi. Definiamo:

$$\hat{w}(x, y) = w(x, y) + h(x) - h(y)$$

**Lemma 2.** Sia  $p = v_1, v_2, \dots, v_k$  un cammino qualsiasi dal vertice  $v_1$  al vertice  $v_k$ . Allora  $p$  è un cammino minimo in  $G$  con la funzione peso  $w$  se e solo se  $p$  è un cammino minimo in  $G$  con la funzione peso  $\hat{w}$ .

*Dimostrazione.* Iniziamo dimostrando che  $\hat{w}(p) = w(p) + h(v_1) - h(v_k)$ . Infatti,

$$\begin{aligned} \hat{w}(p) &= \sum_{i=2}^k \hat{w}(v_{i-1}, v_i) \\ &= \sum_{i=2}^k (w(v_{i-1}, v_i) + h(v_{i-1}) - h(v_i)) \\ &= w(p) + \sum_{i=2}^k (h(v_{i-1}) - h(v_i)) \\ &= w(p) + h(v_1) - h(v_k) \end{aligned}$$

L'ultimo passaggio segue dal fatto che tutti i pesi sui nodi intermedi si annullano. Poichè tutti i cammini fra  $v_1$  e  $v_k$  hanno lo stesso "offset"  $h(v_1) - h(v_k)$ , ne segue che il lemma.  $\square$

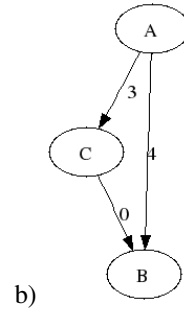
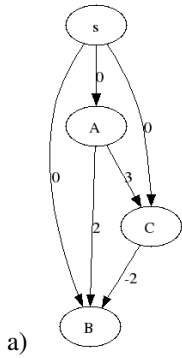
### 4.1 Produrre pesi non negativi

Vogliamo fare in modo che i pesi di tutti gli archi sia non negativi, ovvero che  $\hat{w}(x, y) \geq 0$ .

Definiamo il problema in questo modo: aggiungiamo un nodo  $s \notin V$ , con archi uscenti da  $s$  e incidenti su tutti i nodi  $v$ . Sia  $w(s, v) = 0$  per tutti i nodi  $v$  in  $V$ . Sia  $h(v) = \delta(s, v)$  per tutti i nodi  $v$  in  $V$ .

Per la disuguaglianza triangolare, per ogni arco  $(x, y) \in E$  abbiamo che:

$$\begin{aligned} \delta(s, y) &\leq \delta(s, x) + w(x, y) \Leftrightarrow \\ h(y) &\leq h(x) + w(x, y) \Leftrightarrow \\ \hat{w}(x, y) &= w(x, y) + h(x) - h(y) \geq 0 \end{aligned}$$



## 4.2 Algoritmo

Invece di descrivere lo pseudocodice, descriviamo i vari passi. Assumiamo l'esistenza di archi negativi. L'algoritmo può essere modificato per rilevare cicli negativi (vedi libro).

- Costruiamo un grafo  $G'$  aggiungendo il nodo  $s$  e gli archi relativi: costo  $O(V)$  (per l'aggiunta degli archi);
- Utilizziamo Bellman-Ford su  $G'$  per ottenere i costi minimi di tutti i nodi a partire da  $s$ ; costo  $O(VE)$ . Questi valori vengono utilizzati come funzione  $h$ .
- Calcolare la nuova funzione dei pesi  $\hat{w}$ , utilizzando i valori  $h$ ; costo  $O(E)$
- Per tutti i nodi  $x \in V$ :
  - Utilizziamo DIJKSTRA( $G, \hat{w}, x$ ) e otteniamo  $\hat{\delta}(x, y)$  per ogni vertice  $y \in V$ . Costo (Dijkstra con Fibonacci Heap):  $O(E + V \log V)$ .
  - $d[x, y] = \hat{\delta}(x, y) + h(y) - h(x)$

Il costo totale è quindi  $O(VE + V^2 \log V)$ .

Per un esempio, si veda figura 4.2.