

# Appunti lezione – Capitolo 11

## Strutture di dati e progettazione di algoritmi

Alberto Montresor

03 Giugno, 2016

### 1 Cammini minimi da singola sorgente

#### 1.1 Archi di peso negativo

In alcuni casi, gli archi possono avere peso negativo. Questo influisce (i) sul problema (se è ben definito oppure no) e sulla soluzione (in assenza di archi negativi si devono utilizzare tecniche diverse).

- Algoritmo di Dijkstra/Johnson/Fredman-Tarjan: si suppone che tutti gli archi abbiano peso positivo.
- Algoritmo di Bellman-Ford: gli archi possono avere peso negativo, ma non possono esistere cicli di peso negativo.

#### 1.2 Cicli di peso negativi

Se esiste un ciclo di peso negativo raggiungibile dalla sorgente, non esistono cammini finiti di peso minimo; per qualunque cammino, basterà passare per un ciclo negativo più volte per ottenere un ciclo di costo inferiore.

#### 1.3 Cicli di peso positivo o nullo

Ovviamente, in un cammino minimo non è possibile sia presente un ciclo di peso positivo. I cicli di peso nullo possono essere banalmente eliminati dal cammino minimo, in quanto inutili e ridondanti.

#### 1.4 Rappresentazione dei cammini minimi

Per rappresentare un cammino, utilizziamo una notazione ad albero basata sui padri. È possibile utilizzare la procedura `stampaCammino()` vista per le visite in ampiezza.

---

```
stampaCammino(GRAPH  $G$ , NODE  $r$ , NODE  $s$ , NODE[]  $T$ )
```

---

```
if  $r = s$  then print  $s$   
else if  $T[s] = \text{nil}$  then  
  | print “nessun cammino da  $r$  a  $s$ ”  
else  
  | stampaCammino( $G, r, T[s], T$ )  
  | print  $s$ 
```

---

Durante la visita, non è detto che il sottografo indotto dai campi  $T$  sia un sottoinsieme della soluzione finale; possono essere fatti diversi “aggiustamenti” prima di giungere alla soluzione.

I cammini minimi non sono necessariamente unici, nè lo sono gli alberi dei cammini.

## 1.5 Teorema di Bellman

**Teorema 0.1** (Bellman). Una soluzione ammissibile  $T$  è ottima se e solo se valgono le seguenti condizioni:  $d_v = d_u + w(u, v)$  per ogni arco  $(u, v) \in T$ , e  $d_u + w(u, v) \geq d_v$  per ogni arco  $(u, v) \in E$ .

*Dimostrazione.* (1) Sia  $T$  una soluzione ottima. Consideriamo un generico arco  $(u, v) \in E$  e sia  $w(u, v)$  la sua lunghezza. Ovviamente, se  $(u, v) \in T$ , allora  $d_v = d_u + w(u, v)$ . Se invece  $(u, v) \notin T$ , allora, poiché  $T$  è ottimo, deve risultare  $d_u + w(u, v) \geq d_v$ , poiché altrimenti esisterebbe nel grafo  $G$  un cammino tra  $r$  e  $v$  più corto di quello in  $T$ , e precisamente il cammino tra  $r$  ed  $u$  in  $T$  seguito dall'arco  $(u, v)$ .

(2) Viceversa, sia  $d_u + w(u, v) = d_v$  per ogni  $(u, v) \in T$ , e  $d_u + w(u, v) \geq d_v$ , per ogni arco  $(u, v) \notin T$ . Se, per assurdo, il cammino da  $r$  a  $u$  in  $T$  non fosse ottimo, allora esisterebbe un altro cammino in  $G$  da  $r$  a  $u$  tale che la distanza  $d'_u$  di  $u$  da  $r$  sarebbe minore di  $d_u$ . Sia  $d'_v$  la distanza da  $r$  ad un generico nodo  $v$  che appare in tale cammino. Poiché  $d'_r = d_r = 0$ , ma  $d'_u < d_u$ , esiste un arco  $(h, k)$  in questo cammino per cui  $d'_h \geq d_h$  e  $d'_k < d_k$ . Per costruzione  $d'_h + w(h, k) = d'_k$ , mentre per ipotesi  $d_h + w(h, k) \geq d_k$ . Combinando queste due relazioni, si ottiene:  $d'_k = d'_h + w(h, k) \geq d_h + w(h, k) \geq d_k$ , e quindi  $d'_k \geq d_k$ , che contraddice l'ipotesi.  $\square$

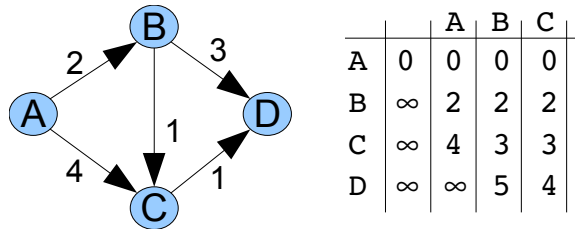


Figura 1: Esempio di funzionamento di Dijkstra

## 1.6 Algoritmo di Bellman-Ford

Risolve il problema nel caso generale in cui i pesi degli archi possono essere negativi. Inoltre, è in grado di dire se esiste un ciclo negativo - nel qual caso, il problema non è ben formulato.

Qual è l'idea generale dell'algoritmo di Bellman? Supponiamo che esista un cammino minimo ignoto fra  $r$  e  $v$ :  $\langle v_1, v_2, \dots, v_k \rangle$ , con  $r \equiv v_1$  e  $v_k \equiv v$ .

Supponiamo di voler calcolare il costo di tale cammino; in base al Teorema di Bellman, sappiamo che  $d_{v_i} = d_{v_{i-1}} + w(v_{i-1}, v_i)$  (per tutti i valori  $i > 1$ ).

Supponiamo quindi di essere in grado di "fare sempre la scelta giusta", ovvero di calcolare i valori  $d$  in quest'ordine:

0.  $d[r] = 0$
1.  $d[v_2] \leftarrow d[r] + w(r, v_2)$
2.  $d[v_3] \leftarrow d[v_2] + w(v_2, v_3)$
3.  $d[v_4] \leftarrow d[v_3] + w(v_3, v_4)$
- ...
- k.  $d[v_k] \leftarrow d[v_{k-1}] + w(v_{k-1}, v_k)$

Ovviamente, noi non conosciamo questo cammino. Non sapendo qual è il passo giusto da fare, li facciamo tutti: ovvero applichiamo il passo di rilassamento a tutti gli archi del sistema. In questo modo, sicuramente avremo completato anche il passo 1.

Poi ri-applichiamo nuovamente il rilassamento, ottenendo anche il passo 2. E così via, per  $n - 1$  iterazioni. Un modo alternativo per vedere l'algoritmo di Bellman è il seguente. Al primo passo, ottengo tutti i migliori cammini di un passo. Al secondo passo, ottengo tutti i migliori cammini di due passi. Così via, fino a quando non raggiungo  $n - 1$  passi.

### 1.7 Cammini minimi in un DAG

In un DAG, i cammini minimi sono sempre ben definiti. Anche in presenza di archi negativi, non possono esistere cicli negativi (perché non esistono cicli).

Nel caso di un DAG, è possibile rilassare gli archi in ordine topologico, una volta sola. Il motivo è semplice: non essendoci cicli, non c'è modo di ritornare ad un nodo già visitato e abbassare il valore del suo campo  $d$ . Sfruttando il nostro algoritmo prototipo, si possono sostituire le seguenti righe.

- (1) `QUEUE S ← Queue(); S.enqueue(r)`
- (2) `u ← S.dequeue()`
- (3) Sezione non necessaria
- (4) Sezione non necessaria

**Correttezza:** Dimostriamo formalmente la correttezza di questo algoritmo.

Vogliamo dimostrare che al termine dell'algoritmo,  $v.d = \delta(s, v)$  per ogni nodo  $v$ . Banalmente, i nodi non raggiungibili da  $s$  iniziano con e mantengono un valore infinito nel campo  $d$ , che è la stima corretta della distanza.

Sia  $v$  un nodo raggiungibile; esiste quindi un cammino  $p = \langle v_1, v_2, \dots, v_k \rangle$ , con  $s \equiv v_1$  e  $v \equiv v_k$ . Prima di rilassare gli archi uscenti da  $v_i$ , tutti i possibili cammini che arrivano al nodo  $v_i$  sono stati percorsi (questo perché non esistono archi  $(v_j, v_i)$ , con  $j > i$ ). Quindi  $v_i.d$  corrisponde al valore finale.

**Complessità:** L'ordinamento topologico può essere realizzato in tempo  $O(m + n)$ .

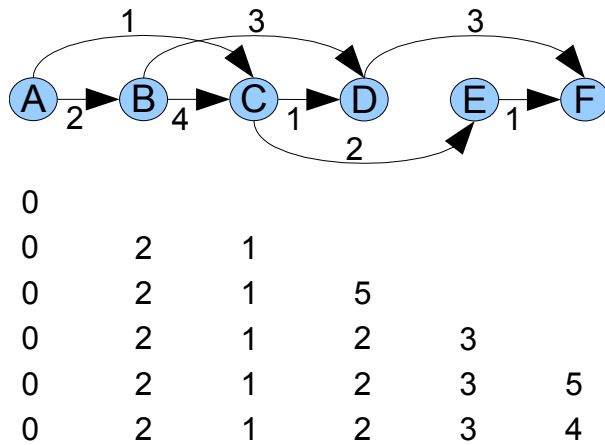


Figura 2: Esempio di funzionamento cammini minimi su DAG