



Algoritmi e Strutture Dati

Cammini minimi

Fabio Patrizi

Cammino minimo: definizione

Sia $G=(V,E)$ un grafo orientato con funzione di costo $w: E \rightarrow \text{Real}$ sugli archi. Il costo $w(\pi)$ di un cammino $\pi = \langle v_0, v_1, v_2, \dots, v_k \rangle$ è dato da:

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

Un **cammino minimo** tra una coppia di vertici x e y è un cammino di costo minore o uguale a quello di ogni altro cammino tra gli stessi vertici.



Proprietà dei cammini minimi (1/3)

Sottostruttura ottima:

Sia $G=(V,E)$ un grafo orientato con funzione di costo $w: E \rightarrow \mathbf{Real}$ sugli archi e sia π un cammino minimo in G . Allora ogni sottocammino di π è esso stesso un cammino minimo in G (tra i nodi di partenza e destinazione)



Proprietà dei cammini minimi (2/3)

Supponiamo che i costi degli archi possano essere negativi e che vi siano in un grafo cicli con costo minore di zero.

Proprietà:

In un grafo con cicli negativi, se due vertici x e y appartengono a un ciclo di costo negativo allora non esiste nessun cammino minimo finito tra di essi



Proprietà dei cammini minimi (3/3)

Se G non contiene cicli negativi, tra ogni coppia di vertici connessi in G esiste sempre un cammino minimo **semplice**



Distanza fra vertici

- La distanza d_{xy} tra due vertici x e y è il costo di **un cammino minimo** tra da x a y , o $+\infty$ se i due vertici non sono connessi:

$$d_{xy} = \begin{cases} w(\pi_{xy}), & \text{Se esiste cammino tra } x \text{ e } y \text{ in } G \\ +\infty, & \text{altrimenti} \end{cases}$$

Distanza fra vertici (proprietà)

- **Disuguaglianza triangolare:** per ogni x , y e z

$$d_{xz} \leq d_{xy} + d_{yz}$$

- **Condizione di Bellman:** per ogni arco (u,v) e per ogni vertice s

$$d_{su} + w(u, v) \geq d_{sv}$$



Cammini minimi e distanze

Un arco (u,v) appartiene ad un **cammino minimo** a partire da un vertice s se e solo se la condizione di Bellman vale con l'uguaglianza, ovvero se u è raggiungibile da s e $d_{su} + w(u,v) = d_{sv}$

Calcolare cammini minimi dalle distanze

algoritmo cammino(*grafo* G , *distanze* d , *vertice* x , *vertice* y)

1. $\pi \leftarrow \langle y \rangle$; $v \leftarrow y$
2. **while** ($v \neq x$) **do**
3. **for each** (arco (u, v) in G) **do**
4. **if** ($d_{xu} + w(u, v) = d_{xv}$) **then**
5. aggiungi u come primo vertice in π
6. $v \leftarrow u$
7. **break**
8. **return** π



Albero dei cammini minimi

I cammini minimi da un vertice s a tutti gli altri vertici del grafo possono essere rappresentati tramite un albero radicato in s . Tale albero è detto **albero dei cammini minimi**



Varianti del problema dei cammini minimi

- Problema dei **cammini minimi fra tutte le coppie**: calcolo delle distanze tra ogni coppia di vertici nel grafo
- Problema dei **cammini minimi a sorgente singola**: calcolo delle distanze a partire da un dato vertice chiamato **sorgente**
- Problema dei **cammini minimi fra singola coppia**: calcolo della distanza tra due vertici prefissati

Tecnica del rilassamento

Gli algoritmi che vedremo per calcolare le distanze per identificare i cammini minimi partono da stime D_{xy} per eccesso delle distanze d_{xy} , ovvero: $D_{xy} \geq d_{xy}$ e le aggiornano progressivamente decrementandole fino a renderle esatte ($D_{xy} = d_{xy}$).

Aggiornamento delle stime basato sul seguente passo di rilassamento:

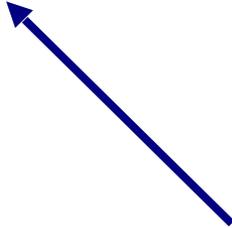
(RILASSAMENTO) **if** $(D_{xv} + w(\pi_{vy}) < D_{xy})$
 then $D_{xy} \leftarrow D_{xv} + w(\pi_{vy})$

Algoritmo generico per il calcolo delle distanze

```
algoritmo distanzeGenerico(grafo  $G$ )  $\rightarrow$  distanze  
  inizializza  $D$  in modo che  $D_{xy} \geq d_{xy}$  per ogni coppia di interesse  $(x, y)$   
  while ( esiste  $(x, y)$  fra le coppie di interesse tale che  $D_{xy} > d_{xy}$  ) do  
    considera un vertice  $v$  e un cammino  $\pi_{vy}$   
    if ( $D_{xv} + w(\pi_{vy}) < D_{xy}$ ) then  $D_{xy} \leftarrow D_{xv} + w(\pi_{vy})$   
  return  $D$ 
```

Gli algoritmi per il calcolo delle distanze che vedremo differiscono principalmente su come scegliere il vertice v ed il cammino π_{vy}

rilassamento





Algoritmo di Bellman e Ford

(per cammini minimi a sorgente singola)

Approccio di Bellman e Ford

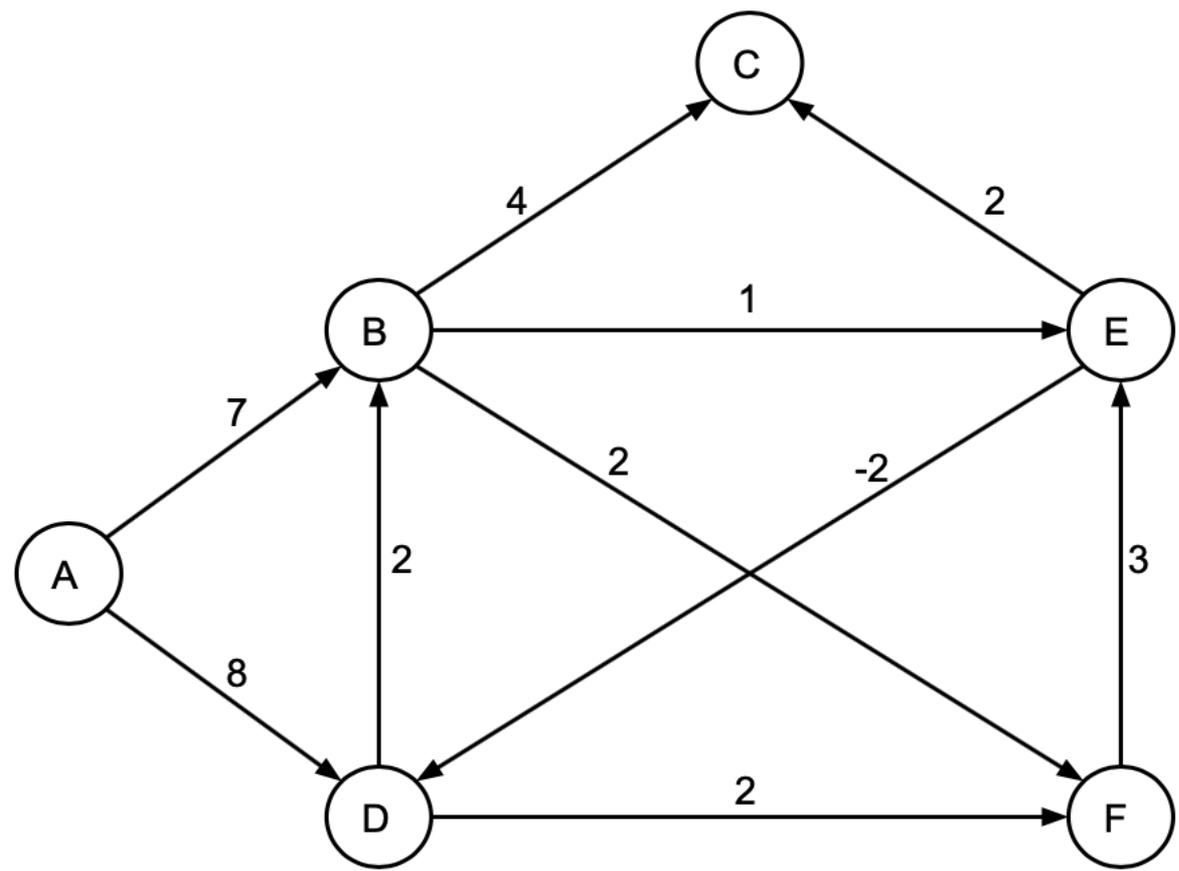
- L'algoritmo di Bellman e Ford si basa sulla seguente idea:
 - Effettuando una prima passata "rilassando" tutti gli archi di G , otteniamo sicuramente $\mathbf{D}_{sv1} = \mathbf{d}_{sv1}$
 - Ripetendo la passata su tutti gli archi otterremo $\mathbf{D}_{sv2} = \mathbf{d}_{sv2}$
 - Con n passate arriveremo ad avere tutte le distanze

Pseudocodice

algoritmo BellmanFord(*grafo* G , *vertice* s) \rightarrow *distanze*
inizializza D tale che $D_{sv} = +\infty$ per $v \neq s$, e $D_{ss} = 0$
for $i = 1$ **to** n **do**
 for each $((u, v) \in E)$ **do**
 if $(D_{su} + w(u, v) < D_{sv})$ **then** $D_{sv} \leftarrow D_{su} + w(u, v)$
return D



Esempio



Approccio di Bellman e Ford

- L'algoritmo cicla per n volte, dove n è il numero di vertici del grafo G .
- In ogni ciclo effettua il rilassamento di tutti gli m archi.



Tempo di esecuzione: **$O(nm)$**

- Dopo la j -esima passata, i primi j rilassamenti corretti sono stati certamente eseguiti.
- Esegue però molti rilassamenti inutili!



Algoritmo per grafi diretti aciclici **(per cammini minimi a sorgente singola)**



Algoritmo per grafi diretti aciclici

- L'algoritmo di Bellman e Ford esegue un grande numero di operazioni di rilassamento inutili.
- Eseguire i passi di rilassamento del giusto ordine è cruciale per evitare di fare operazioni inutili.
- Il prossimo algoritmo sfrutterà alcune proprietà del grafo in modo da rilassare ciascun arco esattamente una volta.



Ordinamento topologico (1/2)

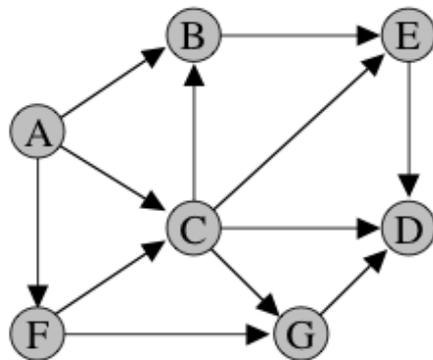
In un grafo aciclico, se esiste un cammino da u a v allora **non esiste** un cammino da v ad u . Questo ci permette di definire un ordinamento dei vertici, tale che u precede v nell'ordinamento se esiste un cammino da u a v .

Tale ordinamento è detto **ordinamento topologico**

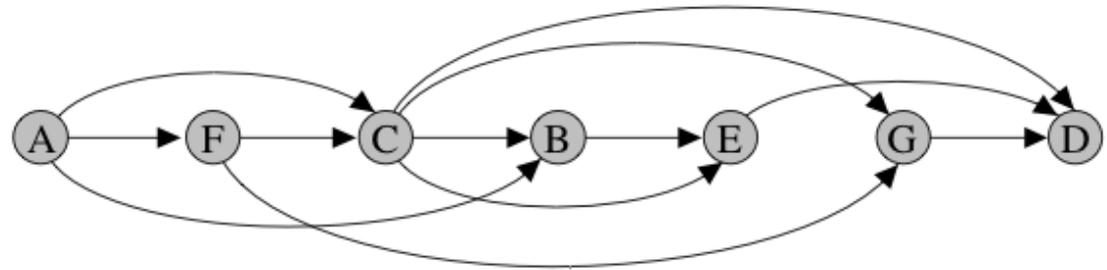
Ordinamento topologico (2/2)

Funzione $\sigma: V \rightarrow \{1, \dots, n\}$ tale che $\sigma(u) < \sigma(v)$ se esiste un cammino da u a v in G

L'ordinamento topologico non è sempre unico.



(a)



(b)

(a) Esempio di grafo aciclico; (b) Ordinamento topologico del grafo (a).



Calcolo di un ordinamento topologico (1/2)

IDEA:

1. scelgo un nodo u senza archi entranti, lo cancello dal grafo e pongo $ord(u) = 1$.
2. Ripeto il ragionamento: scelgo un altro vertice v senza archi entranti, lo cancello dal grafo e pongo $ord(v) = 2$
3. Ripeto finché non ho eliminato tutti i nodi

Calcolo di un ordinamento topologico (2/2)

algoritmo `ordinamentoTopologico`(*grafo* G) \rightarrow *lista*

$\hat{G} \leftarrow G$

ord \leftarrow lista vuota di vertici

while (esiste un vertice u senza archi entranti in \hat{G}) **do**

 appendi u come ultimo elemento di *ord*

 rimuovi da \hat{G} il vertice u e tutti i suoi archi uscenti

if (\hat{G} non è diventato vuoto) **then errore** il grafo G non è aciclico

return *ord*

Tempo di esecuzione: **$O(n+m)$**



Cammini minimi in grafi aciclici: idea

Il prossimo algoritmo è una variante dell'algoritmo di Bellman e Ford per soli grafi aciclici in cui le operazioni di rilassamento vengono effettuate secondo un ordinamento topologico:

Vengono prima rilassati tutti gli archi (u,v) con $ord(u) = 1$ nell'ordinamento topologico ord , poi quelli con $ord(u) = 2$ e così via.

In questo modo si evita di effettuare rilassamenti inutili: ogni arco viene rilassato esattamente una volta.

Cammini minimi in grafi aciclici: pseudocodice

```
algoritmo distanzeAciclico(grafo  $G$ , vertice  $s$ )  $\rightarrow$  distanze  
  inizializza  $D$  tale che  $D_{sv} = +\infty$  per  $v \neq s$ , e  $D_{ss} = 0$   
   $ord \leftarrow$  ordinamentoTopologico( $G$ )  
  for  $i = 1$  to  $n$  do  
    sia  $u$  l' $i$ -esimo vertice nell'ordinamento topologico  $ord$   
    for each  $((u, v) \in E)$  do  
      if  $(D_{su} + w(u, v) < D_{sv})$  then  $D_{sv} \leftarrow D_{su} + w(u, v)$   
  return  $D$ 
```

Tempo di esecuzione: **$O(n+m)$**



Algoritmo di Dijkstra

**(per cammini minimi a sorgente singola
in grafi con costi non negativi)**

Estendere l'albero dei cammini minimi

Negli anni '50 E.W. Dijkstra scoprì la seguente proprietà:

Se T è un albero dei cammini minimi radicato in s che non include tutti i vertici raggiungibili da s , l'arco (u,v) tale che $u \in T$ e $v \notin T$ che minimizza la quantità $d_{su} + w(u,v)$ appartiene a un cammino minimo da s a v

Tale proprietà ci fornisce un semplice metodo per costruire un albero dei cammini minimi che include tutti i vertici raggiungibili da s

Approccio di Dijkstra

IDEA:

Applichiamo il seguente passo fino alla costruzione dell'albero T :

Scegli un arco (u,v) con $u \in T$ e $v \notin T$ che minimizza la quantità $D_{su} + w(u,v)$, effettua il passo di rilassamento $D_{sv} \leftarrow D_{su} + w(u,v)$, ed aggiungilo a T

Pseudocodice

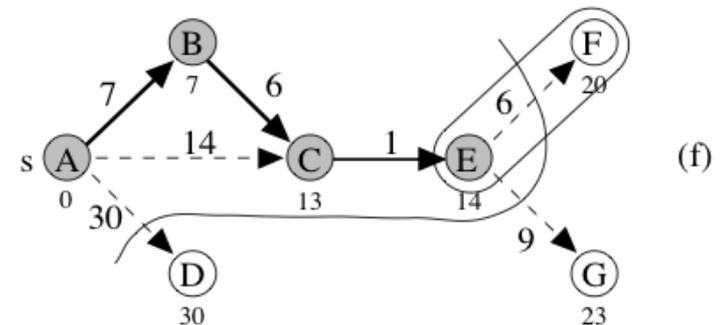
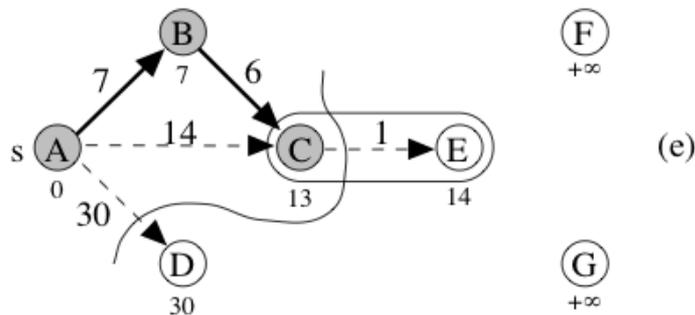
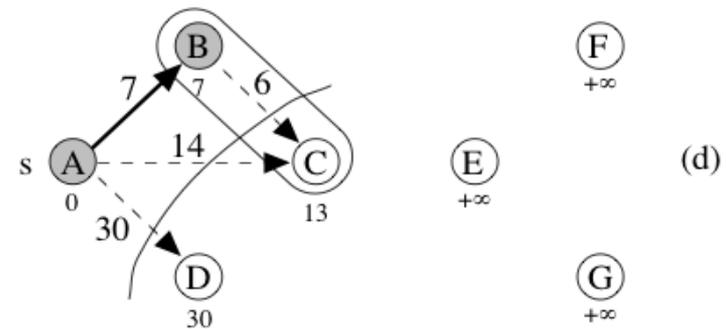
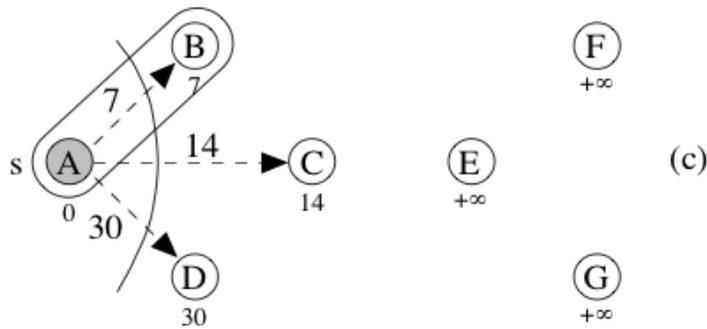
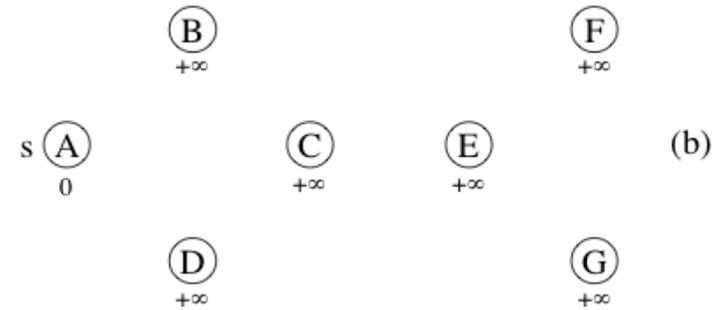
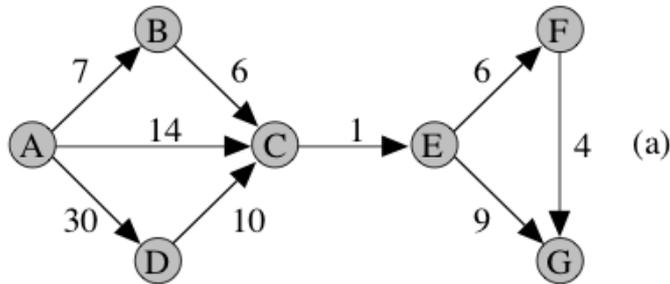
algoritmo DijkstraGenerico(*grafo* G , *vertice* s) \rightarrow *albero*

1. inizializza D tale che $D_{sv} = +\infty$ per $v \neq s$, e $D_{ss} = 0$
2. $T \leftarrow$ albero formato dal solo nodo s
3. **while** (T ha meno di n nodi) **do**
4. trova l'arco (u, v) incidente su T con $D_{su} + w(u, v)$ minimo
5. $D_{sv} \leftarrow D_{su} + w(u, v)$ {rilassamento}
6. rendi u padre di v in T
7. **return** T

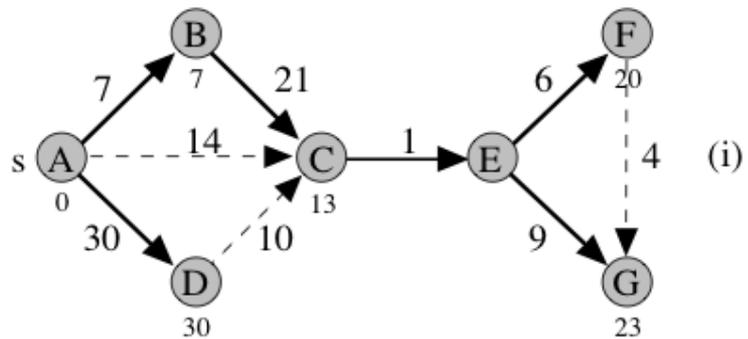
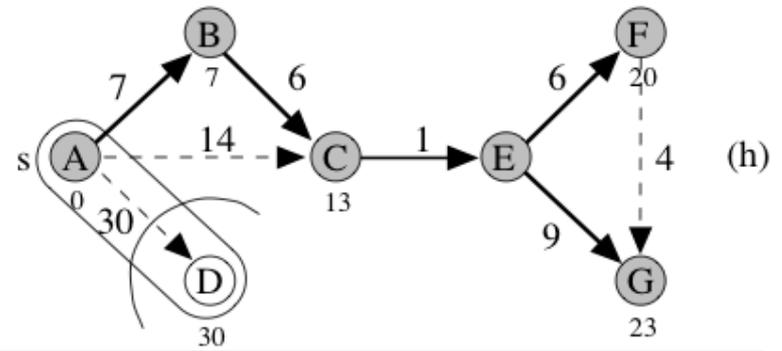
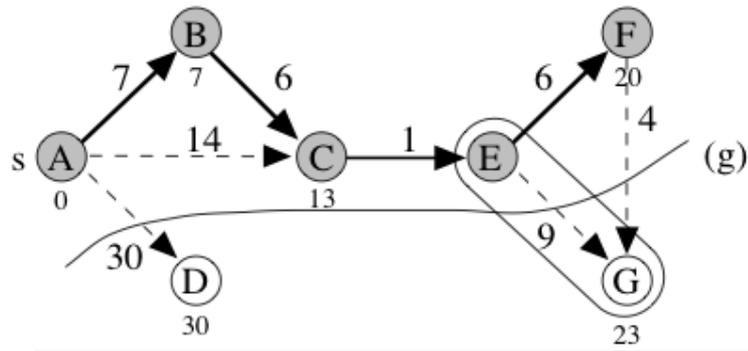
Si noti la somiglianza con l'algoritmo di Prim per il calcolo di un minimo albero ricoprente: come l'algoritmo di Prim anche l'algoritmo di Dijkstra si basa sulla tecnica *golosa*.

È possibile realizzare una versione dell'algoritmo che calcola T in $O(m + n \log n)$

Esempio (1/2)



Esempio (2/2)





Riepilogo

- Algoritmi classici per il calcolo di distanze (e quindi di cammini minimi), basati sulla tecnica del rilassamento:
 - Bellman e Ford: cammini minimi a sorgente singola, grafi diretti senza cicli negativi, tempo $O(nm)$
 - Grafi diretti aciclici: cammini minimi a sorgente singola in tempo $O(n+m)$
 - Dijkstra: cammini minimi a sorgente singola, grafi diretti senza pesi negativi, tempo $O(m+n \log n)$