

13.

Cammini minimi

Pseudocodice

Codice 13.1 Calcolo della lunghezza dei cammini minimi tra ogni coppia di vertici

```
Algoritmo FloydWarshall (grafo pesato  $G = (V, E, \omega) \rightarrow$  matrice distanze  
/* La funzione  $\omega$  assegna un peso a ciascun arco.  
   Il grafo non deve contenere cicli di peso negativo. */  
  
Sia  $D$  una matrice  $n \times n$ , con  $V = \{v_1, v_2, \dots, v_n\}$   
  
// Per ogni coppia di vertici, calcola la minima lunghezza di un cammino  
// che li congiunge, senza passare per vertici intermedi (cioè singolo nodo  
// o singolo arco)  
for  $i \leftarrow 1$  to  $n$  do  
  for  $j \leftarrow 1$  to  $n$  do  
    if  $i = j$  then  $D[i, j] \leftarrow 0$   
    else if  $(v_i, v_j) \in E$  then  $D[i, j] \leftarrow \omega(v_i, v_j)$   
    else  $D[i, j] \leftarrow \infty$   
  
  for  $k \leftarrow 1$  to  $n$  do  
    // per ogni coppia di vertici, calcola la minima lunghezza di un cammino  
    // che li congiunge, i cui vertici intermedi appartengono a  $\{v_1, v_2, \dots, v_k\}$   
    // cioè hanno indice  $\leq k$   
    for  $i \leftarrow 1$  to  $n$  do  
      for  $j \leftarrow 1$  to  $n$  do  
        if  $D[i, k] + D[k, j] < D[i, j]$  then  
           $D[i, j] \leftarrow D[i, k] + D[k, j]$   
  
  return  $D$ 
```

Il presente materiale integra *ma non sostituisce* il libro di testo consigliato.
Data pubblicazione: 16 dicembre 2019

© 2019 Giovanni Pighizzini

Il contenuto di queste pagine è protetto dalle leggi sul copyright e dalle disposizioni dei trattati internazionali. Il titolo ed i copyright relativi alle pagine sono di proprietà dell'autore. Le pagine possono essere riprodotte ed utilizzate liberamente dagli studenti, dagli istituti di ricerca, scolastici e universitari afferenti al Ministero dell'Istruzione, dell'Università e della Ricerca, per scopi istituzionali, non a fine di lucro. Ogni altro utilizzo o riproduzione (ivi incluse, ma non limitatamente a, le riproduzioni a mezzo stampa, su supporti magnetici o su reti di calcolatori) in toto o in parte è vietata, se non esplicitamente autorizzata per iscritto, a priori, da parte dell'autore. L'informazione contenuta in queste pagine è ritenuta essere accurata alla data della pubblicazione. Essa è fornita per scopi meramente didattici e non per essere utilizzata in progetti di impianti, prodotti, ecc. L'informazione contenuta in queste pagine è soggetta a cambiamenti senza preavviso. L'autore non si assume alcuna responsabilità per il contenuto di queste pagine (ivi incluse, ma non limitatamente a, la correttezza, completezza, applicabilità ed aggiornamento dell'informazione). In ogni caso non può essere dichiarata conformità all'informazione contenuta in queste pagine. In ogni caso questa nota di copyright non deve mai essere rimossa e deve essere riportata anche in utilizzi parziali.

Codice 13.2 Calcolo della lunghezza dei cammini minimi da un vertice s a tutti gli altri

```

Algoritmo Dijkstra (grafo pesato  $G = (V, E, \omega)$ , vertice  $s$ )  $\rightarrow$  vettore distanze
/* La funzione  $\omega$  assegna un peso a ciascun arco.
   Il grafo non deve contenere archi di peso negativo. */
Sia  $D$  un vettore con insieme di indici  $V$ 
 $D[s] \leftarrow 0$ 
foreach  $v \in V \setminus \{s\}$  do  $D[v] \leftarrow \infty$ 
 $C \leftarrow V$ 
while  $C \neq \emptyset$  do
     $u \leftarrow$  elemento di  $C$  con  $D[u]$  minima
     $C \leftarrow C \setminus \{u\}$ 
    foreach  $(u, v) \in E$  do
        if  $D[u] + \omega(u, v) < D[v]$  then
             $D[v] \leftarrow D[u] + \omega(u, v)$ 
return  $D$ 

```

Codice 13.3 Calcolo della lunghezza dei cammini minimi da un vertice s a tutti gli altri

```

Algoritmo Dijkstra (grafo pesato  $G = (V, E, \omega)$ , vertice  $s$ )  $\rightarrow$  vettore distanze
/* La funzione  $\omega$  assegna un peso a ciascun arco.
   Il grafo non deve contenere archi di peso negativo. */
Sia  $D$  un vettore con insieme di indici  $V$ 
 $D[s] \leftarrow 0$ 
foreach  $v \in V \setminus \{s\}$  do  $D[v] \leftarrow \infty$ 
Sia  $C$  una coda con priorità inizialmente vuota
foreach  $v \in V$  do  $C.insert(v, D[v])$ 
while  $C \neq \emptyset$  do
     $u \leftarrow C.deleteMin()$ 
    foreach  $(u, v) \in E$  do
        if  $D[u] + \omega(u, v) < D[v]$  then
             $D[v] \leftarrow D[u] + \omega(u, v)$ 
             $C.changeKey(v, D[v])$ 
return  $D$ 

```

Note

Stima del tempo utilizzato dall'algoritmo descritto nel Codice 13.3, nel caso di implementazione delle code con priorità mediante heap rappresentato con vettore posizionale.

- Costruzione coda: tempo $O(n)$.
- Ogni operazione *deleteMin* costa $O(\log n)$, ne viene eseguita una per ogni vertice: in totale tempo $O(n \log n)$.
- Ogni operazione *changeKey* costa $O(\log n)$, ne viene eseguita al più una per ogni arco: in totale tempo $O(m \log n)$.

Il tempo totale risulta $O(n + n \log n + m \log n)$. Se il grafo è connesso m è almeno dello stesso ordine di n . Pertanto il tempo è $O(m \log n)$. È possibile ridurre a $O(m + n \log n)$ se le code con priorità vengono implementate con heap di Fibonacci (v. libro di testo).