13. Cammini minimi

Pseudocodice

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

Algoritmo *FloydWarshall (grafo pesato $G = (V, E, \omega)$)* $\rightarrow$ *matrice distanze*

```
/* La funzione ω assegna un peso a ciascun arco.
   Il grafo non deve contenere cicli di peso negativo.                    */

Sia D una matrice n × n, con V = {v_1, v_2, ..., 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 ← 1 to n do
    for j ← 1 to n do
        if i = j then D[i,j] ← 0
        else if (v_i, v_j) ∈ E then D[i,j] ← ω(v_i, v_j)
        else D[i,j] ← ∞

for k ← 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, ..., v_k}
    // cioè hanno indice ≤ k
    for i ← 1 to n do
        for j ← 1 to n do
            if D[i,k] + D[k,j] < D[i,j] then
                D[i,j] ← 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

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

Algoritmo *Dijsktra (grafo pesato $G = (V, E, \omega)$, vertice s) → vettore distanze*
`/* La funzione` ω `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 mimimi da un vertice s a tutti gli altri

Algoritmo *Dijsktra (grafo pesato $G = (V, E, \omega)$, vertice s) → vettore distanze*
`/* La funzione` ω `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).