/

11.4 Chiusura transitiva

Il problema che consideriamo in questa sezione riguarda il calcolo della chiusura transitiva di un grafo. Dato un grafo orientato $G = \langle V, E \rangle$, si vuole determinare il grafo (orientato) $G^* = \langle V, E^* \rangle$ tale che,

per ogni coppia di nodi $u, v \in V$, esiste il lato (u, v) in G^* se e solo se esiste in G un cammino da u a v di lunghezza maggiore di 0. In questa sezione consideriamo solo cammini di lunghezza maggiore di 0; la trattazione può essere facilmente estesa in modo da comprendere anche i cammini di lunghezza 0 (ovvero quelli della forma (v) , dove $v \in V$).

Un algoritmo classico per risolvere il problema è basato sul calcolo di una famiglia di coefficienti booleani che assumono valore 1 o 0 a seconda se esiste o meno un cammino tra due nodi passante per un certo insieme di vertici.

Siano $v_1, v_2, \dots, v_n$ i nodi del grafo G . L'idea è quella di verificare, per ogni coppia di nodi v_i, v_j , se esiste un lato da v_i a v_j oppure un cammino da v_i a v_j passante per il nodo v_1 ; poi, se esiste un cammino da v_i a v_j passante al più per nodi di indice 1 o 2; quindi se esiste un cammino da v_i a v_j passante per nodi di indice minore o uguale a 3, e così via. Si tratta quindi di eseguire n cicli: al k -esimo ciclo si verifica, per ogni coppia di nodi v_i, v_j , se esiste un cammino da v_i a v_j passante per nodi di indice minore o uguale a k (escludendo i due valori i e j). Tale verifica può essere eseguita tenendo conto dei risultati forniti dal ciclo precedente. Infatti, esiste un cammino da v_i a v_j passante per nodi di indice minore o uguale a k se e solo se si verifica uno dei fatti seguenti:

1. esiste un cammino da v_i a v_j passante per nodi di indice minore o uguale a $k - 1$, oppure
2. esiste un cammino da v_i a v_k e uno da v_k a v_j entrambi passanti per vertici di indice minore o uguale a $k - 1$.

Per ogni coppia di indici $i, j \in \{1, 2, \dots, n\}$ definiamo la famiglia di coefficienti C_{ij}^k , dove $k \in \{0, 1, 2, \dots, n\}$, nel modo seguente:

$$C_{ij}^0 = \begin{cases} 1 & \text{se } (v_i, v_j) \in E, \\ 0 & \text{altrimenti,} \end{cases}$$

mentre, per ogni $k \in \{1, 2, \dots, n\}$,

$$C_{ij}^k = \begin{cases} 1 & \text{se } (v_i, v_j) \in E \text{ oppure esiste in } G \text{ un cammino da} \\ & \text{ } v_i \text{ a } v_j \text{ che passa per nodi di indice } t \text{ tale che } t \leq k, \\ 0 & \text{altrimenti.} \end{cases}$$

Nota che $[C_{ij}^0]$ coincide di fatto con la matrice di adiacenza del grafo di ingresso.

Per il ragionamento precedente i coefficienti C_{ij}^k , al variare di i e j in $\{1, 2, \dots, n\}$, sono legati ai valori C_{ij}^{k-1} dalla seguente equazione:

$$C_{ij}^k = C_{ij}^{k-1} \vee (C_{ik}^{k-1} \wedge C_{kj}^{k-1}) \tag{11.1}$$

Chiaramente $C_{ij}^n = 1$ se e solo se in G esiste un cammino da v_i a v_j . Possiamo allora facilmente descrivere un algoritmo che calcola inizialmente i coefficienti C_{ij}^0 e quindi, usando l'equazione (11.1), tutti i successivi: per ogni $k = 1, 2, \dots, n$ tutti i coefficienti C_{ij}^k , con $i, j = 1, 2, \dots, n$, possono essere ottenuti dai C_{ij}^{k-1} . Questo calcolo può essere ovviamente eseguito mantenendo due matrici di dimensione $n \times n$, una per conservare i valori C_{ij}^{k-1} , l'altra per i corrispondenti C_{ij}^k . Tuttavia, osserviamo che per ogni i, j, k valgono le seguenti identità:

$$C_{ik}^{k-1} = C_{ik}^k, \quad C_{kj}^{k-1} = C_{kj}^k.$$

Questo significa che possiamo usare una sola matrice per mantenere entrambe le famiglie di coefficienti.

L'algoritmo complessivo è quindi descritto dalla seguente procedura che calcola la matrice di coefficienti booleani $[C_{ij}]_{i,j=1,2,\dots,n}$. Tale matrice coincide inizialmente con la matrice di adiacenza del grafo G mentre, al termine della computazione, rappresenta la matrice di adiacenza della sua chiusura transitiva G^* .

```

begin
  for  $i = 1, 2, \dots, n$  do
    for  $j = 1, 2, \dots, n$  do
      if  $(v_i, v_j) \in E$  then  $C_{ij} := 1$ 
      else  $C_{ij} := 0$ 
    for  $k = 1, 2, \dots, n$  do
      for  $i = 1, 2, \dots, n$  do
        for  $j = 1, 2, \dots, n$  do
          if  $C_{ij} = 0$  then  $C_{ij} := C_{ik} \wedge C_{kj}$ 
        end
      end
    end
  end

```

È facile verificare che, assumendo il criterio di costo uniforme, l'algoritmo richiede un tempo di calcolo $\Theta(n^3)$ e uno spazio di memoria $\Theta(n^2)$ su ogni grafo di ingresso di n nodi.

Esercizio

Esegui l'algoritmo sopra descritto sul grafo $G = (\{1, 2, 3\}, \{(1, 2), (2, 3), (3, 1)\})$. Determina in particolare le matrici di coefficienti $[C_{ij}^k]_{i,j=1,2,3}$ per $k = 0, 1, 2, 3$.

11.5 Cammini minimi

Un altro problema classico che si può risolvere mediante programmazione dinamica riguarda il calcolo dei cammini di peso minimo che connettono i nodi in un grafo pesato.

Consideriamo un grafo diretto $G = \langle V, E \rangle$ e una funzione costo $w : E \rightarrow \mathbb{Q}$ tale che $w(e) \geq 0$ per ogni $e \in E$. Per ogni cammino ℓ in G , $\ell = (v_1, v_2, \dots, v_m)$, chiamiamo peso (o costo) di ℓ la somma dei costi dei suoi lati:

$$c(\ell) = \sum_{i=1}^{m-1} w(v_i, v_{i+1})$$

Per ogni coppia di nodi $u, v \in V$ si vuole determinare un cammino ℓ di peso minimo tra tutti i cammini da u a v , insieme al suo costo $c(\ell)$.

Siano $v_1, v_2, \dots, v_n$ i nodi di G . Il problema può essere risolto calcolando le matrici $D = [d_{ij}]_{i,j=1,2,\dots,n}$ e $P = [p_{ij}]_{i,j=1,2,\dots,n}$ le cui componenti sono definite nel modo seguente: per ogni coppia di indici distinti i, j , se esiste un cammino da v_i a v_j in G , d_{ij} rappresenta il costo del cammino di peso minimo che congiunge v_i e v_j e p_{ij} è il predecessore di v_j in tale cammino; se invece non esiste un cammino da v_i a v_j allora d_{ij} assume un valore convenzionale ∞ , superiore al peso di ogni cammino in G , e p_{ij} assume il valore indefinito $\perp$.

Conoscendo la matrice P è possibile determinare un cammino di peso minimo da v_i a v_j semplicemente scorrendo in modo opportuno la i -esima riga della matrice: se $k_1 = p_{ij}$ allora v_{k_1} è il penultimo nodo del cammino, se $k_2 = p_{ik_1}$ allora v_{k_2} è il terzultimo, e così via fino a determinare il primo nodo, ovvero v_i .

Abbiamo così definito il problema per grafi con pesi non negativi. L'algoritmo che presentiamo tuttavia risolve il problema nel caso più generale di grafi con pesi di segno qualsiasi, purché questi non formino cicli di costo negativo. Osserviamo che se esiste un ciclo di peso negativo lo stesso problema non è ben definito.

Il metodo utilizzato per determinare la soluzione è simile a quello descritto nella sezione precedente per calcolare la chiusura transitiva di un grafo. Per ogni tripla di indici $i, j, k \in \{1, 2, \dots, n\}$ definiamo il coefficiente c_{ij}^k come il costo del cammino di peso minimo da v_i a v_j passante per nodi di indice al più uguale a k (esclusi i e j). Chiaramente $d_{ij} = c_{ij}^n$. Inoltre definiamo i coefficienti c_{ij}^0 nel modo seguente:

$$c_{ij}^0 = \begin{cases} w(v_i, v_j) & \text{se } i \neq j \text{ e } (v_i, v_j) \in E \\ 0 & \text{se } i = j \\ \infty & \text{altrimenti} \end{cases}$$

Anche per questa famiglia di coefficienti possiamo definire un'equazione, analoga alla (11.1), che permette di calcolare i valori c_{ij}^k , per $i, j = 1, 2, \dots, n$, conoscendo i c_{ij}^{k-1} :

$$c_{ij}^k = \min\{c_{ij}^{k-1}, c_{ik}^{k-1} + c_{kj}^{k-1}\}. \quad (11.2)$$

Basta osservare infatti che se ℓ è un cammino di peso minimo da v_i a v_j passante per nodi di indice minore o uguale a k allora si verifica uno dei due fatti seguenti:

1. ℓ passa solo per nodi di indice minore o uguale a $k - 1$, oppure
2. ℓ è composto da due cammini che vanno rispettivamente da v_i a v_k e da v_k a v_j , entrambi di peso minimo tra tutti i cammini (congiungenti i rispettivi estremi) passanti per nodi di indice minore o uguale a $k - 1$.

Nel primo caso il costo di ℓ è c_{ij}^{k-1} mentre nel secondo è $c_{ik}^{k-1} + c_{kj}^{k-1}$; inoltre in quest'ultimo il predecessore di v_j in ℓ equivale al predecessore di v_j nel cammino di costo minimo da v_k a v_j passante per nodi di indice minore o uguale a $k - 1$.

Applicando l'equazione (11.2) possiamo allora descrivere un algoritmo del tutto simile a quello presentato nella sezione precedente. Anche in questo caso possiamo limitarci a mantenere una sola matrice di valori c_{ij}^k poichè valgono le identità

$$c_{ik}^{k-1} = c_{ik}^k \text{ e } c_{kj}^{k-1} = c_{kj}^k$$

per ogni tripla di indici i, j, k . Così l'algoritmo calcola la matrice $C = [c_{ij}]$ dei costi e quella $B = [b_{ij}]$ dei predecessori dei cammini minimi che, al termine della computazione, coincideranno con le matrici D e P rispettivamente.

```

begin
  for  $i = 1, 2, \dots, n$  do
    for  $j = 1, 2, \dots, n$  do
      if  $i = j$  then  $\begin{cases} c_{ii} := 0 \\ b_{ii} := i \end{cases}$ 
      else if  $(v_i, v_j) \in E$  then  $\begin{cases} c_{ij} := w(v_i, v_j) \\ b_{ij} := i \end{cases}$ 
      else  $\begin{cases} c_{ij} := \infty \\ b_{ij} := \perp \end{cases}$ 
    for  $k = 1, 2, \dots, n$  do
      for  $i = 1, 2, \dots, n$  do
        for  $j = 1, 2, \dots, n$  do
          if  $c_{ik} + c_{kj} < c_{ij}$  then  $\begin{cases} c_{ij} := c_{ik} + c_{kj} \\ b_{ij} := b_{kj} \end{cases}$ 
    end

```

Concludiamo osservando che, assumendo il criterio di costo uniforme, il tempo di calcolo richiesto dall'algoritmo su ogni grafo di input di n nodi è $\Theta(n^3)$, mentre lo spazio di memoria è $\Theta(n^2)$.