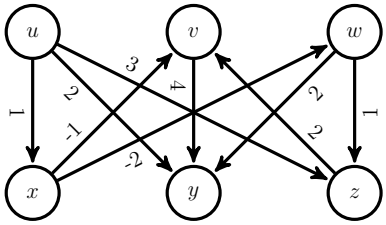
Esercitazione 9: All-Pairs Shortest Paths and More

Giacomo Paesani

May 22, 2024

Esercizio 1. Il problema All-Pairs Shortest Path consiste nel ottenere il peso di un cammino minimo per ogni coppia di vertici di un grafo, cioè dato un grafo diretto $G = (V, E)$, la richiesta è quella di ottenere per ogni due vertici $u, v \in V$, il cammino di peso minimo da u a v. Risolvere questo problema fornendo la matrice dei cammini e dei padri per ognuno dei seguenti grafi:



Soluzione 1. La soluzione a questo esercizio prevede di avere il grafo G salvato sotto forma di matrice di adiacenza, che denotiamo con W. Per risolvere il problema del primo esempio adottiamo una strategia che simula il prodotto di matrici. Per ogni $k \geq 0$, la matrice $L^{(k)}$ contiene le informazioni sui cammini minimi di lunghezza al più k: l'elemento $\ell_{ij}^{(k)}$ rappresenta il peso minimo di un cammino dal vertice i al vertice j di lunghezza al più k.

$$L^{(1)} = L^{(0)} \cdot W = W = \begin{pmatrix} 0 & \infty & \infty & 1 & 2 & 3 \\ \infty & 0 & \infty & \infty & 4 & \infty \\ \infty & \infty & 0 & \infty & 2 & 1 \\ \infty & -1 & -2 & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 & \infty \\ \infty & 2 & \infty & \infty & \infty & 0 \end{pmatrix}$$

$$\Pi^{(1)} = \begin{pmatrix} u & \infty & \infty & u & u & u \\ \infty & v & \infty & \infty & v & \infty \\ \infty & \infty & w & \infty & w & w \\ \infty & x & x & x & \infty & \infty \\ \infty & \infty & \infty & \infty & y & \infty \\ \infty & z & \infty & \infty & \infty & z \end{pmatrix}$$

La matrice $L^{(0)}$ rappresenta solo i cammini di lunghezza e peso nulli, cioè esistono solo quelli da ogni vertice in se. Allo stesso modo, $L^{(1)}$ rappresenta i cammini di lunghezza al più 1 e quindi formati dagli archi di G, per cui abbiamo che $L^{(1)} = W$. Data la matrice $L^{(k)}$ possiamo anche calcolare la matrice dei padri $\Pi^{(k)}$: se esiste un cammino minimo da i a j di lunghezza al più k, cioè se $\ell_{ij}^{(k)} \neq \infty$, allora $p_{ij}^{(k)}$ è il padre di j in tale cammino.

$$L^{(2)} = L^{(1)} \cdot W = W^2 = \begin{pmatrix} 0 & 0 & -1 & 1 & 2 & 3 \\ \infty & 0 & \infty & \infty & 4 & \infty \\ \infty & 3 & 0 & \infty & 2 & 1 \\ \infty & -1 & -2 & 0 & 0 & -1 \\ \infty & \infty & \infty & \infty & 0 & \infty \\ \infty & 2 & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$\Pi^{(2)} = \begin{pmatrix} u & x & x & u & u & u \\ \infty & v & \infty & \infty & v & \infty \\ \infty & z & w & \infty & w & w \\ \infty & x & x & x & w & w \\ \infty & \infty & \infty & \infty & y & \infty \\ \infty & z & \infty & \infty & v & z \end{pmatrix}$$

Allora abbiamo che $\ell_{ij}^{(k)} = \min_t\{\ell_{it}^{(k-1)} + \ell_{tj}^{(k-1)}\}$: infatti per ottenere il cammino minimo da i a j, dobbiamo concatenare due cammini minimi per un vertice t di G. Infine se aggiorniamo il valore, cioè se $\ell_{ij}^{(k-1)} > \ell_{ij}^{(k)}$ allora poniamo $p_{ij}^{(k)} = p_{tj}^{(k-1)}$, dove t è un vertice che realizza il *nuovo* minimo.

$$L^{(4)} = W^2 \cdot W^2 = W^4 = \begin{pmatrix} 0 & 0 & -1 & 1 & 1 & 0 \\ \infty & 0 & \infty & \infty & 4 & \infty \\ \infty & 3 & 0 & \infty & 2 & 1 \\ \infty & -1 & -2 & 0 & 0 & -1 \\ \infty & \infty & \infty & \infty & 0 & \infty \\ \infty & 2 & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$\Pi^{(4)} = \begin{pmatrix} u & x & x & u & w & w \\ \infty & v & \infty & \infty & v & \infty \\ \infty & z & w & \infty & w & w \\ \infty & x & x & x & w & w \\ \infty & \infty & \infty & \infty & y & \infty \\ \infty & z & \infty & \infty & v & z \end{pmatrix}$$

$L^{(8)} = W^4 \cdot W^4 = W^8 = W^4$

L'esercizio prosegue facendo la seguente osservazione. Sfruttando l'associatività di questa operazione tra matrici e notando che la matrice $L^{(k)}$ si stabilizza per ogni $k \geq n-1$, cioè $L^{(k)} = L^{(k+1)}$, adottiamo una strategia che consiste nell'elevare a potenze successive. Tale modifica permette di ridurre il tempo di esecuzione di questo problema da $\mathcal{O}(|V|^4)$ a $\mathcal{O}(log(|V|)|V|^3)$.

Per il secondo grafo si usa invece l'algoritmo di *Floyd-Warshall*. L'idea è che invece di aumentare progressivamente la lunghezza dei cammini considerati, come è stato fatto per il grafo precedente, viene aumentato progressivamente l'insieme dei vertici di G che possono essere usati come vertici interni dei cammini. Allora, per ogni $k \geq 0$, l'elemento $d_{ij}^{(k)}$ nella la matrice $D^{(k)}$ rappresenta il peso minimo di un cammino dal vertice i al vertice j avente i vertici interni con indice non maggiore di k. Ovviamente $D^{(0)} = W$ perchè sono considerati solo i cammini senza vertici interni, cioè gli archi.

$$D^{(0)} = W = \begin{pmatrix} 0 & 3 & 2 & 1 & \infty \\ \infty & 0 & \infty & \infty & \infty \\ \infty & \infty & 0 & 1 & \infty \\ \infty & -2 & \infty & 0 & \infty \\ \infty & 3 & \infty & 2 & 0 \end{pmatrix} \quad \Pi^{(0)} = \begin{pmatrix} u & u & u & u & \infty \\ \infty & v & \infty & \infty & \infty \\ \infty & \infty & x & x & \infty \\ \infty & y & \infty & y & \infty \\ \infty & z & \infty & z & z \end{pmatrix}$$

Data la matrice $D^{(k-1)}$, per calcolare $D^{(k)}$ usiamo la seguente ricorsione: $d_{ij}^{(k)} = \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$. Infatti, o il cammino rappresentato da $d_{ij}^{(k)}$ non ha il vertice k come vertice interno e quindi posso usare $d_{ij}^{(k-1)}$, oppure ha il vertice k come interno e quindi concateno i cammini rappresentati da $d_{ik}^{(k-1)}$ e $d_{kj}^{(k-1)}$. Notiamo che $D^{(3)} = D^{(2)} = D^{(1)} = D^{(0)}$ e $\Pi^{(3)} = \Pi^{(2)} = \Pi^{(1)} = \Pi^{(0)}$ perché il vertice u non ha archi entranti e il vertice v non ha archi uscenti e quindi non possono essere usati come vertici interni di un cammino; inoltre l'unico cammino *nuovo* ammettendo x come vertice interno è quello da u a y passando per x con peso 3 che però è maggiore del peso dell'arco (u, y) (che è uno). Allora dobbiamo calcolare $D^{(4)}$ e $\Pi^{(4)}$.

$$D^{(4)} = \begin{pmatrix} 0 & -1 & 2 & 1 & \infty \\ \infty & 0 & \infty & \infty & \infty \\ \infty & -1 & 0 & 1 & \infty \\ \infty & -2 & \infty & 0 & \infty \\ \infty & 0 & \infty & 2 & 0 \end{pmatrix} \quad \Pi^{(4)} = \begin{pmatrix} u & y & u & u & \infty \\ \infty & v & \infty & \infty & \infty \\ \infty & y & x & x & \infty \\ \infty & y & \infty & y & \infty \\ \infty & y & \infty & z & z \end{pmatrix}$$

L'algoritmo finisce notando che $D = D^{(5)} = D^{(4)}$ e $\Pi = \Pi^{(5)} = \Pi^{(4)}$ perché il vertice z non ha archi entranti e quindi non può essere usato come vertice interno di un cammino. Ricordiamo che l'algoritmo di Floyd-Warshall ha tempo di esecuzione pari a $\mathcal{O}(|V|^3)$. □

Esercizio 2 (25.2-6, [1])**.** Modificare lo pseudo-codice dell'algoritmo di Floyd-Warshall per individuare, se esiste, un ciclo di peso negativo nel grafo in esame. Inoltre, fornire lo pseudo-codice di un algoritmo che, se esiste, ritorna la lunghezza minima di un ciclo di peso negativo con un costo computazionale di $\mathcal{O}(|V|^4)$.

Soluzione 2. La soluzione a questo esercizio consiste semplicemente nel aggiungere un controllo nel ciclo più interno: infatti, se in un qualsiasi momento dell'algoritmo il valore relativo al peso di un cammino minimo da un vertice v a se stesso viene aggiornato, e quindi diventa negativo, allora vuol dire che tale cammino contiene un ciclo negativo contenente v.

Algorithm 1 Algoritmo di Floyd-Warshall modificato per determinare se esiste un ciclo di peso negativo.

Input: $G = (V, E)$ grafo diretto e con archi pesat
Output: `TRUE` se G non ha cicli di peso negativo e `FALSE` altrimenti

```
1: function Floyd-Warshall(W)
2:     n = |W| ↤ numero di righe di W o numero di vertici di G
3:     D^(0) = W
4:     for k = 1, ..., n do
5:         for i = 1, ..., n do
6:             for j = 1, ..., n do
7:                 d_ij^(k) = min{d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1)}
8:                 if d_ii^(k) < 0 then
9:                     return (D^(n), FALSE)
10:    return (D^(n), TRUE)
```

Se questo evento accade allora possiamo interrompere subito e ritornare

la matrice calcolata D e `FALSE`, indicando che esiste un ciclo negativo in G. Altrimenti, l'algoritmo procede identicamente a Floyd-Warshall.

Per risolvere la seconda parte dell'esercizio, cioè quella relativa a determinare la lunghezza minima di un ciclo negativo, notiamo che non è utile usare l'algoritmo di Floyd-Warshall: infatti, dopo aver determinato l'esistenza di un tale ciclo e di tutti i vertici inclusi in cicli negativi, non c'è un metodo efficiente per determinare quanti di questi vertici compongono un ciclo negativo di lunghezza minima.

La soluzione che qui viene proposta usa il metodo che simula il prodotto di matrici (usato per risolvere l'Esercizio 1) per il calcolo dei cammini minimi tra tutte le coppie di vertici del grafo, senza usare il *trucco* dell'elevazione a potenze successive. Sia $k \geq 1$ il minimo intero per cui esiste $i \in \{1, \dots |V|\}$ con $\ell_{ii}^{(k)} < 0$, allora k è la lunghezza minima di un ciclo negativo. □

Esercizio 3 (25.2-8, [1])**.** Dato un grafo diretto $G = (V, E)$, la chiusura transitiva di G è un grafo $G^* = (V, E^*)$ tale che $(u, v) \in E^*$ se e solo se esiste un cammino da u a v in G. Fornire un algoritmo per calcolare la chiusura transitiva di un grafo diretto $G = (V, E)$ in maniera che il tempo di esecuzione sia $\mathcal{O}(|V| \cdot |E|)$.

Soluzione 3. La chiusura transitiva di un grafo può anche essere calcolata a partire dall'algoritmo di Floyd-Warshall che però ha tempo di esecuzione $\mathcal{O}(|V|^3)$. E' chiaro che per questo problema, Floyd-Warshall esegue dei calcoli non necessari: infatti è sufficiente verificare se per ogni coppia di vertici esiste un cammino che li collega, e non trovare il cammino di peso minimo. Allora si esegue una DFS radicata in s, per ogni vertice $s \in V$, e si salva l'insieme dei vertici raggiungibili da s: in questa maniera si costruisce una matrice di raggiungibilità che rappresenta la matrice di adiacenza di G^*. Quindi il tempo di esecuzione è pari a $\mathcal{O}(|V| \cdot (|V| + |E|)) = \mathcal{O}(|V| \cdot |E|)$. □

Esercizio 4 (M. Lauria)**.** Si considera una griglia $n \times n$ con $n > 0$. Un cammino su questa griglia deve partire dalla cella di coordinate $(0, 0)$ in alto a sinistra e deve arrivare alla posizione di coordinate $(n-1, n-1)$ in basso a destra. E' possibile muoversi solo su celle adiacenti, andando di un passo verso il basso o di un passo verso destra. Inoltre, sono vietati i cammini che toccano le celle di coordinate (i, j) con $i > j$, cioè non è permesso andare sotto la diagonale che va da $(0, 0)$ a $(n-1, n-1)$. Fornire in pseudo-codice un algoritmo che calcoli il numero di cammini validi con un tempo di esecuzione $\mathcal{O}(n^2)$.

Soluzione 4. Si costruisce una matrice M di dimensioni $n \times n$ tale che alla fine dell'algoritmo, il valore di $M[i, j]$ indica il numero di cammini validi dalla cella di coordinate $(0, 0)$ alla cella di coordinate (i, j), sfruttando la ricorsione. Iniziamo con i casi base; se la cella di coordinate (i, j) si trova

- sulla prima riga, cioè $i = 0$, allora $M[i, j] = 1$ perché la si può raggiungere solo facendo passi verso destra e quindi in un unico modo;
- sulla diagonale e non è la cella di partenza, cioè $0 < i = j$, allora $M[i, j] = M[i-1, j]$ perché la si può solo raggiungere solo dalla casella consecutiva in alto;
- sotto la diagonale, cioè se $i > j$, allora $M[i, j] = 0$ perché non è possibile raggiungere tale cella con cammini validi.

Supponiamo infine che la cella in esame non soddisfa nessuna delle condizioni precedenti, allora $M[i, j] = M[i-1, j]+M[i, j-1]$ perché è possibile raggiungere questa cella sia da quella consecutiva in alto che da quella consecutiva a sinistra.

Algorithm 2 Algoritmo per calcolare il numero di cammini validi sopra la diagonale.

Input: intero n
Output: numero di cammini tra le due caselle

```
1:  function PathsAboveDiagonal(n)
2:      for i = 0, ..., n − 1 do
3:          for j = i, ..., n − 1 do
4:              if i = 0 then
5:                  M[i, j] = 1
6:              else if i = j and i ≥ 1 then
7:                  M[i, j] = M[i − 1, j]
8:              else
9:                  M[i, j] = M[i − 1, j] + M[i, j − 1]
10:     return M[n − 1, n − 1]
```

Un esempio del codice richiesto è dato dall'Algoritmo 2. La soluzione si conclude facendo una rapida analisi del costo computazionale: in pratica, l'algoritmo proposto consiste in due cicli **for**, uno contenuto nell'altro, in cui le operazioni interne si svolgono in $\mathcal{O}(1)$ e quindi complessivamente abbiamo $\mathcal{O}(n^2)$. □

References

[1] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. Introduction to algorithms. 2022.