Azzolini Riccardo 2019-05-22

Programmazione dinamica

1 Chiusura transitiva di un grafo

- *Input*: un grafo orientato $G = \langle V, E \rangle$;
- *Output*: un grafo $G^\star = \langle V, E^\star \rangle$, tale che $(u, v) \in E^\star$ se e solo se esiste un cammino da u a v in G.

1.1 Soluzione ricorsiva

Per un grafo con n nodi $(v_1, \ldots, v_n)$, dati gli indici $i, j, k \leq n$, si definiscono

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

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

k è un vincolo che determina quali nodi è possibile attraversare nel cammino da i a j:

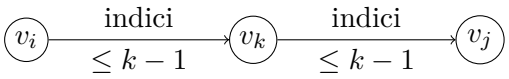
- con il vincolo più ristretto, $k = 0$, non è possibile attraversare altri nodi, quindi si considerano solo cammini formati da singoli lati, ovvero C_{ij}^0 corrisponde alla matrice di adiacenza del grafo G;
- con il vincolo più ampio possibile, $k = n$, è consentito il passaggio da tutti i nodi, quindi $C_{ij}^n = 1$ se e solo se esiste un cammino qualsiasi da i a j.

Di conseguenza, calcolando $C_{ij}^n \quad \forall i, j$ si risolve il problema, poiché si ricava la matrice di adiacenza della chiusura transitiva $G^\star$.

Un cammino da i a j che passa per nodi di indice $\leq k$ può esistere perché

- esiste un cammino che passa per i nodi di indice $\leq k - 1$ (compreso il caso in cui esiste il lato (v_i, v_j), che forma da solo tale cammino), oppure

- esiste un cammino che passa dal nodo v_k: siccome non si transita più volte dallo stesso nodo, v_k viene incontrato una sola volta, quindi questo cammino è sicuramente composto da due cammini che collegano v_i a v_k e v_k a v_j, entrambi passanti solo per nodi di indici $\leq k-1$.



Vale quindi l'equazione di ricorrenza

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

1.2 Implementazione

Dall'equazione di ricorrenza, potrebbe sembrare che sia necessario utilizzare due matrici di ordine $n \times n$: una per C_{ij}^{k-1} e una per C_{ij}^{k}, in modo da poter costruire a ogni passo la nuova matrice in base allo stato precedente, senza rischiare di modificare quest'ultimo.

Si osserva, però, che:

- $C_{ij}^{k-1} = 0$ può diventare $C_{ij}^{k} = 1$ solo se vale $C_{ik}^{k-1} \wedge C_{kj}^{k-1}$;
- gli 1 nella matrice non vengono più modificati;
- $C_{ik}^{k-1} = C_{ik}^{k}$ e $C_{kj}^{k-1} = C_{kj}^{k}$, perché il nodo che diventa disponibile al passaggio k è v_k, ma esso non può essere un nodo intermedio nei cammini da v_i a v_k e da v_k a v_j, quindi l'esistenza di tali cammini non può cambiare rispetto al passaggio $k-1$.

Per questi motivi, è sufficiente una singola matrice $n \times n$, e qualsiasi modo di scandirne gli elementi costituisce un ordine totale valido.

```
Chiusura(V, E) {
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
            if ((i, j) appartiene a E)
                C[i][j] = 1;
            else
                C[i][j] = 0;
        }
    }

    for (k = 1; k <= n; k++) {
        for (i = 1; i <= n; i++) {
            for (j = 1; j <= n; j++) {
                if (C[i][j] == 0)
```

```
                    C[i][j] = C[i][k] && C[k][j];
            }
        }
    }

    return C;
}
```

1. I primi cicli `for` annidati calcolano $C_{ij}^0 \quad \forall i, j$.
2. Per $k = 1, \ldots, n$, nel corpo del secondo `for` si calcola $C_{ij}^k \quad \forall i, j$, applicando l'equazione di ricorrenza.

1.2.1 Complessità

- Spazio $\Theta(n^2)$, per la matrice di ordine $n \times n$.
- Tempo $\Theta(n^3)$, dovuto ai tre cicli `for` annidati, che hanno costo

$$\sum_{k=1}^{n}\sum_{i=1}^{n}\sum_{j=1}^{n} 1 = \sum_{k=1}^{n}\sum_{i=1}^{n} n = \sum_{k=1}^{n} n^2 = n^3$$

  (mentre si trascura il costo $\Theta(n^2)$ per il calcolo iniziale di C_{ij}^0).

2 Cammini minimi

- *Input*: un grafo $G = \langle V, E \rangle$ orientato e pesato, con una funzione peso $w : E \to \mathbb{Q}^+$.
- *Output*: per ogni coppia di nodi (u, v), un cammino di costo minimo da u a v e il relativo costo.

Osservazione: A differenza dell'algoritmo di Dijkstra, che calcola i cammini minimi da un unico nodo sorgente a tutti gli altri, in questo caso si vogliono considerare tutte le coppie di nodi.

La soluzione di entrambi questi problemi non è ben definita se esistono cicli di costo negativo. Per semplicità, si considera quindi una funzione peso che assume solo valori non negativi.

2.1 Soluzione ricorsiva

L'obiettivo è calcolare

$$C[i][j] = \begin{cases} r & \text{se } r \text{ è il costo di un cammino minimo da } v_i \text{ a } v_j \\ \infty & \text{se non esiste un cammino da } v_i \text{ a } v_j \end{cases}$$

$$P[i][j] = \begin{cases} k & \text{se } v_k \text{ precede } v_j \text{ nel cammino minimo da } v_i \text{ a } v_j \\ \perp & \text{se non esiste un cammino da } v_i \text{ a } v_j \end{cases}$$

A tale scopo, si usa una strategia molto simile a quella adottata per la chiusura transitiva, definendo

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

$$C_{ij}^k = \begin{cases} r & \text{se } r \text{ è il costo di un cammino minimo } v_i \text{ a } v_j \\ & \text{che passa solo per nodi di indice} \leq k \\ \infty & \text{altrimenti} \end{cases}$$

e applicando l'equazione di ricorrenza

$$C_{ij}^k = \min\{C_{ij}^{k-1}, C_{ik}^{k-1} + C_{kj}^{k-1}\}$$

dove $C_{ik}^{k-1} + C_{kj}^{k-1}$ è il peso di un possibile nuovo cammino passante per v_k, che potrebbe essere più corto di altri cammini eventualmente già noti.

2.2 Implementazione

Come per la chiusura transitiva, si ha che $C_{ik}^{k-1} = C_{ik}^k$ e $C_{kj}^{k-1} = C_{kj}^k$, perché v_k è già presente (come primo/ultimo nodo) nei cammini da v_i a v_k e da v_k a v_j al passo $k-1$, e transitando da v_k più di una volta il costo potrebbe solo aumentare.

Sono quindi sufficienti due matrici $n \times n$ (la matrice C dei costi e la matrice P dei nodi precedenti), e le si può aggiornare in qualsiasi ordine.

```
MinPath(V, E, w) {
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
            if (i == j) {
                C[i][j] = 0;
                P[i][j] = i;
            } else if ((i, j) appartiene a E) {
                C[i][j] = w(i, j);
                P[i][j] = i;
            } else {
                C[i][j] = MAX_INT;
                P[i][j] = UNDEFINED;
            }
        }
    }

    for (k = 1; k <= n; k++) {
        for (i = 1; i <= n; i++) {
            for (j = 1; j <= n; j++) {
                if (C[i][k] + C[k][j] < C[i][j]) {
                    C[i][j] = C[i][k] + C[k][j];
                    P[i][j] = P[k][j];
                }
            }
        }
    }

    return (C, P);
}
```

Quando viene trovato un cammino da v_i a v_j passante per v_k, il predecessore di v_j in tale cammino ($P[i][j]$) corrisponde al suo predecessore nel cammino da v_k a v_j ($P[k][j]$).

Dato un grafo di n nodi, quest'implementazione richiede:

- spazio $\Theta(n^2)$, per le due matrici $n \times n$;
- tempo $\Theta(n^3)$.