Appunti lezione – Capitolo 13
Programmazione dinamica

Alberto Montresor

12 Novembre, 2015

1 Domanda: Fattore di crescita dei numeri catalani

Vogliamo dimostrare che

$$P(n) = \sum_{k=1}^{n-1} P(k)P(n-k)$$

cresce almeno come 2^n.
La nostra ipotesi induttiva è quindi:

$$P(i) \geq 2^i$$

per tutti gli $i < n$.
Quindi otteniamo:

$$P(n) \geq \sum_{k=1}^{n-1} 2^k 2^{n-k} = (n-1)2^n \geq 2^n$$

2 Domanda: Complessità di recPar

- Sia $T(n)$ il tempo impiegato da recPar per un input di dimensione n.
- Il costo di esecuzione di recPar (a parte le chiamate ricorsive) è $O(1)$.

Si verifica la seguente relazione sulla ricorrenza:

$$\begin{aligned} T(1) &\geq 1 \\ T(n) &\geq 1 + \sum_{k=1}^{n-1} (T(n) + T(n-k) + 1) \end{aligned}$$

Si noti che per $i = 1, 2, \ldots, n-1$, ogni termine $T(i)$ appare una volta come $T(k)$ e una volta come $T(n-k)$.
Raccogliendo i termini, otteniamo:

$$T(n) \geq n + 2\sum_{i=1}^{n-1} T(i)$$

Utilizziamo il metodo di sostituzione per dimostrare che $T(n) \geq 2^{n-1}$ per ogni $n \geq 1$.

$$\begin{aligned} T(n) &\geq n + 2\sum_{i=1}^{n-1} T(i) \\ &\geq n + 2\sum_{i=1}^{n-1} 2^{i-1} \\ &= n + 2\sum_{i=0}^{n-2} 2^{i} \\ &= n + 2(2^{n-1} - 1) \\ &= n + 2^n - 2 \\ &\geq 2^{n-1} \end{aligned}$$

L'ultima disequazione può essere semplificata come

$$n + 2^{n-1} - 2 \geq 0$$

che è vera per qualunque $n > 1$.
Per il passo base: $T(1) \geq 1 = 2^0$.
Abbiamo dimostrato che il numero di chiamate ricorsive è almeno esponenziale.

3 Domanda: Quante sono le possibili sottosequenze?

Se $|X| = m$ (la lunghezza di X è m), allora ognuno dei dei caratteri puó apparire o non apparire nell'insieme. Quindi l'insieme può essere rappresentato da una stringa binaria lunga m bit; ci sono 2^m combinazioni diverse.

4 Cammini minimi fra tutte le coppie

Problema dei cammini minimi fra tutte le coppie: ad esempio, quando si vuole creare la tabella con le distanze fra tutte le città di un atlante stradale.
Per risolvere questo problema, possiamo utilizzare le soluzioni per i cammini minimi da singola sorgente per ogni vertice del grafo.

- In caso di archi non negativi, possiamo utilizzare uno dei seguenti algoritmi:
  - Dijkstra – Complessità $O(n \cdot n^2) = O(n^3)$;
  - Johnson – Complessità $O(n \cdot m \log n)$ = $O(mn \log n)$;
  - Fredman-Tarjan – Complessità $O(n \cdot (m + n \log n)) = O(mn + n^2 \log n)$.
- In caso di archi negativi, possiamo utilizzare Bellman-Ford con un costo $O(n \cdot mn) = O(mn^2)$, che corrisponde a n^4 nel caso peggiore (grafi non sparsi).

È possibile fare di meglio:

- L'algoritmo di Floyd e Warshall permette di risolvere il problema in tempo $O(n^3)$ con archi negativi (ovviamente senza cicli negativi). Nota: sembrerebbe che l'algoritmo funzioni solo con matrice di adiacenza; l'idea è invece che alla fine si produrrà una matrice delle distanze che inizialmente è una matrice dei pesi degli archi. Ma il costo per trasformare una rappresentazione a lista in una rappresentazione a matrice è $O(n^2)$
- Utilizzando una rappresentazione a liste di adiacenza, l'algoritmo di Johnson permette di risolvere il problema nelle complessità viste sopra.

5 Algoritmo di Floyd e Warshall

Questo algoritmo è basato su una tecnica di programmazione dinamica.

Definizione 1. Sia k un valore in $\{1, \ldots, n\}$. Diciamo che un cammino p^k_{xy} è un *cammino minimo k-vincolato* fra x e y se esso ha il costo minimo fra tutti i cammini fra x e y che non passano per nessun vertice in $v_k, \ldots, v_n$ (x e y sono esclusi dal vincolo).

Definizione 2. Se esiste un cammino minimo k-vincolato p^k_{xy} fra x e y, definiamo $d^k(x, y) = w(p^k_{xy})$; nel caso tale cammino non esista, $d^k(x, y) = +\infty$.

Come condizioni iniziali, abbiamo che:

- $d^1(x, y) = w(x, y)$ se $(x, y) \in E$;
- $d^1(x, y) = +\infty$ se $(x, y) \notin E$;

In altre parole, la matrice iniziale corrisponde alla matrice di adiacenza pesata del grafo; questa può essere presente in input, oppure può essere calcolata.

Teorema 1 (Sottostruttura ottima)**.** Ogni sottocammino di un cammino minimo k-vincolato in G è esso stesso un cammino minimo k-vincolato in G.

Dimostrazione. Sia G_k il sottografo ottenuto da G rimuovendo tutti i vertici $v_{k+1}, \ldots, v_n$ diversi da x e y. Chiaramente, un cammino in G è minimo k-vincolato se e solo se è minimo in G_k. Il teorema segue quindi dalla sottostruttura ottima dei cammini minimi. □

Floyd, basandosi su un teorema precedente di Warshall, osservò che è possibile costruire d^k_{xy} in questo modo:

Lemma 1 (Relazione tra distanze vincolate)**.** Per ogni $k \in \{1, \ldots, n\}$ e per ogni $x, y \in V$, si ha:

$$d^{k+1}(x, y) = \min\{d^k(x, y), d^k(x, v_k) + d^k(v_k, y)\}$$

Dimostrazione. Sia p^k_{xy} un cammino minimo $k+1$-vincolato fra x e y. Ci sono due possibilità:

- se v_k non è un vertice interno di p^k_{xy}, allora questo cammino è anche k-vincolato; quindi
$$d^{k+1}(x, y) = d^k(x, y)$$
- se v_k è un vertice interno, allora il cammino può essere spezzato in due sottocammini $p^k_{xv_k}$ e $p^k_{v_k y}$; per la sottostruttura dei cammini minimi k-vincolati, sono anch'essi cammini minimi k-vincolati; inoltre, poiché entrambi i cammini non contengono v_k come vertice interno, sono anche cammini k-vincolati.
$$d^{k+1}(x, y) = d^k(x, v_k) + d^k(v_k, y)$$

Ovviamente, il valore inferiore fra i due è quello che corrisponde al cammino minimo. □

Grazie a questa relazione, possiamo utilizzare la programmazione dinamica per risolvere il problema. Notate che memoization non va bene in questo caso, perchè vogliamo calcolare la lunghezza di tutti i cammini.

5.1 Algoritmo

Complessità: Ovviamente, la complessità è $O(n^3)$.

5.2 Costruzione dei cammini

Al solito, abbiamo ottenuto i pesi, ma dobbiamo tenere conto anche dei cammini. Nelle versioni "a sorgente singola", ogni nodo conosceva un singolo predecessore dell'albero dei cammini minimi; e c'era un unico albero di cammini minimi. Qui, ogni nodo conosce n diversi predecessori, uno per ogni sorgente. Parliamo quindi di matrice dei predecessori p di dimensione $n \times n$, dove $p[x, y] = 0$ se non esiste cammino da x a y, altrimenti rappresenta il predecessore di y nel cammino minimo con sorgente x.

```
floydWarshall(GRAPH G, integer[][] d, integer[][] p)
    foreach u, v ∈ G.V() do
        d[u, v] ← +∞
        p[u, v] ← 0
    foreach u ∈ G.V() do
        foreach v ∈ G.adj(u) do
            d[u, v] ← w(u, v)
            p[u, v] ← u
    for k ← 1 to n do
        foreach u ∈ G.V() do
            foreach v ∈ G.V() do
                if d[u, k] + d[k, v] < d[u, v] then
                    d[u, v] ← d[u, k] + d[k, v]
                    p[u, v] ← p[k, v]
```

6 Chiusura transitiva di un grafo orientato

Dato un grafo orientato $G = (V, E)$, la chiusura transitiva $G^* = (V, E^*)$ è definita nel modo seguente:

$$E^* = \{ (x, y) \mid \text{esiste un cammino da } x \text{ a } y \text{ in } G \}$$

Esempio di utilizzo della chiusura transitiva? Considerate uno foglio di calcolo; esiste un grafo sottostante al foglio, dato dalle relazioni fra celle. Quando una cella viene modificata, bisogna aggiornare tutte le celle che dipendono da quella cella. Avere la chiusura transitiva permette di individuare rapidamente quali celle modificare.
Come ottenere la chiusura transitiva di un grafo? Assegnando un peso uno a tutti gli archi, possiamo utilizzare Floyd-Warshall; esiste un cammino fra x, y se $d[x, y]$ è un numero finito.
Altrimenti, è possibile utilizzare direttamente la matrice binaria di adiacenza, risparmiando spazio. Se m è la matrice di adiacenza, abbiamo che

$$\begin{aligned} m^1[x, y] &= m[x, y] \\ m^{k+1}[x, y] &= m[x, y] \vee (m^k[x, v_k] \wedge m^k[v_k, y]) \end{aligned}$$

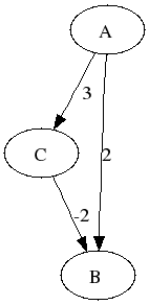
7 Algoritmo di Johnson

Se il grafo contiene solo pesi positivi, ripetendo n volte l'algoritmo di Johnson, si ottiene e utilizzando Heap di Fibonacci si ottiene la complessità prevista.
Se il grafo contiene pesi negativi, si utilizza una tecnica detta **ricalcolo dei pesi** per rimuovere i pesi negativi dai grafi. L'idea è utilizzare una funzione peso $\hat{w}$ tale per cui esiste un cammino minimo p in G con $\hat{w}$ se e solo se p è un cammino minimo anche per G con w. Costruiremo $\hat{w}$ in modo da avere solo pesi positivi.
Possiamo semplicemente prendere il valore minimo dei pesi sugli archi e sommarlo a tutti gli archi? Si veda la figura 7. Qual è il cammino minimo da A a B nella parte a)? Qual è il cammino minimo da A a B nella parte b)? Ovviamente non funziona - bisogna ripensare l'approccio.
Sia $w : E \to \mathbf{R}$ una funzione di peso sugli archi; sia $h : V \to \mathbf{R}$ una funzione di peso sui nodi. Definiamo:

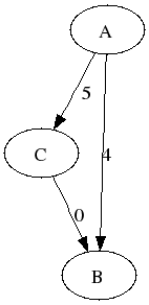
$$\hat{w}(x, y) = w(x, y) + h(x) - h(y)$$

Lemma 2. Sia $p = v_1, v_2, \ldots, v_k$ un cammino qualsiasi dal vertice v_1 al vertice v_k. Allora p è un cammino minimo in G con la funzione peso w se e solo se p è un cammino minimo in G con la funzione peso $\hat{w}$.

a)



b)



Dimostrazione. Iniziamo dimostrando che $\hat{w}(p) = w(p) + h(v_1) - h(v_k)$. Infatti,

$$\begin{aligned}
\hat{w}(p) &= \sum_{i=2}^{k} \hat{w}(v_{i-1}, v_i) \\
&= \sum_{i=2}^{k} (w(v_{i-1}, v_i) + h(v_{i-1}) - h(v_i)) \\
&= w(p) + \sum_{i=2}^{k} (h(v_{i-1}) - h(v_i)) \\
&= w(p) + h(v_1) - h(v_k)
\end{aligned}$$

L'ultimo passaggio segue dal fatto che tutti i pesi sui nodi intermedi si annullano. Poichè tutti i cammini fra v_1 e v_k hanno lo stesso "offset" $h(v_1) - h(v_k)$, ne segue che il lemma. □

7.1 Produrre pesi non negativi

Vogliamo fare in modo che i pesi di tutti gli archi sia non negativi, ovvero che $\hat{w}(x, y) \geq 0$.
Definiamo il problema in questo modo: aggiungiamo un nodo $s \notin V$, con archi uscenti da s e incidenti su tutti i nodi v. Sia $w(s, v) = 0$ per tutti i nodi v in V. Sia $h(v) = d(s, v)$ per tutti i nodi v in V.
Per la disuguaglianza triangolare, per ogni arco $(x, y) \in V$ abbiamo che:

$$\begin{aligned}
d(s, x) + w(x, y) &\geq d(s, y) \Leftrightarrow \\
h(x) + w(x, y) &\geq h(y) \Leftrightarrow \\
\hat{w}(x, y) = w(x, y) + h(x) - h(y) &\geq 0
\end{aligned}$$

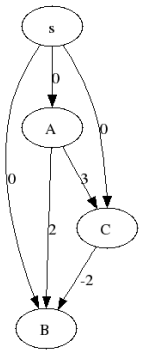
7.2 Algoritmo

Invece di descrivere lo pseudocodice, descriviamo i vari passi. Assumiamo l'esistenza di archi negativi. L'algoritmo può essere modificato per rilevare cicli negativi.

- Costruiamo un grafo G' aggiungendo il nodo s e gli archi relativi: costo $O(n)$ (per l'aggiunta degli archi).
- Utilizziamo Bellman-Ford su G' per ottenere i costi minimi di tutti i nodi a partire da s; costo $O(mn)$. Questi valori vengono utilizzati come funzione h.
- Calcolare la nuova funzione dei pesi $\hat{w}$, utilizzando i valori h; costo $O(m + n)$ o $O(n^2)$
- Per tutti i nodi $x \in V$:
  - Utilizziamo Johnson su G', $\hat{w}$: $O(m \log n)$
  - $d[x, y] = \hat{d}(x, y) + h(y) - h(x)$

Il costo totale è quindi $O(mn \log n)$, che è meglio di $O(n^3)$ per grafi sparsi.
Per un esempio, si veda figura 7.2.

a)



b)