CI2613: Algoritmos y Estructuras III

Blai Bonet

Universidad Simón Bolívar, Caracas, Venezuela

Enero-Marzo 2015

Caminos de costo mínimo en grafos

Caminos entre todos los pares de vértices

Estamos interesados en calcular distancias y caminos más cortos entre todos los pares de vértices

Podemos ejecutar Bellman-Ford desde cada vértice s en un tiempo $O(V^2E)$, ya que Bellman-Ford requiere $O(VE)$ unidades de tiempo

Con pesos no-negativos, se utiliza Dijkstra en lugar de Bellman-Ford. El tiempo dependerá de la implementación de la cola de prioridad:

– **Heap binario:** $O(VE \log V)$ unidades de tiempo

– **Heap de Fibonacci:** $O(VE + V^2 \log V)$ unidades de tiempo

El heap de Fibonacci hace diferencia cuando $O(V^2 \log V)$ es "menor" a $O(VE \log V)$; i.e. cuando $|V| = o(E)$

Caminos entre todos los pares de vértices

La salida de un algoritmo para calcular las distancias y caminos más cortos entre todos los pares de vértices es:

– una **matriz D de distancias** donde la entrada d_{ij} es $\delta(i,j)$

– una **matrix Π de predecesores** tal que el grafo $G_i = (V_i, E_i)$ con $V_i = \{j : \pi_{ij} \neq \texttt{null}\} \cup \{i\}$ y $E_i = \{(\pi_{ij}, j) : j \in V_i \setminus \{i\}\}$ es un **árbol de caminos más cortos relativo al vértice i**

Los algoritmos asumen un grafo dado por una **matriz W de pesos**: w_{ij} es el peso de la arista (i,j) (con $w_{jj} = 0$ y $w_{ij} = \infty$ si no existe la arista)

Esto no causa problema ya que el algoritmo debe invertir tiempo $\Omega(V^2)$ para generar la salida

Primer algoritmo: Programación dinámica

Sea $\ell_{ij}^{(m)}$ el **costo del mejor camino** del vértice i al vértice j que tiene a lo **sumo m aristas**

$$\ell_{ij}^{(0)} = \begin{cases} 0 & \text{si } i = j \\ \infty & \text{si } i \neq j \end{cases}$$

El mejor camino con a lo sumo m aristas es: el mejor camino con a lo sumo $m-1$ aristas ó un camino con m aristas

$$\ell_{ij}^{(m)} = \min \left\{ \ell_{ij}^{(m-1)}, \min_{1 \leq k \leq n} \left(\ell_{ik}^{(m-1)} + w_{kj} \right) \right\}$$

Por definición, $\delta(i,j) = \ell_{ij}^{(n-1)} = \ell_{ij}^{(m)}$, para $m > n-1$, ya que el mejor camino tiene $\leq n-1$ aristas (si no hay ciclos de costo negativo)

Primer algoritmo: Programación dinámica

Observe:

$$\begin{aligned} \ell_{ij}^{(m)} &= \min \left\{ \ell_{ij}^{(m-1)}, \min_{1 \leq k \leq n} \left(\ell_{ik}^{(m-1)} + w_{kj} \right) \right\} \\ &= \min_{1 \leq k \leq n} \left\{ \ell_{ik}^{(m-1)} + w_{kj} \right\} \end{aligned}$$

ya que $w_{jj} = 0$ por definición

El algoritmo computa matrices $L^{(1)}, L^{(2)}, L^{(3)}, \ldots, L^{(n-1)}$ tal que $L^{(m)} = (\ell_{ij}^{(m)})$

Primer algoritmo: Subrutina central

$$\ell_{ij}^{(m)} = \min_{1 \leq k \leq n} \left\{ \ell_{ik}^{(m-1)} + w_{kj} \right\}$$

```
Matriz Extender-Caminos(L, W):
    n = nrows(L)
    L’ = new Matriz(n,n)
    for i = 1 to n
        for j = 1 to n
            L’[i,j] = ∞
            for k = 1 to n
                L’[i,j] = min { L’[i,j], L[i,k] + W[k,j] }
    return L’
```

`Extender-Caminos` toma tiempo $\Theta(n^3)$

Primer algoritmo: Pseudocódigo

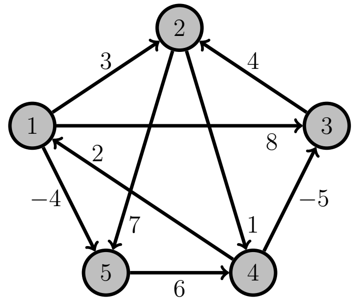
El algoritmo computa las matrices $L^{(1)}, L^{(2)}, L^{(3)}, \ldots, L^{(n-1)}$

```
void Calcular-Distancias(W):
    n = nrows(W)
    L(1) = W
    for m = 2 to n - 1
        L(m) = Extender-Caminos(L(m-1), W)
    return L(n-1)
```

`Calcular-Distancias` toma tiempo $\Theta(n^4)$

$\Theta(n^4)$ iguala el tiempo $\Theta(V^2 E)$ para n corridas de Bellman-Ford en **grafos densos** (i.e. $|E| = \Theta(V^2)$)

Primer algoritmo: Ejemplo



$$L^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$L^{(2)} = \begin{pmatrix} 0 & 3 & 8 & \mathbf{2} & -4 \\ \mathbf{3} & 0 & \mathbf{-4} & 1 & 7 \\ \infty & 4 & 0 & \mathbf{5} & \mathbf{11} \\ 2 & \mathbf{-1} & -5 & 0 & \mathbf{-2} \\ \mathbf{8} & \infty & \mathbf{1} & 6 & 0 \end{pmatrix}$$

$$L^{(4)} = \begin{pmatrix} 0 & \mathbf{1} & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

$$L^{(3)} = \begin{pmatrix} 0 & 3 & \mathbf{-3} & 2 & -4 \\ 3 & 0 & -4 & 1 & \mathbf{-1} \\ \mathbf{7} & 4 & 0 & 5 & \mathbf{3} \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \mathbf{5} & 1 & 6 & 0 \end{pmatrix}$$

Relación con multiplicación de matrices

La operación implementada por `Extender-Caminos(A, B)` está relacionada con la **multiplación de matrices**

Defina $A \otimes B =$ `Extender-Caminos(A, B)` para cualquier par de matrices A, B de dimensión $n \times n$

Lema

Para cualquier $i, j \geq 0$, $L^{(i)} \otimes L^{(j)} = L^{(i+j)}$

Prueba: por inducción en j (para cualquier i fijo)

Base $j = 0$: $L^{(i)} \otimes L^{(0)} = L^{(i)}$ (ejercicio)

Caso $j > 0$:

$$\begin{aligned} L^{(i)} \otimes L^{(j)} &= L^{(i)} \otimes (L^{(j-1)} \otimes W) = (L^{(i)} \otimes L^{(j-1)}) \otimes W \\ &= L^{(i+j-1)} \otimes W = L^{(i+j)} \end{aligned}$$

$\square$

Relación con multiplicación de matrices

Corolario

Para cualquier $m \geq 0$, $L^{(2m)} = L^{(m)} \otimes L^{(m)}$

Por ejemplo,

$$\begin{aligned} L^{(2)} &= L^{(1)} \otimes L^{(1)} = W \otimes W \\ L^{(4)} &= L^{(2)} \otimes L^{(2)} = W \otimes W \otimes W \otimes W \\ L^{(8)} &= L^{(4)} \otimes L^{(4)} = W \otimes W \otimes W \otimes W \otimes W \otimes W \otimes W \otimes W \\ L^{(16)} &= L^{(8)} \otimes L^{(8)} = W \otimes \cdots \otimes W \end{aligned}$$

Segundo algoritmo: Pseudocódigo

El algoritmo computa las matrices $L^{(1)}, L^{(2)}, L^{(4)}, \ldots, L^{(2m)}$ con $2m > n - 1$

```
bool Calcular-Distancias-II(W):
    n = nrows(W)
    L(1) = W
    m = 1
    while m < n - 1
        L(2m) = Extender-Caminos(L(m), L(m))
        m = 2m
    return L(m/2)
```

`Calcular-Distancias-II` toma tiempo $\Theta(n^3 \log n)$ que es **mucho mejor** a $\Theta(n^4)$

Computar caminos más cortos

Hemos visto como computar la matriz $D = L^{(n-1)}$ de distancias en un grafo con n vertices

Caminos más cortos entre todos los pares de vértices se pueden calcular en tiempo $\Theta(n^3)$ a partir de D de la siguiente forma:

```
bool Calcular-Caminos(D, W):
    n = nrows(W)
    Π = new Matriz(n,n)
    for i = 1 to n
        for j = 1 to n
            for k = 1 to n
                if D[i,j] == D[i,k] + W[k,j]
                    Π[i,j] = k
    return Π
```


Algoritmo de Floyd-Warshall

También basado en **programación dinámica** pero con una formulación diferente

Floyd-Warshall toma tiempo $\Theta(n^3)$ que es una mejora sobre $\Theta(n^3 \log n)$

En lugar de considerar caminos de i a j con a lo sumo k aristas, F-W considera caminos de i a j que pasan sólo sobre $\{1, 2, \ldots, k\}$

Defina:

$d_{ij}^{(k)}$ = costo mejor camino de i a j que pasa sólo sobre $\{1, \ldots, k\}$

Por definición, $d_{ij}^{(n)} = \delta(i, j)$ y $d_{ij}^{(0)} = w_{ij}$

Programación dinámica de Floyd-Warshall

Consider un mejor camino de i a j que pasa sobre $\{1, 2, \ldots, k\}$. Existen dos posibilidades:

– el camino **no visita** el vértice k (i.e. pasa sobre $\{1, 2, \ldots, k-1\}$), ó

– el camino **visita** el vértice k

Como todo camino óptimo no tiene ciclos, en el segundo caso se visita k una sóla vez

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{si } k = 0 \\ \min\left\{ d_{ij}^{(k-1)},\ d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \right\} & \text{si } k > 0 \end{cases}$$

Floyd-Warshall: Pseudocódigo

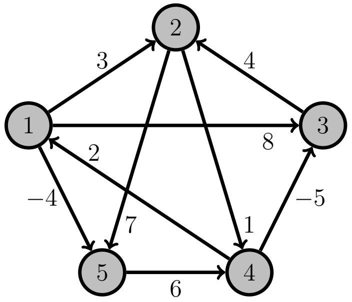
Se computan las matrices $D^{(m)} = (d_{ij}^{(m)})$ para $m = 1, 2, \ldots, n$

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{si } k = 0 \\ \min\left\{ d_{ij}^{(k-1)},\ d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \right\} & \text{si } k > 0 \end{cases}$$

```
Matriz Floyd-Warshall(W):
    n = nrows(L)
    D(0) = W
    for k = 1 to n
        D(k) = new Matriz(n,n)
        for i = 1 to n
            for j = 1 to n
                D(k) = min { D(k-1)[i,j], D(k-1)[i,k] + D(k-1)[k,j] }
    return D(n)
```

`Floyd-Warshall` toma tiempo $\Theta(n^3)$

Floyd-Warshall: Ejemplo

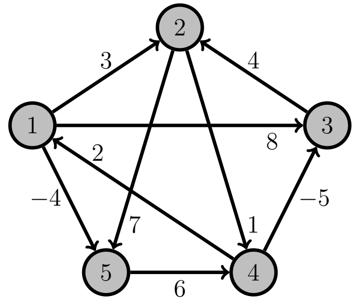


$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \mathbf{5} & -5 & 0 & \mathbf{-2} \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & \mathbf{4} & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \mathbf{5} & \mathbf{11} \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

Floyd-Warshall: Ejemplo



$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & \mathbf{-1} & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & \mathbf{-1} & 4 & -4 \\ \mathbf{3} & 0 & \mathbf{-4} & 1 & \mathbf{-1} \\ \mathbf{7} & 4 & 0 & 5 & \mathbf{3} \\ 2 & -1 & -5 & 0 & -2 \\ \mathbf{8} & \mathbf{5} & \mathbf{1} & 6 & 0 \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & \mathbf{1} & \mathbf{-3} & \mathbf{2} & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$