Caminhos mais curtos entre cada dois vértices de um grafo

All-pairs shortest paths

Problema

Como calcular os caminhos mais curtos entre cada dois vértices de um grafo pesado (orientado ou não)

Soluções

- Aplicar um dos algoritmos anteriores a partir de cada um dos vértices
- . . . ?

Caminhos mais curtos entre cada dois vértices de um grafo

Algoritmo de Floyd-Warshall

Os vértices intermédios de um caminho simples $v_1\ v_2\ \ldots\ v_l$ são os vértices $\{v_2, \ldots, v_{l-1}\}$

Seja $G = (V, E)$ um grafo pesado, com $V = \{1, 2, \ldots, n\}$

Seja p um caminho mais curto do vértice i para o vértice j, cujos vértices intermédios estão contidos em $\{1, 2, \ldots, k\}$

- Se k não é um nó intermédio de p, os nós intermédios de p estão contidos em $\{1, 2, \ldots, k-1\}$
- Se k é um nó intermédio de p, então p pode decompor-se num caminho p_1 de i para k e num caminho p_2 de k para j
  - Os nós intermédios de p_1 e de p_2 estão contidos em $\{1, 2, \ldots, k-1\}$ (porque p é um caminho simples)
  - p_1 e p_2 são caminhos mais curtos de i para k e de k para j, respectivamente

Caminhos mais curtos entre cada dois vértices de um grafo

Função recursiva

w_{ij}: matriz de adjacências do grafo

$$w_{ij} = \begin{cases} 0 & \text{se} \quad i = j \\ w(i,j) & \text{se} \quad i \neq j \wedge (i,j) \in E \\ \infty & \text{se} \quad i \neq j \wedge (i,j) \notin E \end{cases}$$

$d_{ij}^{(k)}$: peso de um caminho mais curto de i para j com nós intermédios contidos em $\{1, 2, \ldots, k\}$

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

Caminhos mais curtos entre cada dois vértices de um grafo

Cálculo iterativo de $d_{ij}^{(k)}$

FLOYD-WARSHALL-1(w)

```
1 n <- w.rows
2 d^(0) <- w
3 for k <- 1 to n do
4   let d^(k)[1..n,1..n] be a new matrix
5   for i <- 1 to n do
6     for j <- 1 to n do
7       d^(k)[i,j] <-
          min(d^(k-1)[i,j], d^(k-1)[i,k] + d^(k-1)[k,j])
8 return d^(n)
```

Complexidade temporal $\Theta(V^3)$

Complexidade espacial $\Theta(V^3)$

Caminhos mais curtos entre cada dois vértices de um grafo

Cálculo iterativo melhorado

FLOYD-WARSHALL(w)

```
1 n <- w.rows
2 d <- w
3 for k <- 1 to n do
4   for i <- 1 to n do
5     for j <- 1 to n do
6       if d[i,k] + d[k,j] < d[i,j] then
7         d[i,j] <- d[i,k] + d[k,j]
8 return d
```

Complexidade temporal $\Theta(V^3)$

Complexidade espacial $\Theta(V^2)$

Caminhos mais curtos entre cada dois vértices de um grafo

Predecessores

O predecessor de v_j no caminho $q = v_i \ldots v_j$

- Não existe, se $q = v_j$
- É v_i, se $q = v_i\ v_j$
- É o predecessor de v_j no caminho $v_k \ldots v_j$, se

$$q = v_i \ldots v_k \ldots v_j$$

π_{ij}: predecessor de v_j num caminho mais curto de v_i para v_j

$$\pi_{ij} = \begin{cases} \text{NIL} & \text{se } i = j \\ i & \text{se um caminho mais curto de } i \text{ para } j \text{ é } v_i\ v_j \\ \pi_{kj} & \text{se um caminho mais curto de } i \text{ para } j \text{ é } v_i \ldots v_k \ldots v_j \\ \text{NIL} & \text{se } d_{ij} = \infty \end{cases}$$

Caminhos mais curtos entre cada dois vértices de um grafo

Inclusão do cálculo dos predecessores

FLOYD-WARSHALL(w)

```
 1 n <- w.rows
 2 d <- w
 3 let p[1..n,1..n] be a new matrix        // p[i,j] ≡ π_ij
 4 for i <- 1 to n do
 5   for j <- 1 to n do
 6     if i = j or w[i,j] = ∞ then
 7       p[i,j] <- NIL
 8     else
 9       p[i,j] <- i
10 for k <- 1 to n do
11   for i <- 1 to n do
12     for j <- 1 to n do
13       if d[i,k] + d[k,j] < d[i,j] then
14         d[i,j] <- d[i,k] + d[k,j]
15         p[i,j] <- p[k,j]
16 return d and p
```

Caminho mais curto entre dois vértices

Reconstrução do caminho

PRINT-ALL-PAIRS-SHORTEST-PATH(p, i, j)

Exercício

Complexidade dos algoritmos

$G = (V, E)$

	Compl. Temporal
Percurso em largura	$O(V + E)$
Percurso em profundidade	$\Theta(V + E)$
Ordenação topológica (ambos os algoritmos)	$\Theta(V + E)$
Grafo transposto	$\Theta(V + E)$
Cálculo das componentes fortemente conexas	$\Theta(V + E)$
Algoritmos de Prim e de Kruskal	$O(E \log V)$
Caminhos mais curtos num DAG	$\Theta(V + E)$
Algoritmo de Dijkstra	$O(E \log V)$
Algoritmo de Bellman-Ford	$O(VE)$
Algoritmo de Floyd-Warshall	$\Theta(V^3)$

Pressupostos

Grafo representado através de listas de adjacências (excepto algoritmos de Kruskal, de Bellman-Ford e de Floyd-Warshall)

Algoritmos de Prim e de Dijkstra recorrem a uma fila tipo *heap* binário com actualização (EXTRACT-MIN e DECREASE-KEY com complexidade temporal logarítmica no número de elementos da fila)

Algoritmo de Kruskal usa Partição com compressão de caminho