Caminhos mínimos de todos os pares

Marco A L Barbosa
malbarbo.pro.br

Departamento de Informática
Universidade Estadual de Maringá

Dado um grafo orientado $G = (V, E)$ e uma função peso $w : E \to R$, queremos encontrar, para todo par de vértices $u, v \in V$, um caminho de peso mínimo de u até v.

Como resolver esse problema?

Podemos resolver este problema aplicando um algoritmo de caminho mínimo de única origem $|V|$ vezes, uma vez para cada vértice.

Dijkstra (sem arestas com pesos negativos)

- Arranjo linear: $O(V^3 + VE) = O(V^3)$
- Heap binário: $O(VE \lg V)$, se o grafo é denso $O(V^3 \lg V)$
- Heap de Fibonacci: $O(V^2 \lg V + VE)$, se o grafo é denso $O(V^3)$

Bellman-Ford (grafos gerais)

- $O(V^2E)$, se o grafo é denso $O(V^4)$

Podemos fazer melhor para o caso geral (grafos com arestas de peso negativo)?

Por conveniência vamos assumir que

- O grafo é representado por matriz de adjacências
- Os vértices são numerados como $1, 2, 3, \ldots, n$, onde $n = |V|$
- O grafo não tem ciclos de peso negativos

A entrada é uma matriz $n \times n$ W que representa os pesos das arestas. Isto é, $W = (w_{ij})$, onde

$$w_{ij} = \begin{cases} 0 & \text{se } i = j \\ \text{o peso da aresta } (i,j) & \text{se } i \neq j \text{ e } (i,j) \in E \\ \infty & \text{se } i \neq j \text{ e } (i,j) \notin E \end{cases}$$

E a saída?

- Uma matriz $n \times n$ $D = (d_{ij})$, onde a entrada d_{ij} contém o peso do caminho mínimo do vértice i até o vértice j, ou seja, $d_{ij} = \delta(i,j)$;
- Uma matriz $n \times n$ $\Pi = (\pi_{ij})$, onde π_{ij} é o vértice predecessor de j em um caminho mínimo a partir de i.

Vamos tentar usar o modelo de programação dinâmica que desenvolvemos para o problema de caminhos mínimos de única origem.

A ideia é descrever caminhos mínimos com até k arestas em termos de caminhos mínimos com até $k - 1$ arestas.

Caminhos mínimos de única origem

$\delta^k(s, v)$ – peso do caminho mínimo da origem s para v que utiliza até k arestas

$$\delta^k(s,v) = \begin{cases} 0 & \text{se } s = v \text{ e } k = 0 \\ \infty & \text{se } s \neq v \text{ e } k = 0 \\ \min \begin{cases} \delta^{k-1}(s,v) \\ \min\limits_{(u,v)\in E} (\delta^{k-1}(s,u) + w(u,v)) \end{cases} & \text{caso contrário} \end{cases}$$

Caminhos mínimos de todos os pares

Como a entrada e saída são matrizes, vamos mudar um pouco a notação.

δ_{ij}^k – peso do caminho mínimo de i para j que utiliza até k arestas

$$\delta_{ij}^k = \begin{cases} 0 & \text{se } i = j \text{ e } k = 0 \\ \infty & \text{se } i \neq j \text{ e } k = 0 \\ \min \begin{cases} \delta_{ij}^{k-1} \\ \min\limits_{1 \leq u \leq n} (\delta_{iu}^{k-1} + w_{uj}) \end{cases} & \text{caso contrário} \end{cases}$$

$$\delta_{ij}^k = \begin{cases} w_{ij} & \text{se } k = 1 \\ \min\limits_{1 \leq u \leq n} (\delta_{iu}^{k-1} + w_{uj}) & \text{se } k > 1 \end{cases}$$

Podemos simplificar a equação? Sim!

- $\delta_{ij}^1 = w_{ij}$; e
- $w_{ii} = 0$ para todo i.

$$\delta_{ij}^{k} = \begin{cases} w_{ij} & \text{se } k = 1 \\ \min\limits_{1 \le u \le n} \left(\delta_{iu}^{k-1} + w_{uj}\right) & \text{se } k > 1 \end{cases}$$

Qual o tempo de execução do algoritmo que implementa essa equação diretamente? Vamos escrever o algoritmo e verificar!

SLOW-ALL-PAIRS-SHORTEST-PATHS(W)

```
1  n = W.linhas
2  D^1 = (d_ij^1 = w_ij) uma matriz n × n.
3  for k = 2 to n − 1
4      Seja D^k = (d_ij^k) uma matriz n × n
5      for i = 1 to n
6          for j = 1 to n
7              d_ij^k = ∞
8              for u = 1 to n
9                  d_ij^k = min(d_ij^k, d_iu^{k−1} + w_uj)
10 return D^{n−1}
```

Qual o tempo de execução? $\Theta(V^4)$, o mesmo que executar BELLMAN-FORD para cada vértice!

A forma do código da iteração que calcula D^k a partir de D^{k-1} é semelhante a forma da multiplicação das matrizes $D^{k-1} \cdot W$. Se usarmos a notação $D^{k-1} \cdot W$ para representar o cálculo de D^k a partir de D^{k-1} e W, podemos escrever:

$$
\begin{aligned}
D^1 &= W \\
D^2 &= D^1 \cdot W = W^2 \\
D^3 &= D^2 \cdot W = W^3 \\
&\dots \\
D^{n-1} &= D^{n-2} \cdot W = W^{n-1}
\end{aligned}
$$

Qual o resultado obtemos se fizermos $D^{n-1} \cdot W$? Se o grafo não tem ciclos de peso negativo, obtemos a própria matriz D^{n-1}.

Será que podemos "acelerar" o cálculo de D^{n-1}? Sim!

Repetição do quadrado

$$D^1 \cdot D^1 = D^2$$
$$D^2 \cdot D^2 = D^4$$
$$D^4 \cdot D^4 = D^8$$
$$\dots$$

Quantas vezes precisamos repetir o quadrado para chegar em D^m, onde $m \geq n - 1$? $\Theta(\lg V)$ vezes.

Então, podemos calcular D^{n-1} a partir de D^1 no tempo $\Theta(V^3 \lg V)$.

Uma boa melhora! Mas será que podemos fazer melhor? Vamos tentar uma outra modelagem de programação dinâmica.

Vértice intermediário

Para um caminho $p = \langle v_1, v_2, \ldots, v_l \rangle$, um **vértice intermediário** é qualquer vértice de p que não seja v_1 ou v_l.

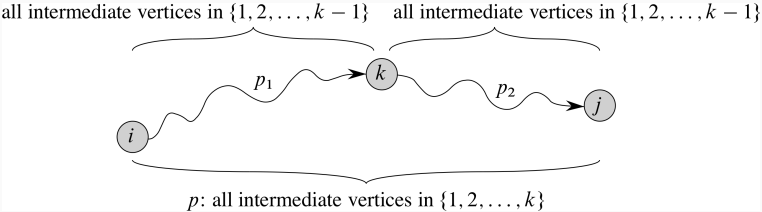
Vamos usar o conceito de vértice intermediário para formular a estrutura ótima de caminhos mínimos.

Considere um caminho mínimo $i \overset{p}{\rightsquigarrow} j$ com todos os vértices intermediários em $\{1, 2, \ldots, k\}$, o vértice k é um vértice intermediário de p?

Ele pode ou não ser, então vamos analisar as duas possibilidades.

Se k não é um vértice intermediário de p, então, todos os vértices intermediários de p estão em $\{1, 2, \ldots, k-1\}$. Deste modo, um caminho mínimo $i \rightsquigarrow j$ com todo os vértices intermediários no conjunto $\{1, 2, \ldots, k-1\}$, também é um caminho mínimo $i \rightsquigarrow j$ com todos os vértices intermediários no conjunto $\{1, 2, \ldots, k\}$.

Se k é um vértice intermediário do caminho p, então desmembramos o caminho p em $i \overset{p_1}{\rightsquigarrow} k \overset{p_2}{\rightsquigarrow} j$. p_1 é um caminho mínimo de i até k, com todos os vértices intermediários no conjunto $\{1, 2, \ldots, k-1\}$. A mesma ideia se aplica a p_2



Seja $D^{(k)} = \left(d_{ij}^{(k)}\right)$ uma matriz, onde $d_{ij}^{(k)}$ o peso de um caminho mínimo $i \rightsquigarrow j$ com todos os vértices intermediários em $\{1, 2, \ldots, k\}$

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

Observe que a matriz $D^{(n)}$ fornece a reposta desejada: $d_{ij}^{(n)} = \delta(i, j)$ para todo $i, j \in V$, isto porque para qualquer caminho mínimo todos os vértices intermediários estão no conjunto $\{1, 2, \ldots, n\}$.

FLOYD-WARSHALL(W)

1 $n = W.linhas$
2 $D^{(0)} = W$
3 **for** $k = 1$ **to** n
4 seja $D^{(k)} = \left(d_{ij}^{(k)}\right)$ uma matriz $n \times n$
5 **for** $i = 1$ **to** n
6 **for** $j = 1$ **to** n
7 $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
8 **return** $D^{(n)}$

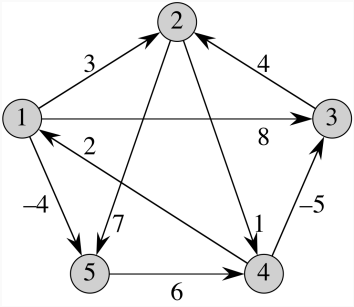
Análise do tempo de execução

- Cada execução da linha 7 demora $O(1)$
- A linha 7 é executada n^3 vezes
- Portanto, o tempo de execução do algoritmo é $\Theta(n^3) = \Theta(V^3)$

Exemplo de execução

FLOYD-WARSHALL(W)

1 $n = W.linhas$
2 $D^{(0)} = W$
3 **for** $k = 1$ **to** n
4 seja $D^{(k)} = \left(d_{ij}^{(k)}\right)$ uma matriz $n \times n$
5 **for** $i = 1$ **to** n
6 **for** $j = 1$ **to** n
7 $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
8 **return** $D^{(n)}$



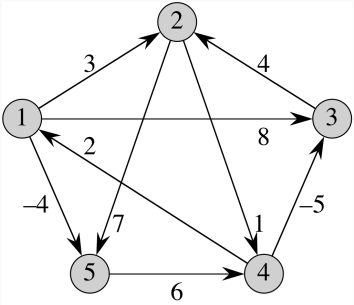
$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$\Pi^{(0)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

Exemplo de execução

FLOYD-WARSHALL(W)

1 $n = W.linhas$
2 $D^{(0)} = W$
3 **for** $k = 1$ **to** n
4 seja $D^{(k)} = \left(d_{ij}^{(k)}\right)$ uma matriz $n \times n$
5 **for** $i = 1$ **to** n
6 **for** $j = 1$ **to** n
7 $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
8 **return** $D^{(n)}$



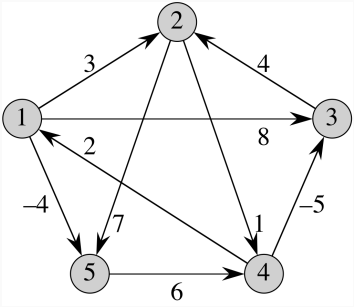
$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$\Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

Exemplo de execução

FLOYD-WARSHALL(W)

1 $n = W.linhas$
2 $D^{(0)} = W$
3 **for** $k = 1$ **to** n
4 seja $D^{(k)} = \left(d_{ij}^{(k)}\right)$ uma matriz $n \times n$
5 **for** $i = 1$ **to** n
6 **for** $j = 1$ **to** n
7 $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
8 **return** $D^{(n)}$



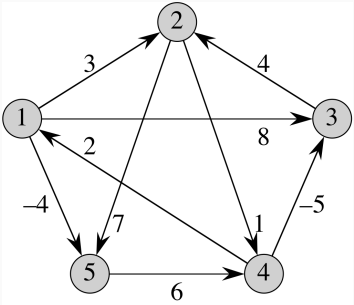
$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$\Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

Exemplo de execução

FLOYD-WARSHALL(W)

1 $n = W.linhas$
2 $D^{(0)} = W$
3 **for** $k = 1$ **to** n
4 seja $D^{(k)} = \left(d_{ij}^{(k)}\right)$ uma matriz $n \times n$
5 **for** $i = 1$ **to** n
6 **for** $j = 1$ **to** n
7 $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
8 **return** $D^{(n)}$



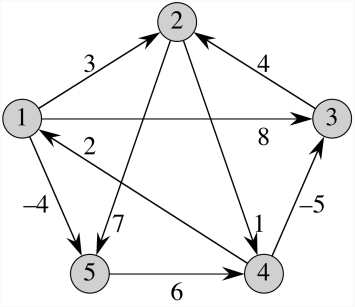
$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$\Pi^{(3)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

Exemplo de execução

FLOYD-WARSHALL(W)

1 $n = W.linhas$
2 $D^{(0)} = W$
3 **for** $k = 1$ **to** n
4 seja $D^{(k)} = \left(d_{ij}^{(k)}\right)$ uma matriz $n \times n$
5 **for** $i = 1$ **to** n
6 **for** $j = 1$ **to** n
7 $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
8 **return** $D^{(n)}$



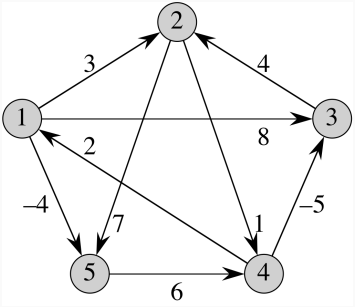
$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

$$\Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

Exemplo de execução

FLOYD-WARSHALL(W)

1 $n = W.linhas$
2 $D^{(0)} = W$
3 **for** $k = 1$ **to** n
4 seja $D^{(k)} = \left(d_{ij}^{(k)}\right)$ uma matriz $n \times n$
5 **for** $i = 1$ **to** n
6 **for** $j = 1$ **to** n
7 $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
8 **return** $D^{(n)}$



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

$$\Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

Como construir um caminho mínimo?

- Calcular a matriz predecessora Π, durante o calculo da matriz de distância de caminhos mínimos D
- Quando $k = 0$, um caminho mínimo de i até j não tem nenhum vértice intermediário, então

$$\pi_{ij}^{(0)} = \begin{cases} \text{NIL} & \text{se } i = j \text{ ou } w_{ij} = \infty \\ i & \text{se } i \neq j \text{ e } w_{ij} < \infty \end{cases}$$

- Quando $k \geq 1$

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{se } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{kj}^{(k-1)} & \text{se } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases}$$

Considerando o exercício 25.2-4 e acrescentando o calculo de Π, podemos reescrever o procedimento FLOYD-WARSHALL

O algoritmo de Floyd-Warshall

FLOYD-WARSHALL(W)

```
1  n = W.linhas
2  D = W
3  Π = (π_ij) uma matriz n × n
4  for i = 1 to n
5      for j = 1 to n
6          if i = j ou w_ij = ∞
7              π_ij = NIL
8          else
9              π_ij = i
10 for k = 1 to n
11     for i = 1 to n
12         for j = 1 to n
13             if d_ij > d_ik + d_kj
14                 d_ij = d_ik + d_kj
15                 π_ij = π_kj
16 return D, Π
```

Referências

Thomas H. Cormen et al. Introduction to Algorithms. 3rd edition. Capítulo 25.