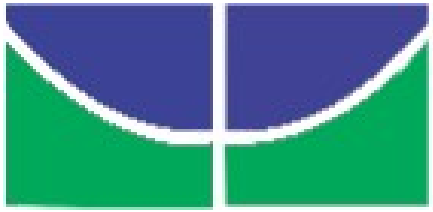


Caminho mínimo parte II

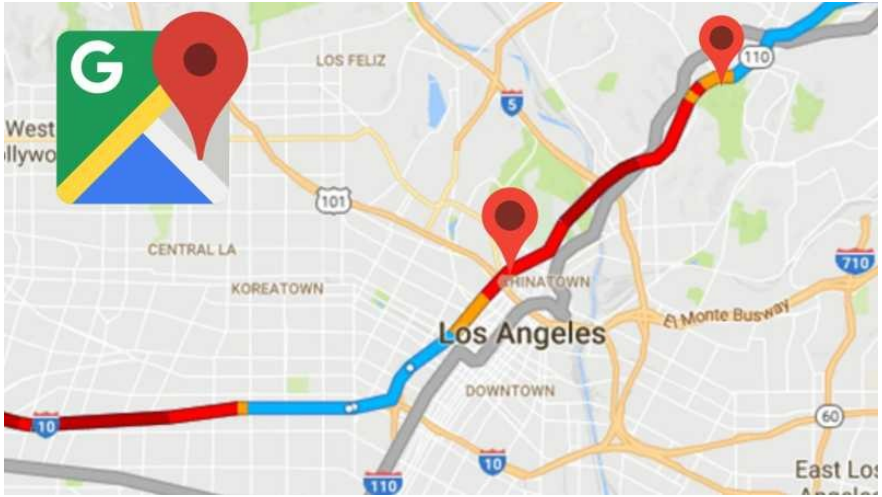
prof. Dr. Wesin Ribeiro

Neste capítulo

- Introdução
- Caminhos mínimos e multiplicação de matrizes
- O Algoritmo de Floyd-Warshal
- Algoritmo de Jonhson
- Revisão

Introdução

No capítulo anterior descobrimos como encontrar o caminho mínimo de fonte única em um grafo, neste capítulo veremos como resolver o problema de encontrar o caminho mínimo entre todos os pares. Uma alternativa seria utilizar os algoritmos do capítulo anterior considerando cada vértice como fonte de cada vez, mas não seria muito eficiente. Esse capítulo mostra dois algoritmos mais adequados para contornar essa ineficiência.



Introdução

Pode ser resolvido usando um algoritmo para o problema de caminhos mínimos de única origem V vezes, onde V representa a quantidade de vértices do grafo.

A solução do problema de caminhos mínimos entre todos os pares irá precisar de duas estruturas: matriz de adjacências e matriz de predecessores.

$$V_{\pi,\,i} = \{j \in V : \pi_{ij} \neq \text{NIL}\} \cup \{i\}$$

e

$$E_{\pi,\,i} = \{(\pi_{ij}, j)) : j \in V_{\pi,\,i} - \{i\}\} .$$

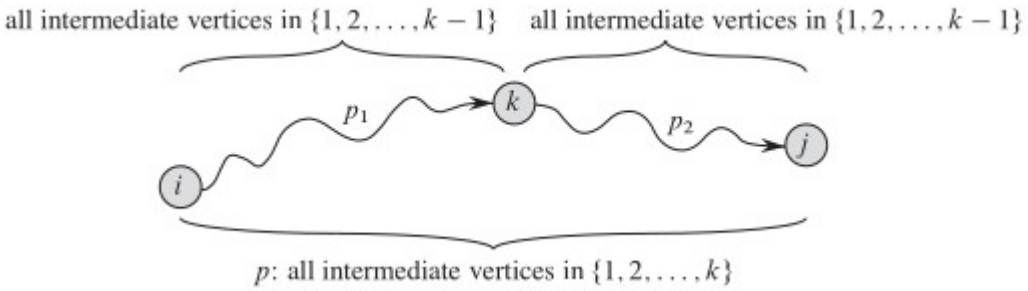
- ✓ Menor caminho entre cidades.
- ✓ Quantidade de conexões em um nó central

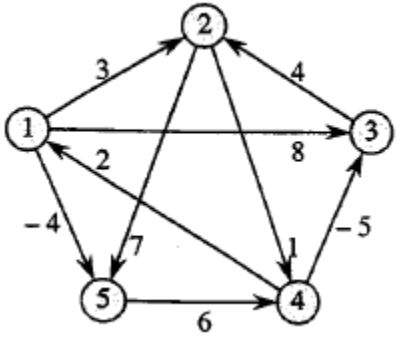
$$w_{ij} = \begin{cases} 0 & \text{se } i = j , \\ \text{o peso da aresta orientada } (i, j) & \text{se } i \neq j \text{ e } (i, j) \in E , \\ \infty & \text{se } i \neq j \text{ e } (i, j) \notin E . \end{cases}$$

O Algoritmo de Floyd Marshal

Esse algoritmo considera vértices intermediários de um caminho mais curto, como sendo qualquer vértice p de um caminho mais curto diferente da origem e do destino.

- ✓ Permite arestas de peso negativo
- ✓ Não admite ciclos de peso negativo





Pseudo código

```
FLOYD-WARSHALL(W)
1  n = W.rows
2  D^(0) = W
3  for k = 1 to n
4      let D^(k) = (d_ij^(k)) be a new n × n matrix
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)
```

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

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

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

O Algoritmo de Floyd Marshal

A entrada do algoritmo é uma matriz Wnxn representando os pesos de cada aresta do grafo conforme definida anteriormente.

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

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

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

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

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

O Algoritmo de Floyd Marshal

Se k não é um vértice intermediário do caminho mais curto, então o caminho mais curto é o caminho sem passar pelo vértice k.

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

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

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

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

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

O Algoritmo de Floyd Marshal

Se k é um vértice intermediário, então o caminho mais curto é o caminho que passa pelo vértice k.

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

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

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

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

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

O algoritmo de Jonhson

Usado para calcular o caminho crítico entre todos os pares com desempenho melhor quando o grafo é esparso. Retorna uma matriz de peso ou informa se o grafo contém um ciclo de peso negativo.

- Usa os algoritmos dijkstra e bellman-ford como subrotinas
- Emprega a técnica de reponderar
- Arestas de pesos negativos w para arestas de peso não negativo w’
- P é um caminho mais curto se for usando ambos os casos w ou w’
- o novo peso w’ é não negativo.

O algoritmo de Jonhson

Usado para calcular o caminho crítico entre todos os pares com desempenho melhor quando o grafo é esparso. Retorna uma matriz de peso ou informa se o grafo contém um ciclo de peso negativo.

- Usa os algoritmos dijkstra e bellman-ford como subrotinas
- Emprega a técnica de reponderar
- Arestas de pesos negativos w para arestas de peso não negativo w’
- P é um caminho mais curto se for usando ambos os casos w ou w’
- o novo peso w’ é não negativo.

$$\hat{w}(c) = w(c) + b(v_0) - b(v_k)$$

Pseudo código

```
JOHNSON(G, w)
 1  compute G', where G'.V = G.V ∪ {s},
        G'.E = G.E ∪ {(s, v) : v ∈ G.V}, and
        w(s, v) = 0 for all v ∈ G.V
 2  if BELLMAN-FORD(G', w, s) == FALSE
 3      print "the input graph contains a negative-weight cycle"
 4  else for each vertex v ∈ G'.V
 5          set h(v) to the value of δ(s, v)
                computed by the Bellman-Ford algorithm
 6      for each edge (u, v) ∈ G'.E
 7          ŵ(u, v) = w(u, v) + h(u) − h(v)
 8      let D = (d_uv) be a new n × n matrix
 9      for each vertex u ∈ G.V
10          run DIJKSTRA(G, ŵ, u) to compute δ̂(u, v) for all v ∈ G.V
11          for each vertex v ∈ G.V
12              d_uv = δ̂(u, v) + h(v) − h(u)
13      return D
```

O algoritmo de Jonhson

a) Função peso original. b) função peso ponderado. c-g) aplicação do algoritmo de dijkstra sobre os vértices usando a função de peso ponderado.