Caminhos Mínimos entre Todos os Vértices

Eduardo Camponogara

Departamento de Automação e Sistemas
Universidade Federal de Santa Catarina

DAS-9003: Introdução a Algoritmos

Sumário

Todos os caminhos mínimos

Contexto

- Consideramos o problema de encontrar o caminho mais curto entre cada par de vértices de um grafo $G = (V, E)$.
- Este problema pode surgir na construção de uma tabela de distâncias entre cidades de um atlas.
- Nos é dado um grafo direcionado $G = (V, E)$ com função peso das arestas dada por $w : E \to \mathbb{R}$.
- **Problema:** para cada par $u, v \in V$, encontre o caminho mais curto (de menor peso) de u para v.

Todos os caminhos mínimos

Métodos de solução

Podemos resolver o *"problema de caminhos mínimos entre todos os pares de vértices"* resolvendo $|V| = n$ *"problemas de caminhos mínimos com uma fonte."*

Todos os caminhos mínimos

Métodos de solução

Se todos os pesos são não negativos, podemos utilizar o algoritmo de Dijkstra.

1. Com fila de prioridade implementada com vetor: custo computacional $O(n(n^2 + m)) = O(n^3 + nm) = O(n^3)$, onde $|E| = m$.
2. Com fila de prioridade implementada com heap binário: custo computacional $(nm\lg n)$ que é uma melhoria para grafos esparsos.
3. Com fila de prioridade implementada com heap Fibonacci: custo computacional: $O(n(n\lg n + m)) = O(n^2\lg n + nm)$.

Todos os caminhos mínimos

Métodos de solução

- Se há arestas com pesos negativos, então não podemos utilizar o algoritmo de Dijkstra.
- Utilizamos o algoritmo de Bellman-Ford, mais lento, que deve ser executado a partir de cada vértice.
- O resultado é um custo computacional de $O(n \times nm) = O(n \times nn^2) = O(n^4)$.
- **Questão:** podemos resolver o problema de forma mais rápida?

Todos os caminhos mínimos

Propósito

- Mostraremos que é possível resolver o *"problema de todos os caminhos mínimos"* mais rapidamente do que $O(n^4)$.
- Podemos ainda estabelecer uma relação entre este problema e o problema multiplicação de matrizes.
- Diferente dos algoritmos anteriores, que utilizaram lista de adjacência, aqui representaremos o grafo por meio de uma matriz de adjacência.
- Por conveniência, também assumiremos que os vértices são numerados $1, 2, \ldots, |V| = n$.

Todos os caminhos mínimos

- ciclos com peso negativo são permitidos, mas assumiremos por enquanto que o grafo não contém ciclo negativo.
- A saída tabulada do algoritmo é uma matriz $D = [d_{ij}]$.
- Cada entrada d_{ij} contém o comprimento do caminho mais curto do vértice i ao vértice j.
- Ou seja, $d_{ij} = \delta(i, j)$.

Todos os caminhos mínimos

- Computamos não apenas a distância, mas também o caminho. Para tanto, utilizamos uma matriz $\Pi = [\pi_{ij}]$ com os predecessores.
- A entrada $\pi_{ij} = nil$ se $i = j$ e π_{ij} é o predecessor de j em um caminho mais curto de i para j.
- Da mesma forma que nos problemas anteriores, o subgrafo predecessor G_π é uma árvore de caminhos mínimos a partir de um vértice.
- O subgrafo predecessor $G_{\pi,i}$ induzido pela i-ésima linha é uma árvore de caminhos mínimos em G a partir de i.

Todos os caminhos mínimos

- O grafo predecessor $G_{\pi,i}$ é definido por:

$$V_{\pi,i} = \{j \in V : \pi_{i,j} \neq nil\} \cup \{i\}$$
$$E_{\pi,i} = \{(\pi_{ij}, j) : j \in V_{\pi,i} - \{i\}\}$$

Todos os caminhos mínimos

Print-all-pairs-shortest-path(Π, i, j)
1) if $i = j$
2) then print i
3) else if $\pi_{ij} = nil$
4) then print "no path from" i "to" j exists
5) else Print-all-pairs-shortest-path(Π, i, π_{ij})
6) print j

Sumário

- Apresenta-se um algoritmo de programação dinâmica para o cômputo de todos os caminhos mínimos em um grafo direcionado $G = (V, E)$.
- Iniciamos com um algoritmo de tempo $\Theta(n^4)$ que depois é reduzido para $\Theta(n^3 \lg n)$.

Estrutura de Caminhos Mínimos

- Suponha que o grafo é representado por uma matriz de adjacência $W = (w_{i,j})$.
- Seja p o caminho mais curto de i para j e suponha que p contém m arestas. Assumindo a não existência de ciclos negativos, m é finito.
- Se $i = j$, então p tem comprimento 0 e nenhuma aresta.
- Se $i \neq j$, então $i \rightsquigarrow k \rightarrow j$, sendo que o caminho $p' : i \rightsquigarrow k$ tem no máximo $m - 1$ arestas.
- Além disso p' é um caminho mais curto de i para k.

Solução Recursiva

- Seja d_{ij}^m o comprimento mínimo de qualquer caminho de i para j que contenha no máximo m arestas.
- Quando $m = 0$, o caminho não possui arestas o que ocorre somente quando $i = j$. Logo,

$$d_{ij}^0 = \begin{cases} 0 & \text{if } i = j \\ \infty & \text{if } i \neq j \end{cases}$$

- Para $m \geq 1$, computa-se d_{ij}^m como o mínimo de d_{ij}^{m-1} e o menor comprimento de qualquer caminho de i para j consistindo de até m arestas, obtido ao examinar-se todos os possíveis predecessores k de j.

- A solução recursiva é obtida conforme segue,

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

A última igualdade segue do fato que $w_{ii} = 0$ para todo i.

Qual é o comprimento $\delta(i,j)$ do caminho mais curto?

- Se o grafo não possui ciclos negativos, então todos os caminhos são simples e possuem no máximo $n-1$ arestas.
- Um caminho de i para j com mais do que $n-1$ arestas não pode ser mais leve que um caminho mais curto.
- O comprimento do caminho mais curto é portanto dado por:

$$\delta(i,j) = d_{ij}^{n-1} = d_{ij}^{n} = d_{ij}^{n+1} = \cdots$$

Computando o Caminho Mais Curto de Baixo para Cima

- Tomando a matriz $W = (w_{ij})$, computa-se uma série de matrizes $D^1, D^2, \ldots, D^{n-1}$, sendo $D^m = (d_{ij}^m)$.
- A matriz D^{n-1} contém o comprimento dos caminhos mais curtos.
- Note que $d_{ij}^1 = w_{ij}$ para todo $i, j \in V$, o que implica $D^1 = W$.
- O algoritmo a seguir computa, a partir das matrizes D^{m-1} e W, a matriz D^m.

Extend-Shortest-Paths(D, W)

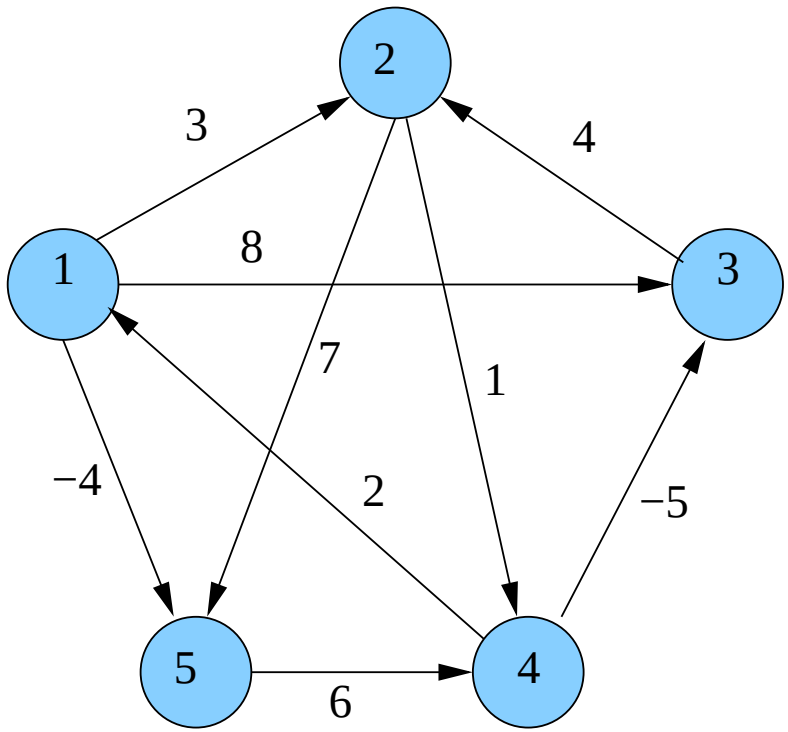
1) $n \leftarrow rows(D)$
2) $D' = (d'_{ij})$ is a matrix of dimension $n \times n$
3) for $i \leftarrow 1$ to n
4) do for $j \leftarrow 1$ to n
5) do $d'_{ij} \leftarrow \infty$
6) for $k \leftarrow 1$ to n
7) do $d'_{ij} \leftarrow \min\{d'_{ij}, d_{ik} + w_{kj}\}$
8) return D'

O algoritmo Extended-Shortest-Paths roda em tempo $\Theta(n^3)$.

Slow-All-Pairs-Shortest-Paths(W)

1) $n \leftarrow rows(W)$
2) $D^1 \leftarrow W$
3) for $m \leftarrow 2$ to $n-1$
4) do $D^m \leftarrow$ Extend-Shortest-Paths(D^{m-1}, W)
5) return D^{n-1}

O algoritmo Slow-All-Pairs-Shortest-Paths roda em tempo $\Theta(n^4)$.


2
3
4
1
8
3
7
1
−4
2
−5
5
4
6

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

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

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

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

Caminhos Mínimos como Multiplicação de Matrizes

- Podemos ver a relação entre caminhos mínimos e multiplicação.
- Considere o cômputo do produto $C = A \cdot B$ de duas matrizes A e B de dimensão $n \times n$. Então $c_i j$ é calculado como segue:

$$c_{ij} = \sum_{k=1}^{n} \cdot a_{ik} b_{kj}$$

Observe que se fizermos as modificações na equação de caminhos mínimos:

- $d^{m-1} \rightarrow a$, $w \rightarrow b$
- $d^{m} \rightarrow c$, $\min \rightarrow +$
- $+ \rightarrow \cdot$

Então verifica-se que

$$d_{ij}^{m} = \min_{1 \leq k \leq n} \{ d_{ik}^{m-1} + w_{kj} \} \implies c_{ij} = \sum_{k=1}^{n} \cdot a_{ik} b_{kj}$$

Fazendo as modificações acima em Extend-Shortest-Paths e também substituindo ∞ (identidade de min) por 0 (identidade de $+$), obtemos um procedimento $\Theta(n^3)$ para multiplicação de matrizes.

Matrix-Multiply(A, B)

1) $n \leftarrow rows(A)$
2) C is an $n \times n$ matrix
3) for $i \leftarrow 1$ to n
4) do for $j \leftarrow 1 \leftarrow 1$ to n
5) do $c_{ij} \leftarrow 0$
6) for $k \leftarrow 1$ to n
7) do $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$
8) return C

Acelerando o Algoritmo

Seja $A\cdot$ o "produto" gerado por Extend-Shorthest-Paths(A, B), calcula-se a sequência de $n-1$ matrizes conforme segue:

$$
\begin{aligned}
D^1 &= D^0 \cdot W = W \\
D^2 &= D^1 \cdot W = W^2 \\
D^3 &= D^2 \cdot W = W^3 \\
\vdots &= \quad \vdots \\
D^{n-1} &= D^{n-2} \cdot W = W^{n-1}
\end{aligned}
$$

Uma vez que queremos computar D^{n-1}, não precisamos computar todas as matrizes D^m. Pode-se computar D^{n-1} com apenas $\lceil \lg(n-1) \rceil$ produtos de matrizes, conforme a sequência:

$$
\begin{aligned}
D^1 &= W \\
D^2 &= W^2 = W \cdot W \\
D^4 &= W^4 = W^2 \cdot W^2 \\
D^8 &= W^8 = W^4 \cdot W^4 \\
\vdots &= \quad \vdots \\
D^{2^{\lceil \lg(n-1) \rceil}} &= W^{2^{\lceil \lg(n-1) \rceil}} = W^{2^{(\lceil \lg(n-1) \rceil - 1)}} \cdot W^{2^{(\lceil \lg(n-1) \rceil - 1)}}
\end{aligned}
$$

Faster-All-Pairs-Shortest-Paths(W)

1) $n \leftarrow rows(W)$
2) $D^1 \leftarrow W$
3) $m \leftarrow 1$
4) while $n - 1 > m$
5) do $D^{2m} \leftarrow$ Extend-Shortest-Paths(D^m, D^m)
6) $m \leftarrow 2m$
7) return D^m

O algoritmo Faster-All-Pairs-Shortest-Paths executa em tempo $\Theta(n^3 \log n)$, uma vez que realiza $\lceil \lg(n-1) \rceil$ produtos de matrizes, cada um deles levando tempo $\Theta(n^3)$.

Sumário

Algoritmo de Floyd-Warshall

Fundamentos

- Usamos uma formulação em programação dinâmica para resolver o *"problema de caminhos mínimos entre todos os pares de vértices."* sobre um grafo direcionado $G = (V, E)$.
- O algoritmo resultante, conhecido por algoritmo de *Floyd-Warshall*, roda em tempo $O(n^3)$.
- Arestas com peso negativo podem estar presentes, mas assumimos por conveniência que não há ciclos com pesos negativos. (O algoritmo é capaz de detectar tais ocorrências.)

Algoritmo de Floyd-Warhsall

Estrutura

- Consideramos um *vértice intermediário* de um *"caminho simples"* $p = (v_1, v_2, \ldots, v_k)$ qualquer vértice v_i diferente de v_1 e v_k, ou seja, v_i com $1 < i < k$.
- Para qualquer par de vértices $i, j \in V$, considere todos os caminhos de i para j cujos vértices intermediários são elementos do conjunto de índices $\{1, 2, \ldots, k\}$, denotado por S_{ij}^k.
- Seja p_{ij}^k o caminho mais curto dentre os caminhos em S_{ij}^k.
- O algoritmo de Floyd-Warshall explora a relação entre o caminho p_{ij}^k e o caminho mais curto p_{ij}^{k-1} dentre os caminhos em S_{ij}^{k-1}.

Algoritmo de Floyd-Warshall

Estrutura

- O relacionamento depende se k é um vértice intermediário em p_{ij}^k ou não.

Algoritmo de Floyd-Warshall

Caso a) $k \notin p_{ij}^k$

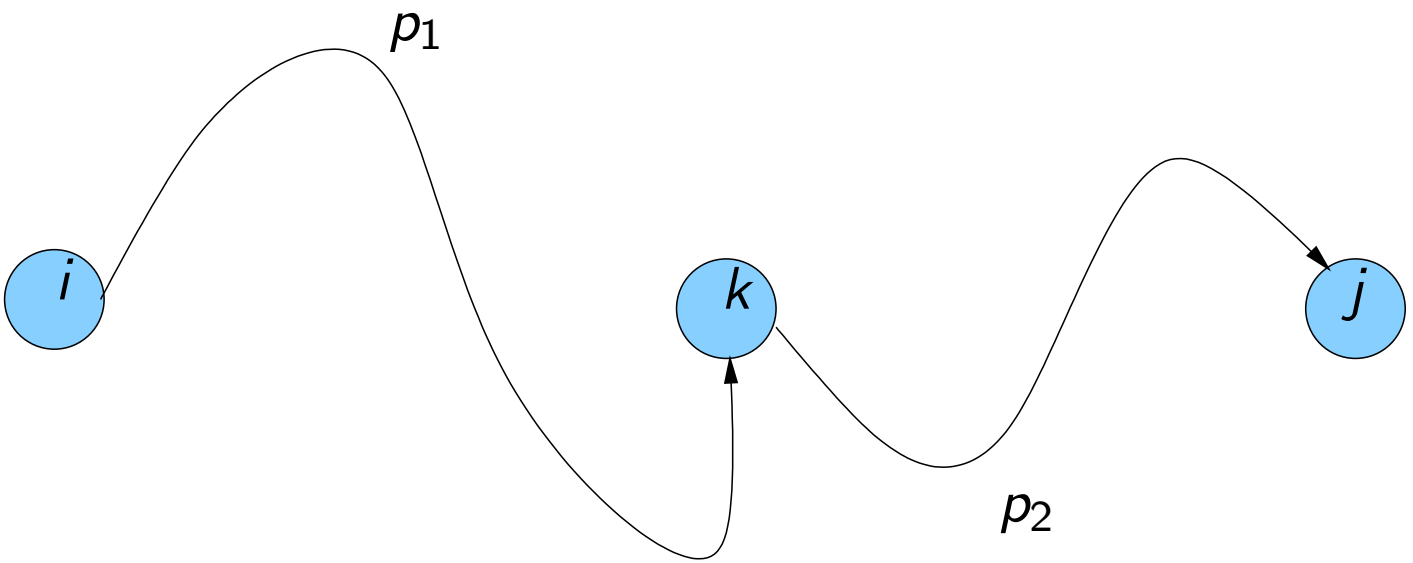
- k não é um vértice intermediário do caminho p_{ij}^k.
- Então todos os vértices intermediários em p_{ij}^k são elementos do conjunto $\{1, \ldots, k-1\}$.
- Logo um caminho mais curto $p_{ij}^k \in S_{ij}^k$ é também um elemento de S_{ij}^{k-1}.
- Ou seja, $p_{ij}^k \in S_{ij}^{k-1} \subset S_{ij}^k \Rightarrow p_{ij}^k = p_{ij}^{k-1}$.

Algoritmo de Floyd-Warshall

Caso b) $k \in p_{ij}^k$

- Se k é um vértice intermediário de p_{ij}^k, então podemos quebrar p_{ij}^k em $i \overset{p_1}{\rightsquigarrow} k \overset{p_2}{\rightsquigarrow} j$ como mostra a figura.
- Todos os vértices intermediários de p_1 são vértices do conjunto $\{1, \ldots, k-1\}$, ou seja, $p_1 = p_{ik}^{k-1} \in S_{ik}^{k-1}$.
- Todos os vértices intermediários de p_2 são vértices do conjunto $\{1, \ldots, k-1\}$, ou seja, $p_2 = p_{kj}^{k-1} \in S_{kj}^{k-1}$.

Algoritmo de Floyd-Warshall



Algoritmo de Floyd-Warshall

Caso b) $k \in p_{ij}^k$

- Sabemos que p_1 é um caminho simples, tal que $p_1 = p_{ik}^{k-1} \in S_{ik}^{k-1}$, pois k não é vértice intermediário de p_1.
- Sabemos que p_2 é um caminho simples, tal que $p_2 = p_{kj}^{k-1} \in S_{kj}^{k-1}$, pois k não é vértice intermediário de p_2.
- Portanto $p_{ij}^k = p_{ik}^{k-1} \cup p_{kj}^{k-1}$.

Solução recursiva

Solução recursiva

- Seja d_{ij}^k o comprimento do caminho mais curto de i para j, para o qual os vértices intermediários são elementos do conjunto $\{1, \ldots, k\}$.
- Quando $k = 0$, um caminho p_{ij}^0 não contém vértices intermediários, logo $d_{ij}^0 = w_{ij}$.
- Recursão:

$$d_{ij}^k = \begin{cases} w_{ij} & \text{if } k = 0 \\ \min\{d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}\} & \text{if } k \geqslant 1 \end{cases}$$

Solução recursiva

- Uma vez que para qualquer caminho, todos os vértices intermediários devem pertencer ao conjunto $\{1, \ldots, n\}$, concluímos que a matriz $D^n = [d_{ij}^n]$ nos dá a resposta final:

$$d_{ij}^n = \delta(i, j) \text{ para todo par } i, j \in V$$

Algoritmo

Floyd-Warshall(W)

1) $n \leftarrow rows(W)$
2) $D^0 \leftarrow W$
3) for $k \leftarrow 1$ to n
4) do for $i \leftarrow 1$ to n
5) do for $j \leftarrow 1$ to n
6) do $d_{ij}^k \leftarrow \min\{d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}\}$
7) return D^n

Construindo o caminho

- Há uma variedade de métodos para construção do caminho mais curto no algoritmo de Floyd-Warshall.
- Um jeito é computar a matriz D^n de distâncias e depois construir a matriz predecessora Π a partir de D^n.
- Isto pode ser implementando em tempo $\Theta(n^3)$.

Construindo o caminho

- Outra maneira é computar a matriz predecessora Π *"on-line"*, da mesma forma que o algoritmo Floyd-Warshall calcula as matrizes D^k.
- Podemos computar $\Pi^0, \Pi^1, \ldots, \Pi^n$, onde $\Pi = \Pi^n$.
- Π^k_{ij} é definida como a matriz predecessora do vértice j no caminho mais curto a partir de i, tendo como vértices intermediários os elementos do conjunto $\{1, 2, \ldots, k\}$.

Construindo o caminho

- Podemos dar uma fórmula recursiva para Π_{ij}^{k}.
- Quando $k = 0$, um caminho mais curto de i para j não tem vértices intermediários, logo:

$$\pi_{ij}^{0} = \begin{cases} nil & \text{if } i = j \text{ or } w_{ij} = \infty \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty \end{cases}$$

Construindo o caminho

- Para $k \geqslant 1$, se tomamos o caminho $i \rightsquigarrow k \rightsquigarrow j$, com $k \neq j$, então o predecessor de j é o mesmo que o predecessor de j escolhido no caminho mais curto de k para j, com vértices intermediários do conjunto $\{1, \ldots, k-1\}$.
- Ou seja, o predecessor de j no caminho p_{kj}^{k-1}.
- Formalmente, temos:

$$\pi_{ij}^{k} = \begin{cases} \pi_{ij}^{k-1} & \text{if } d_{ij}^{k-1} \leqslant d_{ik}^{k-1} + d_{kj}^{k-1} \\ \pi_{kj}^{k-1} & \text{if } d_{ij}^{k-1} > d_{ik}^{k-1} + d_{kj}^{k-1} \end{cases}$$

Conclusões

- Fim!
- Obrigado pela presença