Projeto e Análise de Algoritmos II (MC558)
Caminhos mínimos

Prof. Dr. Ruben Interian

Resumo

Resumo

Revisão do conteúdo

- Podemos resolver o problema dos **caminhos mínimos de fonte única** em grafos sem arcos negativos usando o **Algoritmo de Dijkstra** (ou usando a ordenação topológica em DAGs = *directed acyclic graphs*).
- O algoritmo de **Bellman-Ford** resolve o problema de encontrar os **caminhos mínimos de fonte única**, mesmo quando há arcos de peso negativo (mas não há ciclos negativos).

Objetivo

- Dado um grafo ponderado sem ciclos negativos, encontrar um caminho mínimo de u a v **para todo par** de vértices u, v.

Resumo

1. Revisão do conteúdo e objetivo
2. Caminhos mínimos entre todos os pares de vértices
3. Síntese

Caminhos mínimos entre todos os pares

Problema: Dado um grafo G, ponderado sem ciclos negativos, queremos encontrar um caminho mínimo de u a v **para todo par** de vértices u, v.

Caminhos mínimos entre todos os pares

Problema: Dado um grafo G, ponderado sem ciclos negativos, queremos encontrar um caminho mínimo de u a v **para todo par** de vértices u, v.

Ideia: Podemos executar $|V|$ vezes um dos algoritmos de Caminhos Mínimos de fonte única!

Caminhos mínimos entre todos os pares

- Se (G, w) não possui arcos negativos, usamos **Dijkstra**:

Tipo de fila	Uma execução	$\lvert V \rvert$ vezes
Heap	$O(E \log V)$	$O(VE \log V)$
Fibonacci	$O(V \log V + E)$	$O(V^2 \log V + VE)$

Caminhos mínimos entre todos os pares

- Se (G, w) não possui arcos negativos, usamos **Dijkstra**:

Tipo de fila	Uma execução	$\lvert V \rvert$ vezes
Heap	$O(E \log V)$	$O(VE \log V)$
Fibonacci	$O(V \log V + E)$	$O(V^2 \log V + VE)$

- Se (G, w) possui arcos negativos, usamos **BellmanFord**:

Uma execução	$\lvert V \rvert$ vezes
$O(VE)$	$O(V^2 E)$

Caminhos mínimos entre todos os pares

- Se (G, w) não possui arcos negativos, usamos **Dijkstra**:

Tipo de fila	Uma execução	$\|V\|$ vezes
Heap	$O(E \log V)$	$O(VE \log V)$
Fibonacci	$O(V \log V + E)$	$O(V^2 \log V + VE)$

- Se (G, w) possui arcos negativos, usamos **BellmanFord**:

Uma execução	$\|V\|$ vezes
$O(VE)$	$O(V^2 E)$

Algoritmos mais apropriados para **grafos esparsos**.

O algoritmo de Floyd-Warshall

Floyd-Warshall é um algoritmo baseado em programação dinâmica que resolve o problema diretamente.

- Vamos supor que **não há ciclo de custo negativo**.
- **Floyd-Warshall** é melhor se G for denso (i.e., se $|E|$ é próximo de $\frac{|V|(|V|-1)}{2}$).
- **Complexidade**: $O(V^3)$.

O algoritmo de Floyd-Warshall

Floyd-Warshall é um algoritmo baseado em programação dinâmica que resolve o problema diretamente.

- Vamos supor que **não há ciclo de custo negativo**.
- **Floyd-Warshall** é melhor se G for denso (i.e., se $|E|$ é próximo de $\frac{|V|(|V|-1)}{2}$).
- **Complexidade**: $O(V^3)$.

Floyd-Warshall vai supor que G é **completo**:

- Se (v_i, v_j) não é aresta, definimos $w(v_i, v_j) = \infty$.

Subproblema

Seja $V = \{1, 2, \ldots, n\}$ o conjunto de vértices. Considere um caminho

$$P = (v_1, v_2, \ldots, v_{l-1}, v_l).$$

- Os vértices internos de P são $\{v_2, \ldots, v_{l-1}\}$.

Subproblema

Seja $V = \{1, 2, \ldots, n\}$ o conjunto de vértices. Considere um caminho

$$P = (v_1, v_2, \ldots, v_{l-1}, v_l).$$

- Os vértices internos de P são $\{v_2, \ldots, v_{l-1}\}$.
- P é chamado k-interno se $\{v_2, \ldots, v_{l-1}\} \subseteq \{1, \ldots, k\}$.

Subproblema

Seja $V = \{1, 2, \ldots, n\}$ o conjunto de vértices. Considere um caminho

$$P = (v_1, v_2, \ldots, v_{l-1}, v_l).$$

- Os vértices internos de P são $\{v_2, \ldots, v_{l-1}\}$.
- P é chamado k-interno se $\{v_2, \ldots, v_{l-1}\} \subseteq \{1, \ldots, k\}$.

Subproblema ótimo

Sejam i e j vértices de G e $k \geq 0$ um inteiro. Encontrar o caminho de custo mínimo **apenas considerando** caminhos k-internos de i até j.

Estrutura de um caminho mínimo

O algoritmo de **Floyd-Warshall** explora a relação entre P, um caminho k-interno de custo mínimo de i até j, e os caminhos mínimos $(k-1)$-internos.

Estrutura de um caminho mínimo

O algoritmo de **Floyd-Warshall** explora a relação entre P, um caminho k-interno de custo mínimo de i até j, e os caminhos mínimos $(k-1)$-internos.

Caso 1: k não é um vértice interno de P.

Estrutura de um caminho mínimo

O algoritmo de **Floyd-Warshall** explora a relação entre P, um caminho k-interno de custo mínimo de i até j, e os caminhos mínimos $(k-1)$-internos.

Caso 1: k não é um vértice interno de P.

- todos os vértices internos de P estão em $\{1,\ldots,k-1\}$

Estrutura de um caminho mínimo

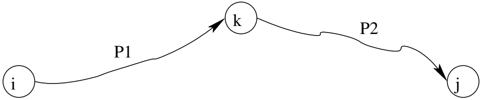
O algoritmo de **Floyd-Warshall** explora a relação entre P, um caminho k-interno de custo mínimo de i até j, e os caminhos mínimos $(k-1)$-internos.

Caso 1: k não é um vértice interno de P.

- todos os vértices internos de P estão em $\{1, \ldots, k-1\}$
- então P é um caminho $(k-1)$-interno de custo mínimo

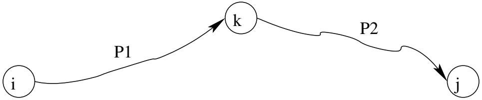
Estrutura de um caminho mínimo

Caso 2: k é um vértice interno de P.
Então P pode ser dividido em dois caminhos P_1 (com início em i e fim em k) e P_2 (com início em k e fim em j).



Estrutura de um caminho mínimo

Caso 2: k é um vértice interno de P.
Então P pode ser dividido em dois caminhos P_1 (com início em i e fim em k) e P_2 (com início em k e fim em j).



- P_1 é um caminho mínimo de i a k com vértices internos em $\{1, \ldots, k-1\}$;
- P_2 é um caminho mínimo de k a j com vértices internos em $\{1, \ldots, k-1\}$.

Recorrência para caminhos mínimos

Seja $d_{ij}^{(k)}$ o peso de um caminho k-interno mínimo de i a j.

- Se $k = 0 \Rightarrow d_{ij}^{(0)} = w(i, j)$.

Recorrência para caminhos mínimos

Seja $d_{ij}^{(k)}$ o peso de um caminho k-interno mínimo de i a j.

- Se $k = 0 \Rightarrow d_{ij}^{(0)} = w(i, j)$.
- Senão, usamos a recorrência:

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

Recorrência para caminhos mínimos

Seja $d_{ij}^{(k)}$ o peso de um caminho k-interno mínimo de i a j.

- Se $k = 0 \Rightarrow d_{ij}^{(0)} = w(i,j)$.
- Senão, usamos a recorrência:

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

Veja que $d_{ij}^{(n)} = \mathrm{dist}(i,j)$. (Por quê?)

Recorrência para caminhos mínimos

Seja $d_{ij}^{(k)}$ o peso de um caminho k-interno mínimo de i a j.

- Se $k = 0 \Rightarrow d_{ij}^{(0)} = w(i, j)$.
- Senão, usamos a recorrência:

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

Veja que $d_{ij}^{(n)} = \text{dist}(i, j)$. (Por quê?)

- Calculamos as matrizes $D^{(k)} = (d_{ij}^{(k)})$ para $k = 1, 2, \ldots, n$.

Recorrência para caminhos mínimos

Seja $d_{ij}^{(k)}$ o peso de um caminho k-interno mínimo de i a j.

- Se $k = 0 \Rightarrow d_{ij}^{(0)} = w(i, j)$.
- Senão, usamos a recorrência:

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

Veja que $d_{ij}^{(n)} = \text{dist}(i, j)$. (Por quê?)

- Calculamos as matrizes $D^{(k)} = (d_{ij}^{(k)})$ para $k = 1, 2, \ldots, n$.
- A solução do problema é $D^{(n)}$.

Algoritmo de Floyd-Warshall

- Entrada: Grafo direcionado e ponderado G dado pela matriz de adj. $W = (w(i,j))$, $n \times n$.
- Saída: Matriz $D^{(n)}$.

Algoritmo de Floyd-Warshall

- Entrada: Grafo direcionado e ponderado G dado pela matriz de adj. $W = (w(i,j))$, $n \times n$.
- Saída: Matriz $D^{(n)}$.

FloydWarshall (W)

1: $D^{(0)} \leftarrow W$
2: **para cada** $k \leftarrow 1, \ldots, n$ **faça**
3:     **para cada** $i \leftarrow 1, \ldots, n$ **faça**
4:         **para cada** $j \leftarrow 1, \ldots, n$ **faça**
5:             $d_{ij}^{(k)} \leftarrow \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
devolva $D^{(n)}$

Complexidade:

Algoritmo de Floyd-Warshall

- Entrada: Grafo direcionado e ponderado G dado pela matriz de adj. $W = (w(i, j))$, $n \times n$.
- Saída: Matriz $D^{(n)}$.

FloydWarshall (W)

1: $D^{(0)} \leftarrow W$
2: **para cada** $k \leftarrow 1, \ldots, n$ **faça**
3: **para cada** $i \leftarrow 1, \ldots, n$ **faça**
4: **para cada** $j \leftarrow 1, \ldots, n$ **faça**
5: $d_{ij}^{(k)} \leftarrow \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
devolva $D^{(n)}$

Complexidade: $O(V^3)$.

Algoritmo de Floyd-Warshall

O algoritmo devolve a matriz de distâncias. Como encontrar os caminhos?

Podemos devolver a **matriz de predecessores** $\Pi = (\pi_{ij})$. Nessa matriz:

- Se $i = j$, ou se não existe caminho de i a j, $\pi_{ij} = \text{NIL}$.

Algoritmo de Floyd-Warshall

O algoritmo devolve a matriz de distâncias. Como encontrar os caminhos?

Podemos devolver a **matriz de predecessores** $\Pi = (\pi_{ij})$. Nessa matriz:

- Se $i = j$, ou se não existe caminho de i a j, $\pi_{ij} = \text{NIL}$.
- Caso contrário, π_{ij} é o predecessor de j em um caminho mínimo a partir de i.

Algoritmo de Floyd-Warshall

O algoritmo devolve a matriz de distâncias. Como encontrar os caminhos?

Podemos devolver a **matriz de predecessores** $\Pi = (\pi_{ij})$. Nessa matriz:

- Se $i = j$, ou se não existe caminho de i a j, $\pi_{ij} = \text{NIL}$.
- Caso contrário, π_{ij} é o predecessor de j em um caminho mínimo a partir de i.

Como calcular? Obtemos uma sequência de matrizes $\Pi^{(0)}, \Pi^{(1)}, \ldots, \Pi^{(n)}$.

Algoritmo de Floyd-Warshall

O algoritmo devolve a matriz de distâncias. Como encontrar os caminhos?

Podemos devolver a **matriz de predecessores** $\Pi = (\pi_{ij})$. Nessa matriz:

- Se $i = j$, ou se não existe caminho de i a j, $\pi_{ij} = \text{NIL}$.
- Caso contrário, π_{ij} é o predecessor de j em um caminho mínimo a partir de i.

Como calcular? Obtemos uma sequência de matrizes $\Pi^{(0)}, \Pi^{(1)}, \ldots, \Pi^{(n)}$.

Quando $k = 0$:

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

Algoritmo de Floyd-Warshall

Para calcular $\pi_{ij}^{(k)}$ se $k \geq 1$, seja P um caminho k-interno mínimo de i a j. Precisamos o predecessor de j:

- **Caso 1**: k não aparece em P.
  - Usamos o predecessor de um caminho $(k-1)$-interno de i a j.

Algoritmo de Floyd-Warshall

Para calcular $\pi_{ij}^{(k)}$ se $k \geq 1$, seja P um caminho k-interno mínimo de i a j. Precisamos o predecessor de j:

- **Caso 1**: k não aparece em P.
  - Usamos o predecessor de um caminho $(k-1)$-interno de i a j.
- **Caso 2**: k aparece em P.
  - Usamos o predecessor de um caminho $(k-1)$-interno de k a j.

Algoritmo de Floyd-Warshall

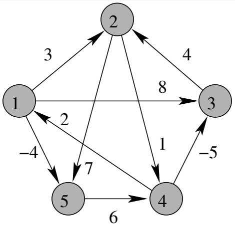
Para calcular $\pi_{ij}^{(k)}$ se $k \geq 1$, seja P um caminho k-interno mínimo de i a j. Precisamos o predecessor de j:

- **Caso 1**: k não aparece em P.
  - Usamos o predecessor de um caminho $(k-1)$-interno de i a j.
- **Caso 2**: k aparece em P.
  - Usamos o predecessor de um caminho $(k-1)$-interno de k a j.

Formalmente,

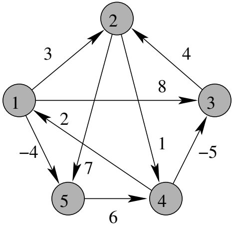
$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{se } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, \\ \pi_{kj}^{(k-1)} & \text{se } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}. \end{cases}$$

Exemplo



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

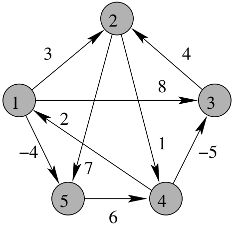
Exemplo



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

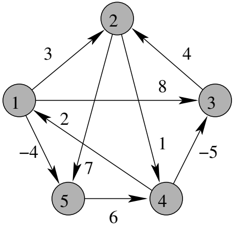
$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \color{red}{5} & -5 & 0 & \color{red}{-2} \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(1)} = \begin{pmatrix} N & 1 & 1 & N & 1 \\ N & N & N & 2 & 2 \\ N & 3 &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

Exemplo



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

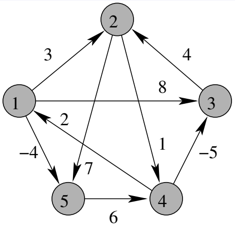
Exemplo



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

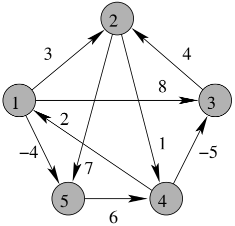
$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & \color{red}{4} & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \color{red}{5} & \color{red}{11} \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(2)} = \begin{pmatrix} N & 1 & 1 & \color{red}{2} & 1 \\ N & N & N & 2 &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

Exemplo



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

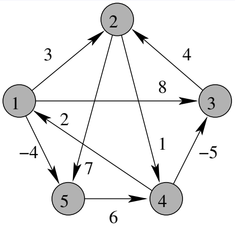
Exemplo



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

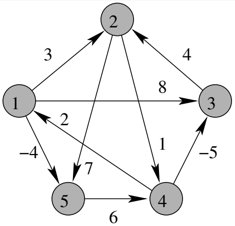
$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(3)} = \begin{pmatrix} N & 1 & 1 & 2 & 1 \\ N & N & N & 2 & 2 \\ N & 3 & N & 2 & 2 \\ 4 & 3 & 4 & N & 1 \\ N &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

Exemplo



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

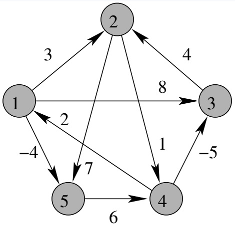
Exemplo



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

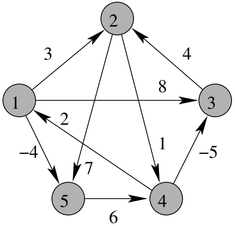
$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(4)} = \begin{pmatrix} N & 1 & 4 & 2 & 1 \\ 4 & N & 4 & 2 & 1 \\ 4 & 3 & N & 2 & 1 \\ 4 & 3 & 4 & N & 1 \\ 4 & 3 & 4 & 5 & N \end{pmatrix}$$

Exemplo



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

Exemplo



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

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

Fecho transitivo de grafos direcionados

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

O fecho transitivo de $G = (V, E)$ é o grafo $G^* = (V, E^*)$ onde

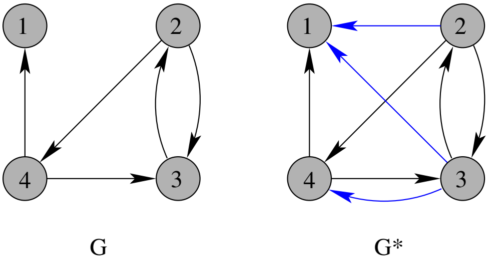
$$E^* = \{(i, j) : \text{existe um caminho de } i \text{ a } j \text{ em } G\}.$$

Fecho transitivo de grafos direcionados

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

O fecho transitivo de $G = (V, E)$ é o grafo $G^* = (V, E^*)$ onde

$$E^* = \{(i, j) : \text{existe um caminho de } i \text{ a } j \text{ em } G\}.$$



Fecho transitivo de grafos direcionados

Determinando o fecho transitivo de $G = (V, E)$:

1. Atribuír custo 1 a cada arco;
2. Executar **FloydWarshall** em tempo $\Theta(V^3)$;
3. Existe um caminho de i a j se e somente se $d_{ij} < |V|$.

Fecho transitivo de grafos direcionados

Determinando o fecho transitivo de $G = (V, E)$:

1. Atribuír custo 1 a cada arco;
2. Executar **FloydWarshall** em tempo $\Theta(V^3)$;
3. Existe um caminho de i a j se e somente se $d_{ij} < |V|$.

Na prática, podemos modificar o algoritmo **FloydWarshall**:

- Substituir min por $\vee$ (OU lógico).
- Substituir $+$ por $\wedge$ (E lógico).

Este algoritmo modificado possui a **mesma complexidade** assintótica, mas é um pouco mais eficiente (economiza tempo e espaço).

Fecho transitivo de grafos direcionados

Defina $t_{i,j}^{(k)}$ o valor booleano:

- TRUE se existe caminho k-interno de i a j,
- FALSE se **não** existe caminho k-interno de i a j.

Fecho transitivo de grafos direcionados

Defina $t_{i,j}^{(k)}$ o valor booleano:

- TRUE se existe caminho k-interno de i a j,
- FALSE se **não** existe caminho k-interno de i a j.

Para $k = 0$:

$$t_{ij}^{(0)} = \begin{cases} 0 & \text{se } i \neq j \text{ e } (i,j) \notin E, \\ 1 & \text{se } i = j \text{ ou } (i,j) \in E. \end{cases}$$

Fecho transitivo de grafos direcionados

Defina $t_{i,j}^{(k)}$ o valor booleano:

- TRUE se existe caminho k-interno de i a j,
- FALSE se **não** existe caminho k-interno de i a j.

Para $k = 0$:

$$t_{ij}^{(0)} = \begin{cases} 0 & \text{se } i \neq j \text{ e } (i,j) \notin E, \\ 1 & \text{se } i = j \text{ ou } (i,j) \in E. \end{cases}$$

Para $k \geq 1$:

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}\right).$$

Fecho transitivo de grafos direcionados

Defina $t_{i,j}^{(k)}$ o valor booleano:

- TRUE se existe caminho k-interno de i a j,
- FALSE se **não** existe caminho k-interno de i a j.

Para $k = 0$:

$$t_{ij}^{(0)} = \begin{cases} 0 & \text{se } i \neq j \text{ e } (i,j) \notin E, \\ 1 & \text{se } i = j \text{ ou } (i,j) \in E. \end{cases}$$

Para $k \geq 1$:

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}\right).$$

Calculamos as matrizes $T^{(k)} = (t_{ij}^{(k)})$.

Algoritmo de Transitive-Closure

- Entrada: Matriz de adjacência A de G.
- Saída: Matriz de adjacência $T^{(n)}$ de G^*.

Algoritmo de Transitive-Closure

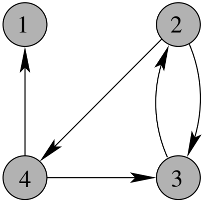
- Entrada: Matriz de adjacência A de G.
- Saída: Matriz de adjacência $T^{(n)}$ de G^*.

TransitiveClosure (A)

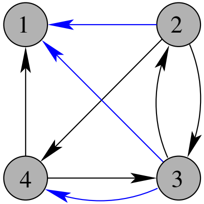
1: $T^{(0)} \leftarrow A + I_n$
2: **para cada** $k \leftarrow 1, \ldots, n$ **faça**
3: **para cada** $i \leftarrow 1, \ldots, n$ **faça**
4: **para cada** $j \leftarrow 1, \ldots, n$ **faça**
5: $t_{ij}^{(k)} \leftarrow t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}\right)$
devolva $T^{(n)}$

Complexidade: $O(V^3)$.

Exemplo



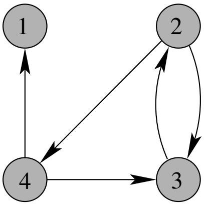
G



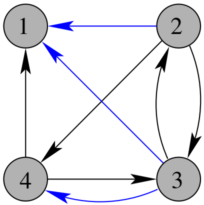
G*

$$T^{(0)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

Exemplo



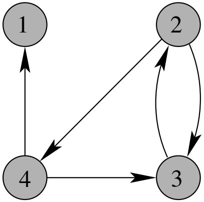
G



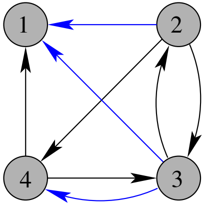
G*

$$T^{(0)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

Exemplo



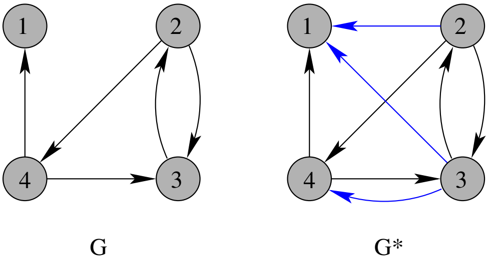
G



G*

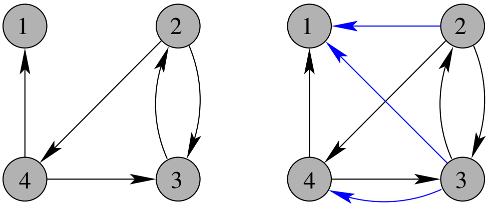
$$T^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

Exemplo



$$T^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & \color{red}{1} \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

Exemplo

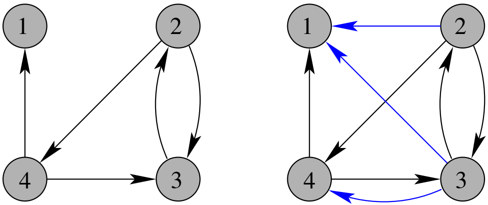


G

G*

$$T^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

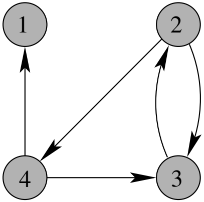
Exemplo



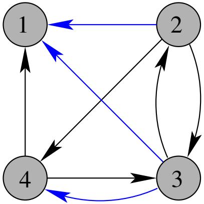
G   G*

$$T^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Exemplo



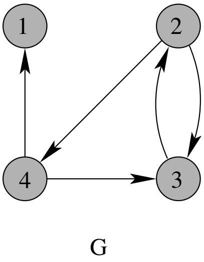
G

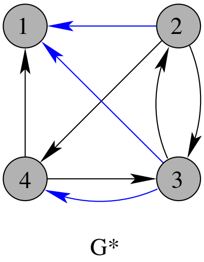


G*

$$T^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Exemplo





$$T^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad T^{(4)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ {\color{red}1} & 1 & 1 & 1 \\ {\color{red}1} & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Intuição: por que os algoritmos funcionam em grafos não direcionados?

Por que os algoritmos de caminhos mínimos funcionam em **grafos não direcionados**?

- Todos os algoritmos assumem que se existe um arco (v, w), podemos usar esse arco no caminho para chegar até w a partir de um vértice x que alcança v.
- Em grafos **não direcionados**, cada aresta (v, w) pode ser usada:
  - No caminho para chegar até w a partir de um vértice x que alcança v;
  - No caminho para chegar até v a partir de um vértice x que alcança w.
- Ou seja, se substituímos cada aresta entre v e w, pelos arcos (v, w) e (w, v), podemos usar os mesmos algoritmos!

Intuição: por que os algoritmos funcionam em grafos não direcionados?

Por que os algoritmos de caminhos mínimos funcionam em **grafos não direcionados**?

- Todos os algoritmos assumem que se existe um arco (v, w), podemos usar esse arco no caminho para chegar até w a partir de um vértice x que alcança v.
- Em grafos **não direcionados**, cada aresta (v, w) pode ser usada:
  - No caminho para chegar até w a partir de um vértice x que alcança v;
  - No caminho para chegar até v a partir de um vértice x que alcança w.
- Ou seja, se substituímos cada aresta entre v e w, pelos arcos (v, w) e (w, v), podemos usar os mesmos algoritmos!

Veja que é perfeitamente possível considerar grafos "**mistos**": aqueles que podem incluir arestas direcionadas e também arcos. Mais uma vez, podemos usar os mesmos algoritmos nestes grafos!

Resumo

1. Revisão do conteúdo e objetivo
2. Caminhos mínimos entre todos os pares de vértices
3. Síntese

Síntese

- **Floyd-Warshall** é um algoritmo baseado em programação dinâmica que resolve o problema dos caminhos mínimos entre **todos os pares** de vértices.
- O algoritmo **Floyd-Warshall** deve ser usado se o grafo é denso.
- Mas, em muitos casos mesmo o grafo sendo "não tão denso", o algoritmo **Floyd-Warshall** se comporta muito bem na prática pela sua simplicidade e menores constantes escondidas na notação assintótica.

Material bibliográfico e exercícios

T. Cormen et al. Algoritmos - Teoria e Prática (3a ed.). – **Cap. 25**

Exercícios: ver exercícios no final dos (sub)capítulos do Cap. 25.

Dúvidas

Dúvidas?