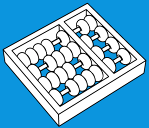
Caminhos mínimos

MO417 - Complexidade de Algoritmos I

23

06/24



Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

UNICAMP

“Sometimes the most productive thing you can do is relax.”

Mark Black

Algoritmo de Bellman-Ford

Arestas vs ciclos de peso negativo

- O algoritmo de Dijkstra resolve o Problema dos Caminhos Mínimos quando (G, w) **NÃO TEM ARESTAS DE PESO NEGATIVO**.
- Quando (G, w) possui arestas negativas, o algoritmo de Dijkstra não funciona.
- Uma das dificuldades com arestas negativas é a possível existência de **CICLOS DE PESO NEGATIVO** ou simplesmente ciclos negativos.

Ciclos negativos — uma dificuldade

- O Problema dos Caminhos Mínimos para instâncias com ciclos negativos é NP-**DIFÍCIL**.
  - Acreditamos que **NÃO** existem algoritmos eficientes para resolver problemas NP-difíceis.
  - Assim, vamos nos restringir ao Problema de Caminhos Mínimos **SEM CICLOS NEGATIVOS**.
- Um algoritmo que resolve o problema restrito é o algoritmo de Bellman-Ford, que também é baseado em relaxação.

Definição do problema

Problema

Entrada: *Um grafo direcionado* $G = (V, E)$*, uma função de peso* w *nas arestas e um vértice origem* s.

Saída: FALSE, *se existe um ciclo negativo atingível a partir de* s. *Caso contrário, além de* TRUE, *também devolve um vetor* d *com* $d[v] = \text{dist}(s,v)$ *para* $v \in V$ *e um vetor* π *definindo uma* ***ÁRVORE DE CAMINHOS MÍNIMOS***.

Ideia do algoritmo de Bellman-Ford

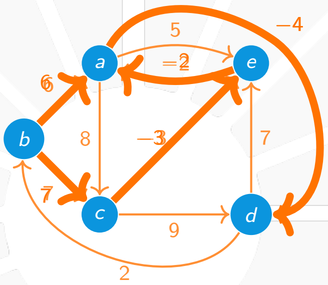
Relaxamento de caminho: Para **QUALQUER** caminho mínimo $(\mathbf{v_0}, \mathbf{v_1}, \dots, \mathbf{v_k})$, queremos relaxar $(\mathbf{v_0}, \mathbf{v_1})$, $(\mathbf{v_1}, \mathbf{v_2})$, $\dots$, $(\mathbf{v_{k-1}}, \mathbf{v_k})$ em ordem.

1. Executamos $\textsc{Relax}$ para todas as arestas:
   $\Rightarrow$ $(\mathbf{v_0}, \mathbf{v_1})$ relaxada.
2. Executamos novamente $\textsc{Relax}$ para todas as arestas:
   $\Rightarrow$ $(\mathbf{v_0}, \mathbf{v_1})$, $(\mathbf{v_1}, \mathbf{v_2})$ relaxadas em ordem.
3. Executamos novamente $\textsc{Relax}$ para todas as arestas:
   $\Rightarrow$ $(\mathbf{v_0}, \mathbf{v_1})$, $(\mathbf{v_1}, \mathbf{v_2})$, $(\mathbf{v_2}, \mathbf{v_3})$ relaxadas em ordem.
4. Repetimos esse passo até $|V| - 1$ vezes. Por quê?
   $\Rightarrow$ $(\mathbf{v_0}, \mathbf{v_1})$, $(\mathbf{v_1}, \mathbf{v_2})$, $\dots$, $(\mathbf{v_{k-1}}, \mathbf{v_k})$ relaxadas em ordem.

Podemos verificar se o grafo contém **CICLOS NEGATIVOS** executando mais uma vez:

- Se algum valor $d[\mathbf{v}]$ diminuir, então há ciclo negativo.

Exemplo



	a	b	c	d	e
d	∞ 6 2	0	∞ 7	∞ 2 -2	∞ 4

Ordem: **(a,c),(a,d),(a,e),(c,d),(c,e),(d,b),(d,e),(e,a),(b,a),(b,c)**.

O algoritmo de Bellman-Ford

Algoritmo: Bellman-Ford(G, w, s)

Initialize-Single-Source(G, s)
para $i \leftarrow 1$ **até** $|V[G]| - 1$
    para cada $(u, v) \in E[G]$
        Relax(u, v, w)
para cada $(u, v) \in E[G]$
    se $d[v] > d[u] + w(u, v)$
        devolva FALSE
devolva TRUE, d, π

Complexidade de tempo: $O(VE)$

Correção do algoritmo Bellman-Ford

Teorema

BELLMAN-FORD *devolve:*

- FALSE, *se existe um ciclo negativo atingível a partir de* s.
- TRUE, *caso contrário; neste caso devolve também:*
  - *Um vetor* d *com* $d[v] = \text{dist}(s,v)$ *para* $v \in V$.
  - *Um vetor* π *definindo uma* ***ÁRVORE DE CAMINHOS MÍNIMOS***.

Correção do algoritmo Bellman-Ford

Primeiro, suponha que não há ciclos negativos atingíveis por **s**.

Considere $\mathbf{v} \in \mathbf{V[G]}$ e os valores de d e π após o primeiro laço:

- Se **v** não é atingível, $d[\mathbf{v}] = \infty$ (por **Inexistência de caminho**).
- Senão, existe caminho mínimo $(\mathbf{v_0}, \mathbf{v_1}, \ldots, \mathbf{v_k})$ de $\mathbf{s} = \mathbf{v_0}$ a $\mathbf{v} = \mathbf{v_k}$.
- Como $k \leq |V| - 1$, então $(\mathbf{v_0}, \mathbf{v_1}),\ (\mathbf{v_1}, \mathbf{v_2}),\ \ldots,\ (\mathbf{v_{k-1}}, \mathbf{v_k})$ foram relaxadas **NESTA ORDEM**.
- Por **Relaxamento de caminho**, $d[\mathbf{v}] = \text{dist}(\mathbf{s},\mathbf{v})$.
- Também, por **Sugrafo de predecessores**: π induz um caminho mínimo de **s** a **v**.

Correção do algoritmo Bellman-Ford

Também temos que mostrar que nesse caso BELLMAN-FORD devolve TRUE.

- Considere d imediatamente após o primeiro laço.
- Nesse instante, $d[\mathbf{v}] = \text{dist}(\mathbf{s},\mathbf{v})$ para todo vértice $\mathbf{v}$.
- Por **Convergência**, sabemos que d nunca mais muda.
- Portanto, o teste da linha 6 falha sempre.
- Concluindo que o algoritmo devolve TRUE.

Correção do algoritmo Bellman-Ford

Suponha agora que (G, w) contenha **CICLO NEGATIVO** alcançável por **s**.

Queremos mostrar que o algoritmo devolve FALSE:

- Seja $C = (\mathbf{v_0}, \mathbf{v_1}, \ldots, \mathbf{v_k} = \mathbf{v_0})$ um ciclo tal que

$$w(C) = \sum_{i=1}^{k} w(\mathbf{v_{i-1}}, \mathbf{v_i}) < 0.$$

- Suponha, por contradição que o algoritmo devolve TRUE.
- Como relaxamos cada aresta $(\mathbf{v_{i-1}}, \mathbf{v_i})$:

$$d[\mathbf{v_i}] \leq d[\mathbf{v_{i-1}}] + w(\mathbf{v_{i-1}}, \mathbf{v_i}).$$

Correção do algoritmo Bellman-Ford

- Somando as desigualdades anteriores para cada aresta do ciclo, temos:

$$\sum_{i=1}^{k} d[\mathbf{v_i}] \leq \sum_{i=1}^{k} (d[\mathbf{v_{i-1}}] + w(\mathbf{v_{i-1}}, \mathbf{v_i}))$$
$$= \sum_{i=1}^{k} d[\mathbf{v_{i-1}}] + \sum_{i=1}^{k} w(\mathbf{v_{i-1}}, \mathbf{v_i}).$$

- Como $\mathbf{v_0} = \mathbf{v_k}$, temos que $\sum_{i=1}^{k} d[\mathbf{v_i}] = \sum_{i=1}^{k} d[\mathbf{v_{i-1}}]$.
- Logo, $0 \leq \sum_{i=1}^{k} w(\mathbf{v_{i-1}}, \mathbf{v_i}) = w(C)$.
- Mas isso é uma contradição, pois C é ciclo negativo.
- Concluindo que, nesse caso, o algoritmo devolve FALSE.

Sistemas de restrições de diferenças

Exemplo

Uma aplicação de caminhos mínimos é encontrar $x_1, x_2, \ldots, x_n$ que satisfaçam:

$$
\begin{aligned}
x_1 - x_2 &\leq 0 \\
x_1 - x_5 &\leq -1 \\
x_2 - x_5 &\leq 1 \\
x_3 - x_1 &\leq 5 \\
x_4 - x_1 &\leq 4 \\
x_4 - x_3 &\leq -1 \\
x_5 - x_3 &\leq -3 \\
x_5 - x_4 &\leq -3
\end{aligned}
$$

Onde:

- x_i representa a hora do evento i
- $x_j - x_i \leq b_k$ significa que deve haver um intervalo de pelo menos b_k horas entre eventos i e j

Exemplo

Podemos reescrever as restrições de forma matricial:

$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} \leq \begin{pmatrix} 0 \\ -1 \\ 1 \\ 5 \\ 4 \\ -1 \\ -3 \\ -3 \end{pmatrix}$$

Algumas soluções:

- $x = (-5, -3, 0, -1, -4)$,
- $x' = (0, 2, 5, 4, 1)$,...

Soluções de sistemas de restrições de diferenças

Lema

Seja $x = (x_1, \ldots, x_n)$ uma solução de um sistema de restrições de diferença $Ax \leq b$ e d uma constante. Então

$$x + d = (x_1 + d, \ldots, x_n + d)$$

também é uma solução de $Ax \leq b$.

Mostraremos a seguir como encontrar uma solução de um sistema $Ax \leq b$ de restrições de diferença resolvendo um problema de caminhos mínimos.

Grafo de restrições

Construímos o chamado **GRAFO DE RESTRIÇÕES** fazendo o seguinte:

1. Primeiro criamos um grafo em que:
   - Cada vértice $\mathbf{v_i}$ corresponde a uma variável x_i.
   - Cada aresta $(\mathbf{v_i}, \mathbf{v_j})$ tem peso b_k e corresponde a uma restrição $x_j - x_i \leq b_k$.
   - Logo, a matriz A do sistema de restrições corresponde à transposta matriz de incidência desse grafo obtido
2. Finalmente, adicionamos um vértice especial $\mathbf{v_0}$ e uma aresta de $\mathbf{v_0}$ a cada outro vértice $\mathbf{v_i}$ com peso 0.

Grafo de restrições

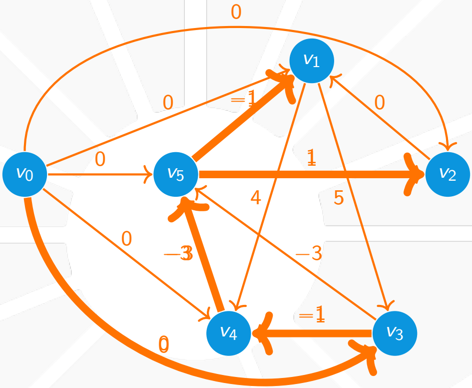
Formalmente, dado o sistema $Ax \leq b$ de restrições de diferença, construímos o grafo $G = (\mathbf{V}, \mathbf{E})$ tal que:

- $\mathbf{V} = \{\mathbf{v_0}, \mathbf{v_1}, \dots, \mathbf{v_n}\}$.
- $\mathbf{E} = \{(\mathbf{v_0}, \mathbf{v_1}), \dots, (\mathbf{v_0}, \mathbf{v_n})\} \cup \{(\mathbf{v_i}, \mathbf{v_j}) : \mathbf{x_j} - \mathbf{x_i} \leq \mathbf{b_k} \text{ é restrição}\}$

E associamos pesos:

- $w(\mathbf{v_i}, \mathbf{v_j}) = \begin{cases} b_k & \text{se } x_j - x_i \leq b_k \text{ é restrição} \\ 0 & \text{se } i = 0 \end{cases}$

Grafo de restrições



Uma solução viável é $\mathbf{x} = (-\mathbf{5}, -\mathbf{3}, \mathbf{0}, -\mathbf{1}, -\mathbf{4})$.

Grafo de restrições

Teorema

Seja $Ax \leq b$ um sistema de restrições de diferença e $G = (V, E)$ o grafo de restrições associado. Então:

- *Se G não contém ciclos negativos,*

$$x = (\mathrm{dist}(v_0, v_1), \mathrm{dist}(v_0, v_2), \ldots, \mathrm{dist}(v_0, v_n))$$

  é uma solução viável do sistema.

- *Se G contém ciclos negativos, o sistema não possui solução. viável.*

Resolvendo um sistema de restrições de diferença

Para **RESOLVER O SISTEMA**, basta resolver caminhos mínimos:

- Executamo BELLMAN-FORD a partir de $\mathbf{v_0}$ no grafo de restrições G.
- Como todo vértice é atingível de $\mathbf{v_0}$, se houver ciclo negativo, o algoritmo devolve FALSE.
- Se não existir ciclo negativo, então obtemos a solução $x = (d[\mathbf{v_1}], d[\mathbf{v_2}], \ldots, d[\mathbf{v_n}])$.

O **TEMPO DE EXECUÇÃO** é calculado como segue:

- A matriz A tem dimensões $m \times n$.
- Então, G possui $n + 1$ vértices e $n + m$ arestas.
- Logo, o tempo de execução de BELLMAN-FORD em G é $O((n + 1)(n + m)) = O(n^2 + nm)$.

Caminhos mínimos entre todos os pares de vértices

Novo problema

Dado grafo ponderado (G, w) sem ciclos negativos, queremos encontrar um caminho mínimo de **u** a **v** para **TODO** par de vértices $\mathbf{u}, \mathbf{v}$.

Algoritmos para grafos esparsos

Podemos executar $|V|$ vezes um algoritmo de Caminhos Mínimos com Mesma Origem:

- Se (G, w) não possui arestas negativas, usamos Dijkstra:

Tipo de fila	Uma vez	$\|V\|$ 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 arestas negativas, usamos Bellman-Ford:

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

O algoritmo de Floyd-Warshall

Floyd-Warshall é um algoritmo que resolve o problema diretamente e é melhor se G for **DENSO**.

- Um grafo é denso se $|E| = \Omega(V^2)$.
- É basado em programação dinâmica.
- Resolve o problema em tempo $O(V^3)$.
- Supomos que G é completo:
  $\Rightarrow$ Se (i, j) não é aresta, definimos $w(i, j) = \infty$

Subproblema

Para simplificar a discussão, suponha que $\mathbf{V} = \{\mathbf{1}, \mathbf{2}, \ldots, \mathbf{n}\}$.

Considere um caminho $P = (\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_{l-1}}, \mathbf{v_l})$:

- Os **VÉRTICES INTERNOS** de P são $\{\mathbf{v_2}, \ldots, \mathbf{v_{l-1}}\}$.
- P é chamado k-**INTERNO** se $\{\mathbf{v_2}, \ldots, \mathbf{v_{l-1}}\} \subseteq \{\mathbf{1}, \ldots, \mathbf{k}\}$.

Problema (Subproblema ótimo)

*Sejam **i** e **j** vértices de G e k um inteiro com $k \geq 0$. Dentre todos os caminhos k-internos de **i** até **j**, encontre algum que tenha custo mínimo.*

Estrutura de um caminho mínimo

O algoritmo de Floyd-Warshall explora a relação entre:

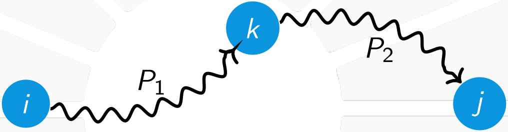
- Um caminho k-interno P de **i** até **j** que tem custo mínimo
- e outros caminhos $(k-1)$-internos.

Caso 1: Se **k** não é um vértice interno de P:

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

Estrutura de um caminho mínimo

Caso 2: Se **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 $\{\mathbf{1}, \ldots, \mathbf{k-1}\}$.
- P_2 é um caminho mínimo de **k** a **j** com vértices internos em $\{\mathbf{1}, \ldots, \mathbf{k-1}\}$.

Recorrência para caminhos mínimos

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

- Se $k = 0$ então $d_{ij}^{(0)} = w(i, j)$.
- Senão, caímos em algum dos dois casos anteriores.

Obtemos a seguinte recorrência:

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

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

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

Algoritmo de Floyd-Warshall

Problema

Entrada: *Matriz* $W = (w(i,j))$ *com dimensões* $n \times n$*.*
Saída: *Matriz* $D^{(n)}$*.*

Algoritmo: $\textsc{Floyd-Warshall}(W, n)$

```
D^(0) ← W
para k ← 1 até n
    para i ← 1 até n
        para j ← 1 até n
            d_ij^(k) ← min(d_ij^(k−1), d_ik^(k−1) + d_kj^(k−1))
devolva D^(n)
```

$$D^{(0)} \leftarrow W$$

$$d_{ij}^{(k)} \leftarrow \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$$

Complexidade: $O(V^3)$.

Como encontrar os caminhos?

Devolvemos matriz de predecessores $\Pi = (\pi_{ij})$

(a) $\pi_{ij} = \text{NIL}$ se $\mathbf{i} = \mathbf{j}$ ou se não existe caminho de **i** a **j**,

(b) π_{ij} é o **PREDECESSOR** de **j** em algum caminho mínimo a partir de **i**, caso contrário.

- Calculamos do mesmo modo que $D^{(k)}$.
- Obtemos uma sequência de matrizes $\Pi^{(0)}, \Pi^{(1)}, \ldots, \Pi^{(n)}$

Quando $k = 0$:

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

Como encontrar os caminhos?

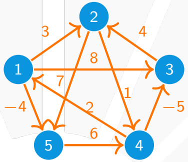
Se $k \geq 1$:

- Considere um caminho k-interno mínimo P de **i** a **j**.
- Obtemos o **PREDECESSOR DE j**:
  - Se k não aparece em P, então usamos o predecessor de um caminho $(k-1)$-interno de **i** a **j**.
  - Se k aparece em P, então usamos o predecessor de um caminho $(k-1)$-interno de **k** a **j**.

Formalmente,

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

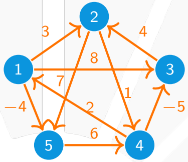
Exemplo



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

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

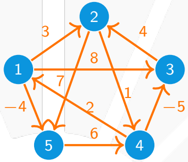
Exemplo



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

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

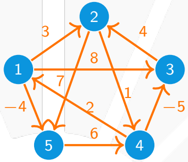
Exemplo



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

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

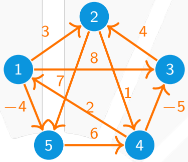
Exemplo



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

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

Exemplo



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

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

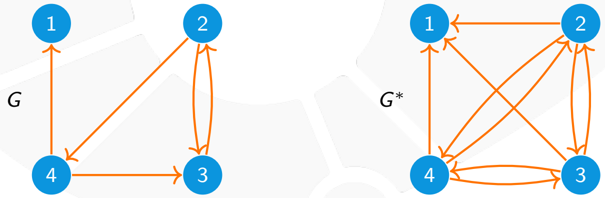
Fecho transitivo de grafos direcionados

Fecho transitivo de grafos direcionados

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

O **FECHO TRANSITIVO** de $G = (\mathbf{V}, \mathbf{E})$ é o grafo $G^* = (\mathbf{V}, \mathbf{E}^*)$ onde:

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



Um detalhe de implementação

Determinando o fecho transitivo de $G = (\mathbf{V}, \mathbf{E})$:

1. Atribuímos peso 1 a cada aresta.
2. Executamos Floyd-Warshall em tempo $\Theta(V^3)$.
3. Existe um caminho de **i** a **j** se e somente se $d_{ij} < |V|$.

Na prática:

- Executamos Floyd-Warshall substituindo:
  - min por $\vee$ (OU lógico).
  - $+$ por $\wedge$ (E lógico).
- Tem a mesma complexidade assintótica.
- 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**.

Para $k = 0$

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

e para $k \geq 1$,

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

Como em Floyd-Warshall, calculamos matrizes $T^{(k)} = (t_{ij}^{(k)})$.

Algoritmo de Transitive-Closure

Problema

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

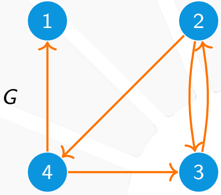
Algoritmo: $\textsc{Transitive-Closure}(A, n)$

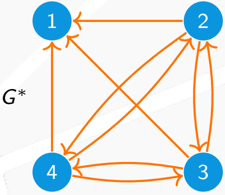
```
T^(0) ← A + I_n
para k ← 1 até n
    para i ← 1 até n
        para j ← 1 até n
            t_ij^(k) ← t_ij^(k-1) ∨ (t_ik^(k-1) + t_kj^(k-1))
devolva T^(n)
```

$T^{(0)} \leftarrow A + I_n$
para $k \leftarrow 1$ **até** n
  para $i \leftarrow 1$ **até** n
    para $j \leftarrow 1$ **até** n
      $t_{ij}^{(k)} \leftarrow t_{ij}^{(k-1)} \vee (t_{ik}^{(k-1)} + t_{kj}^{(k-1)})$
devolva $T^{(n)}$

Complexidade: $O(V^3)$.

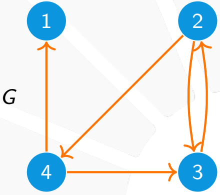
Exemplo

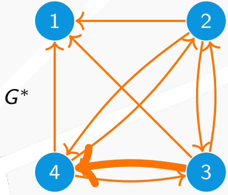




$$T^{(0)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

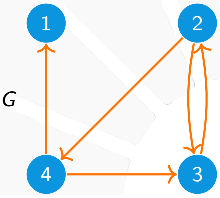
Exemplo

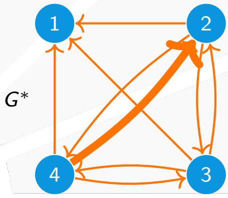




$$T^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & \mathbf{1} \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

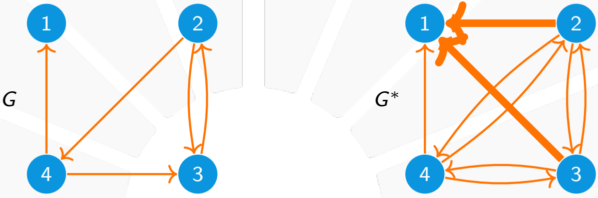
Exemplo





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

Exemplo



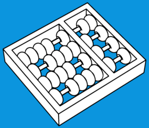
$$T^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad T^{(4)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Caminhos mínimos

MO417 - Complexidade de Algoritmos I

23

06/24



UNICAMP

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br