# Caminhos mínimos de todos os pares

## Algoritmos em Grafos

Marco A L Barbosa

# Conteúdo

O estudo utilizando apenas este material **não é suficiente** para o entendimento do conteúdo. Recomendamos a leitura das referências no final deste material e a resolução (por parte do aluno) de todos os exercícios indicados.

# Introdução

# Introdução

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

# Introdução

- 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)
    - $O(V^2 E)$, se o grafo é denso $O(V^4)$
- Veremos um algoritmo $O(V^3)$ que não usa nenhuma estrutura de dados especial

# Introdução

- Considerações
  - Supomos que não existem ciclos de pesos negativos
  - Os vértices estão numerados como $1, 2, 3, \dots, n$, onde $n = |V|$

# Introdução

- Entrada
  - Uma matriz $Wn \times n$ que representa os pesos das arestas. Isto é, $W = w_{ij}$, onde

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

- Saída
  - Matriz $Dn \times n = d_{ij}$, onde a entrada $d_{ij}$ contém o peso do caminho mínimo do vértice $i$ até o vértice $j$, ou seja, $d_{ij} = \delta(i,j)$
  - Matriz predecessora $\Pi n \times n = \pi_{ij}$, onde $\pi_{ij}$ é o vértice predecessor de $j$ em um caminho mínimo a partir de $i$

# Algoritmos

# Baseado em multiplicação de matrizes

# Baseado em multiplicação de matrizes

- Não estudaremos este algoritmo

# Algoritmo de Floyd-Warshall

# O algoritmo de Floyd-Warshall

- Algoritmo de programação dinâmica com tempo $\Theta(V^3)$

# O algoritmo de Floyd-Warshall

- Algoritmo de programação dinâmica com tempo $\Theta(V^3)$
- Ideia
  - O caminho mínimo pode ser calculado baseado nos caminhos mínimos para subproblemas já calculados e memorizados

# O algoritmo de Floyd-Warshall

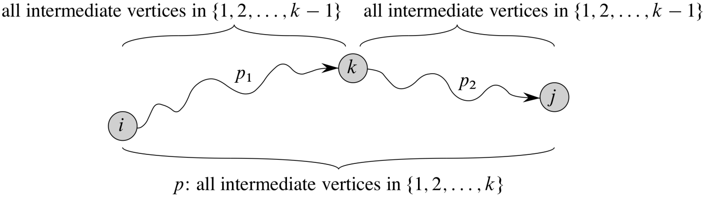
- Algoritmo de programação dinâmica com tempo $\Theta(V^3)$
- Ideia
  - O caminho mínimo pode ser calculado baseado nos caminhos mínimos para subproblemas já calculados e memorizados
- Etapas para resolver um problema com programação dinâmica
  - Caracterizar a estrutura de uma solução ótima
  - Definir recursivamente o valor da solução ótima
  - Computar o valor da solução ótima de baixo para cima (bottom-up)
  - Construir a solução ótima a partir das informações computadas

# O algoritmo de Floyd-Warshall

- 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 $v_l$
- Lembramos que $n = |V|$

# Caracterização da estrutura da solução ótima

- Considere um caminho mínimo $i \overset{p}{\leadsto} j$ com todo os vértices intermediários em $\{1, 2, \ldots, k\}$
  - 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 $i \leadsto j$ com todo os vértices intermediários no conjunto $\{1, 2, \ldots, k-1\}$, também é um caminho mínimo $i \leadsto j$ 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 $i \overset{p_1}{\leadsto} k \overset{p_2}{\leadsto} j$. $p_1$ 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 $p_2$



# Definição recursiva do custo da solução ótima

- Seja $d_{ij}^{(k)}$ o peso de um caminho mínimo $i \rightsquigarrow j$ com todos os vértices intermediários em $\{1, 2, \dots, k\}$

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

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

# O algoritmo de Floyd-Warshall

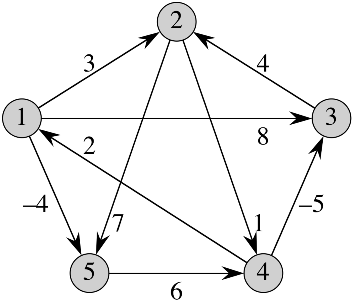
floyd-warshall($W$)
$n$ = $W$.linhas
$D^{(0)}$ = $W$
for $k = 1$ to $n$
    seja $D^{(k)} = \left(d_{ij}^{(k)}\right)$ uma matriz $n \times n$
    for $i = 1$ to $n$
        for $j = 1$ to $n$
            $d_{ij}^{(k)}$ = min($d_{ij}^{(k-1)}$, $d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$)
return $D^{(n)}$

# O algoritmo de Floyd-Warshall

```
floyd-warshall(W)
1 n = W.linhas
2 D^(0) = W
3 for k = 1 to n
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) = 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 $\Theta(n^3) = \Theta(V^3)$

# O algoritmo de Floyd-Warshall



# O algoritmo de Floyd-Warshall

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

# O algoritmo de Floyd-Warshall

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

# O algoritmo de Floyd-Warshall

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

# O algoritmo de Floyd-Warshall

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

# O algoritmo de Floyd-Warshall

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

# O algoritmo de Floyd-Warshall

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

# O algoritmo de Floyd-Warshall

- Como construir um caminho mínimo
  - Calcular a matriz predecessora $\Pi$, 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 $k \geq 1$

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

# O algoritmo de Floyd-Warshall

- Considerando o exercício 25.2-4 e acrescentando o calculo de $\Pi$, podemos reescrever o procedimento `floyd-warshall`

# O algoritmo de Floyd-Warshall

```
floyd-warshall(W)
n = W.linhas
D = W
Π = (π_ij) uma matriz n × n
for i = 1 to n
    for j = 1 to n
        if i = j ou w_ij = ∞
            π_ij = nil
        if i ≠ j e w_ij < ∞
            π_ij = i
for k = 1 to n
    for i = 1 to n
        for j = 1 to n
            if d_ij > d_ik + d_kj
                d_ij = d_ik + d_kj
                π_ij = π_kj
return D, Π
```

# Agoritmo de Johnson para grafos esparsos

# Algoritmo de Johnson para grafos esparsos

- Não estudaremos este algoritmo

# Referências

# Referências


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