TÉCNICO LISBOA

Análise e Síntese de Algoritmos

SSSPs: Bellman-Ford, DAGs. APSPs: Floyd-Warshall.

CLRS Cap. 24 e 25

Instituto Superior Técnico

2022/2023

Contexto

- Revisão [CLRS, Cap.1-13]
  - Fundamentos; notação; exemplos
- Algoritmos em Grafos [CLRS, Cap.21-26]
  - Algoritmos elementares
  - Caminhos mais curtos
  - Fluxos máximos
  - Árvores abrangentes
- Técnicas de Síntese de Algoritmos [CLRS, Cap.15-16]
  - Programação dinâmica
  - Algoritmos greedy
- Programação Linear [CLRS, Cap.29]
  - Algoritmos e modelação de problemas com restrições lineares
- Tópicos Adicionais [CLRS, Cap.32-35]
  - Emparelhamento de Cadeias de Caracteres
  - Complexidade Computacional
  - Algoritmos de Aproximação

Resumo

Algoritmo de Bellman-Ford

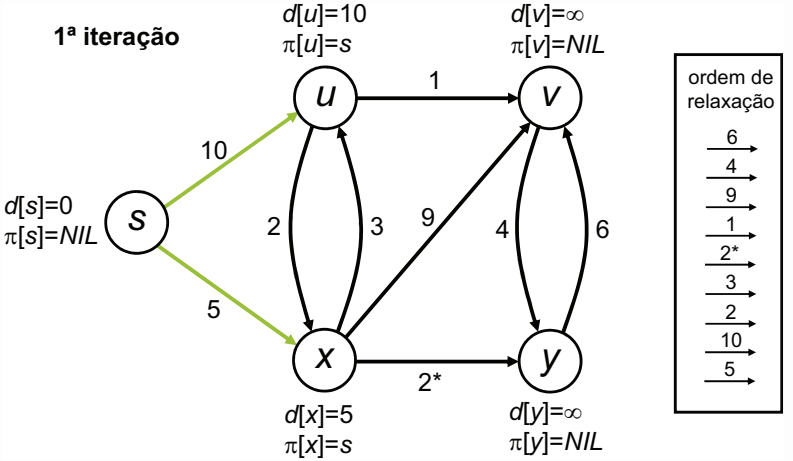
Intuição

- Permite pesos negativos
- Identifica existência de ciclos negativos
- Baseado em sequência de passos de relaxação
- Requer manutenção da estimativa associada a cada vértice

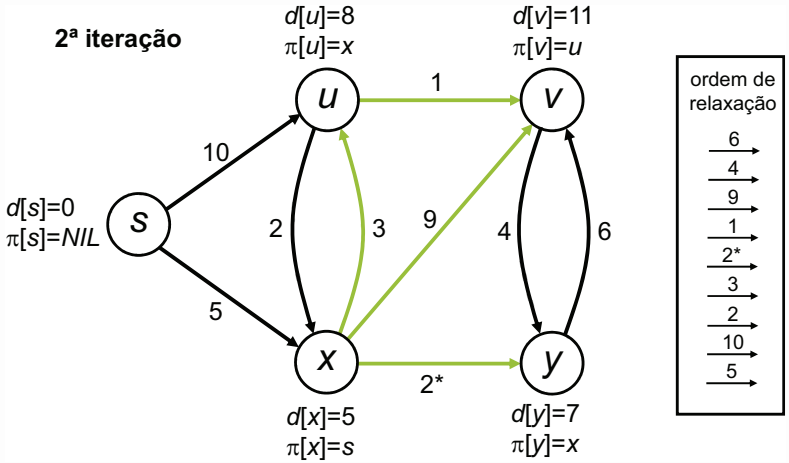
Algoritmo de Bellman-Ford

Bellmand-Ford(G, w, s)
  Initialize-Single-Source(G, s)
  for $i \leftarrow 1$ **to** $|G.V| - 1$ **do**
    for each $(u, v) \in G.E$ **do**
      Relax(u, v, w)
    end for
  end for
  for each $(u, v) \in G.E$ **do**
    if $d[v] > d[u] + w(u, v)$ **then**
      return *FALSE*
    end if
  end for
  return *TRUE*

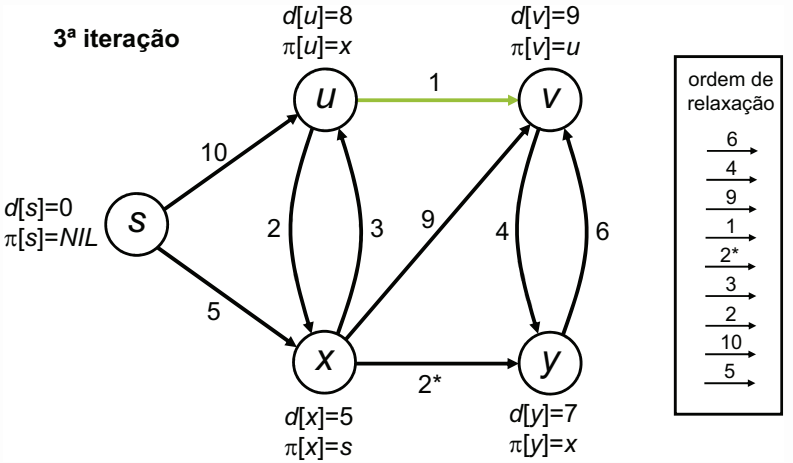
Algoritmo de Bellman-Ford: Exemplo



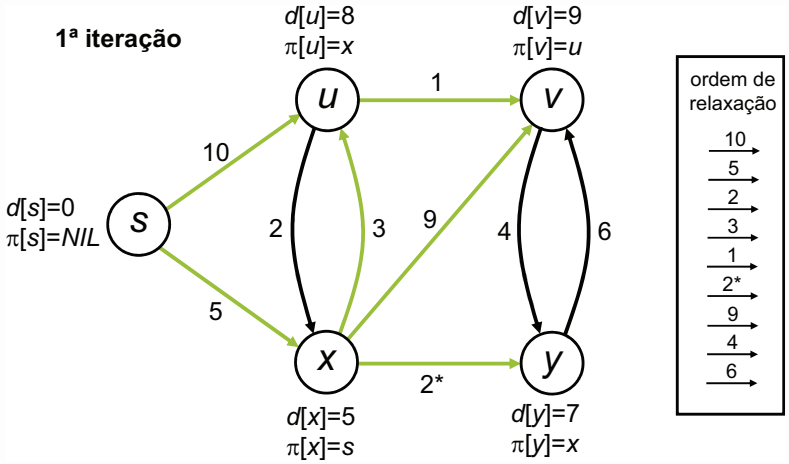
Algoritmo de Bellman-Ford: Exemplo



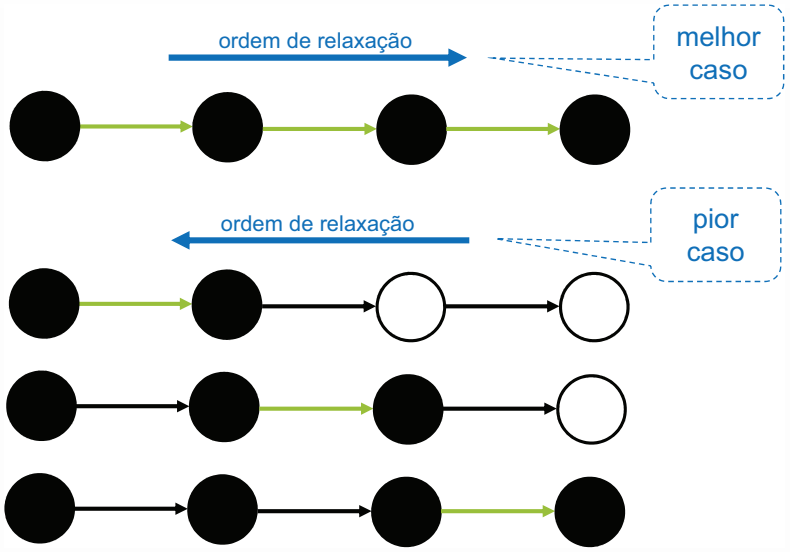
Algoritmo de Bellman-Ford: Exemplo



Algoritmo de Bellman-Ford: Outro Exemplo



Algoritmo de Bellman-Ford



Algoritmo de Bellman-Ford

Complexidade

- Inicialização: $\Theta(V)$
- A complexidade dos ciclos é $O(VE)$
  - Dois ciclos aninhados em V e E
  - Em cada iteração todos os arcos são relaxados
- Complexidade do algoritmo de Bellman-Ford: $O(VE)$

Algoritmo de Bellman-Ford

Correção do Algoritmo

Se $G = (V, E)$ não contém ciclos negativos, então após a aplicação do algoritmo de Bellman-Ford, $d[v] = \delta(s, v)$ para todos os vértices atingíveis a partir de s

- Seja v atingível a partir de s, e seja $p = \langle v_0, v_1, \ldots, v_k \rangle$ um caminho mais curto entre s e v, com $v_0 = s$ e $v_k = v$
- p é simples, pelo que $k \leq |V| - 1$

Prova por Indução

Provar que $d[v_i] = \delta(s, v_i)$ para $i = 1, \ldots, k$, após iteração i sobre os arcos de G, e que valor não é alterado posteriormente

- Base: $d[v_0] = \delta(s, v_0) = 0$ após inicialização (e não se altera)
- Passo indutivo: assumir $d[v_{i-1}] = \delta(s, v_{i-1})$ após iteração $(i-1)$
- Arco (v_{i-1}, v_i) relaxado na iteração i, pelo que $d[v_i] = \delta(s, v_i)$ após iteração i (e não se altera)

Correção do Algoritmo (cont.)

Se $G = (V, E)$ não contém ciclos negativos (atingíveis a partir de s), o algoritmo de Bellman-Ford retorna *TRUE*, caso contrário *FALSE*

- Se não existem ciclos negativos, o resultado anterior assegura que para qualquer arco $(u, v) \in E$, $d[v] \leq d[u] + w(u, v)$, dado que $d[u] = \delta(s, u)$ e $d[v] = \delta(s, v)$, pelo que o teste do algoritmo falha para todo o (u, v) e o valor retornado é *TRUE*
- Caso contrário, na presença de pelo menos um ciclo negativo atingível a partir de s, $c = \langle v_0, v_1, \ldots, v_k \rangle$, onde $v_0 = v_k$, temos que $\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$

Correção do Algoritmo(cont.)

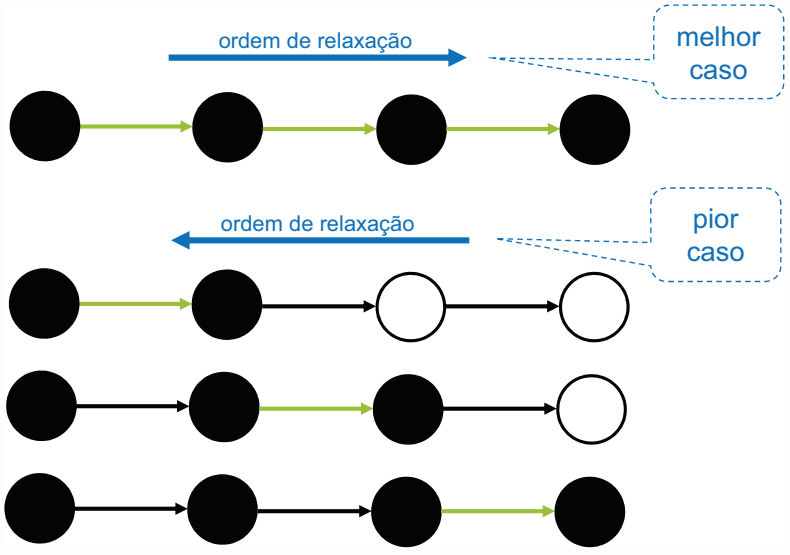
Prova por Contradição

Admitir que algoritmo retorna *TRUE* na presença de ciclo negativo

- Para que devolva *TRUE* é necessário que:
  $$d[v_i] \leq d[v_{i-1}] + w(v_{i-1}, v_i), \text{ para } i = 1, \ldots, k$$
- Somando as desigualdades ao longo do ciclo temos que:
  $$\sum_{i=1}^{k} d[v_i] \leq \sum_{i=1}^{k} d[v_{i-1}] + \sum_{i=1}^{k} w(v_{i-1}, v_i)$$
- Note-se que $\sum_{i=1}^{k} d[v_i] = \sum_{i=1}^{k} d[v_{i-1}]$ por ser um ciclo
- Temos então que $\sum_{i=1}^{k} w(v_{i-1}, v_i) \geq 0$, o que contradiz a existência de um ciclo negativo. Logo, o algoritmo retorna *FALSE*

Intuição

- Um DAG não tem ciclos
- Mesmo com pesos negativos, o caminho mais curto é bem definido
- Ordenar os vértices por ordem topológica
- Passagem única pelos vértices ordenados



DAG-Shortest-Path(G, w, s)
  Ordenação topológica dos vértices de G
  Initialize-Single-Source(G, s)
  for each $u \in G.V$ (por ordem topológica) **do**
    for each $v \in Adj[u]$ **do**
      Relax(u, v, w)
    end for
  end for

Caminhos mais curtos em DAGs

DAG-Shortest-Path(G, w, s)
  Ordenação topológica dos vértices de G
  Initialize-Single-Source(G, s)
  for each $u \in G.V$ (por ordem topológica) **do**
    for each $v \in Adj[u]$ **do**
      Relax(u, v, w)
    end for
  end for

Complexidade

- $\Theta(V + E)$

Análise e Síntese de Algoritmos - 2022/2023 16/33

Caminhos mais curtos em DAGs

Correção do Algoritmo

Dado $G = (V, E)$, dirigido, acíclico, como resultado do algoritmo, temos que $d[v] = \delta(s, v)$ para todo o $v \in V$

- Seja v atingível a partir de s, e seja $p = \langle v_0, v_1, \ldots, v_k \rangle$ um caminho mais curto entre s e v, com $v_0 = s$ e $v_k = v$
- Ordenação topológica implica que analisados por ordem (v_0, v_1), (v_1, v_2), ..., (v_{k-1}, v_k)

Caminhos mais curtos em DAGs

Correção do Algoritmo

Dado $G = (V, E)$, dirigido, acíclico, como resultado do algoritmo, temos que $d[v] = \delta(s, v)$ para todo o $v \in V$

- Seja v atingível a partir de s, e seja $p = \langle v_0, v_1, \ldots, v_k \rangle$ um caminho mais curto entre s e v, com $v_0 = s$ e $v_k = v$
- Ordenação topológica implica que analisados por ordem (v_0, v_1), (v_1, v_2), ..., (v_{k-1}, v_k)

Prova por Indução

$d[v_i] = \delta(s, v_i)$ sempre que cada vértice v_i é terminado

- Base: Estimativa de s não alterada após inicialização; $d[s] = d[v_0] = \delta(s, v_0) = 0$
- Indução: $d[v_{i-1}] = \delta(s, v_{i-1})$ após terminar análise de v_{i-1}
- Relaxação do arco (v_{i-1}, v_i) causa $d[v_i] = \delta(s, v_i)$, pelo que $d[v_i] = \delta(s, v_i)$ após terminar análise de v_i

Análise e Síntese de Algoritmos - 2022/2023 17/33

Resumo

Algoritmo Dijkstra

- Apenas permite pesos não negativos
- Complexidade: $O((V + E)\ log\ V)$

Algoritmo Bellman-Ford

- Permite pesos negativos e identifica ciclos negativos
- Complexidade: $O(VE)$

Caminhos mais curtos em DAGs

- Grafos acíclicos (ordenação topológica dos vértices)
- Complexidade: $O(V + E)$

Caminhos mais curtos entre todos os pares

Motivação

Considere que está a gerir o serviço de reencaminhamento de mercadorias mundial de uma operadora

- Existe um conjunto de armazens em locais específicos no mundo
- Existem rotas pré-definidas e com custos associados
- O serviço recebe pedidos tais como:
  "enviar mercadoria X do local A para o local B"
- O serviço deve estar automatizado por forma a conseguir satisfazer todos os possíveis pedidos de reencaminhamento de mercadorias

Caminhos mais curtos entre todos os pares

Motivação

Encontrar caminhos mais curtos entre todos os pares de vértices

- Se pesos não negativos, utilizar algoritmo de Dijkstra, assumindo cada vértice como fonte: $O(V.(V + E)\ lg\ V)$
  (que é $O(V^3\ lg\ V)$ se o grafo for denso)
- Se existem pesos negativos, utilizar algoritmo de Bellman-Ford, assumindo cada vértice como fonte: $O(V.VE)$
  (que é $O(V^4)$ se o grafo é denso)

Objetivo: Encontrar algoritmos mais eficientes

Caminhos mais curtos entre todos os pares

Representação

Dado um grafo $G = (V, E)$, e $n = |V|$, podemos representar G através de uma matriz de adjacências

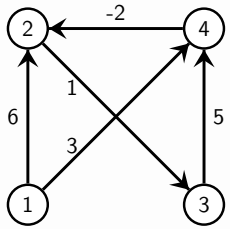
- Pesos dos arcos: matriz W $(n \times n)$

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

- Representação dos caminhos mais curtos: matriz D $(n \times n)$
  - d_{ij} é o peso do caminho mais curto entre os vértices i e j
  - $d_{ij} = \delta(v_i, v_j)$

Exemplo Representação

Grafo



Matriz W

	1	2	3	4
1	0	6	∞	3
2	∞	0	1	∞
3	∞	∞	0	5
4	∞	-2	∞	0

Matriz D

	1	2	3	4
1	0	1	2	3
2	∞	0	1	6
3	∞	3	0	5
4	∞	-2	-1	0

Representação

- Representação dos predecessores: matriz Π $(n \times n)$
- $\pi_{ij} = \text{NIL}$ se $i = j$ ou não existe caminho de i para j
- Caso contrário: π_{ij} denota o predecessor de j num caminho mais curto de i para j
- Para cada vértice i, definimos um sub-grafo de predecessores de G para i, $G_{\pi,i} = (V_{\pi,i}, E_{\pi,i})$:

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

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

- Sub-grafo de predecessores $G_{\pi,i}$ é induzido pela linha i da matriz Π

- Propriedade de sub-estrutura óptima dos caminhos mais curtos: sub-caminhos de caminhos mais curtos são também caminhos mais curtos
- $d_{ij}^{(m)}$: denota o peso mínimo dos caminhos do vértice i para o vértice j não contendo mais do que m arcos
- Com $m = 0$, existe caminho de i para j se e só se $i = j$

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

- Para $m \geq 1$:

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

Extend-Shortest-Paths(D, W)

```
n = rows[W]
D' = new matrix(n × n)
for i = 1 to n do
  for j = 1 to n do
    d'_ij = ∞
    for k = 1 to n do
      d'_ij = min(d'_ij, d_ik + w_kj)
    end for
  end for
end for
return D'
```

- Calcula $D^{(m)} = D'$ a partir de $D^{(m-1)} = D$

Solução Recursiva

Observações

- Nota: $D^{(1)} = W$
- Calcular sequência de matrizes $D^{(1)}, \ldots, D^{(n-1)}$, onde $D^{(n-1)}$ contém os pesos dos caminhos mais curtos entre todos os pares de vértices
- Calculo de $D^{(m)}$ é feito utilizando apenas de $D^{(m-1)}$ (e W)

Solução Recursiva

Observações

- Nota: $D^{(1)} = W$
- Calcular sequência de matrizes $D^{(1)}, \ldots, D^{(n-1)}$, onde $D^{(n-1)}$ contém os pesos dos caminhos mais curtos entre todos os pares de vértices
- Calculo de $D^{(m)}$ é feito utilizando apenas de $D^{(m-1)}$ (e W)

Complexidade

- $\Theta(n^3)$ para cada matriz
- $\Theta(n^4)$ para cálculo de $D^{(n)}$

Solução Recursiva

Observações

- Nota: $D^{(1)} = W$
- Calcular sequência de matrizes $D^{(1)}, \ldots, D^{(n-1)}$, onde $D^{(n-1)}$ contém os pesos dos caminhos mais curtos entre todos os pares de vértices
- Calculo de $D^{(m)}$ é feito utilizando apenas de $D^{(m-1)}$ (e W)

Complexidade

- $\Theta(n^3)$ para cada matriz
- $\Theta(n^4)$ para cálculo de $D^{(n)}$
  - É possível melhorar complexidade reduzindo número de matrizes calculadas: $O(n^3 \lg n)$
  - A cada iteração, calcular $D^{(2m)}$ em função de $D^{(m)}$ e de $D^{(m)}$ (em vez de W): $D^{(2m)} = \text{Extend-Shortest-Paths}(D^{(m)}, D^{(m)})$

Algoritmo Floyd-Warshall

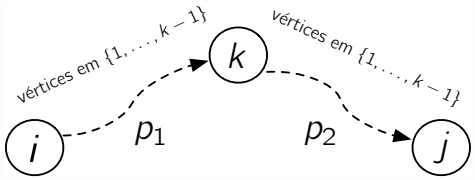
Definições

- Caracterização de caminho mais curto $p = \langle v_1, v_2, \ldots, v_{l-1}, v_l \rangle$
  - Vértices intermédios de caminho p são $\{v_2, \ldots, v_{l-1}\}$
- Considerar todos os caminhos entre i e j com vértices intermédios retirados de um conjunto $\{1, \ldots, k\} \subseteq V$ e seja p um caminho mais curto
  Nota: p é simples
- Se k não é vértice intermédio de p, então todos os vértices intermédios de p estão em $\{1, \ldots, k-1\}$

todos os vértices em $\{1, \ldots, k-1\}$

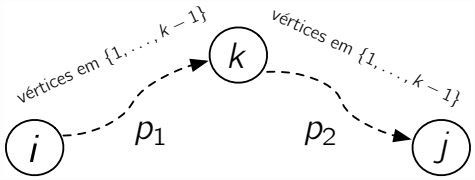
Definições (cont.)

- Se k é vértice intermédio de p, então existem caminhos p_1 e p_2, respetivamente de i para k e de k para j com vértices intermédios em $\{1, \ldots, k\}$
  - k não é vértice intermédio de p_1 e de p_2
  - p_1 e p_2 com vértices intermédios em $\{1, \ldots, k-1\}$



Definições (cont.)

- Se k é vértice intermédio de p, então existem caminhos p_1 e p_2, respetivamente de i para k e de k para j com vértices intermédios em $\{1, \ldots, k\}$
  - k não é vértice intermédio de p_1 e de p_2
  - p_1 e p_2 com vértices intermédios em $\{1, \ldots, k-1\}$



Formulação

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

Floyd-Warshall(D, W)

$n = rows[W]$
$D^{(0)} = W$
for $k = 1$ **to** n **do**
  for $i = 1$ **to** n **do**
    for $j = 1$ **to** n **do**
      $d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$
    end for
  end for
end for
return $D^{(n)}$

Floyd-Warshall(D, W)

$n = rows[W]$
$D^{(0)} = W$
for $k = 1$ **to** n **do**
  for $i = 1$ **to** n **do**
    for $j = 1$ **to** n **do**
      $d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$
    end for
  end for
end for
return $D^{(n)}$

Complexidade

- Tempo: $\Theta(n^3)$
- Espaço: $\Theta(n^3)$

Observação

Podemos evitar uma matriz por cada passo do algoritmo
A linha e a coluna k não são alteradas na iteração k:

$$d_{ik}^{(k)} = \min(d_{ik}^{(k-1)}, d_{ik}^{(k-1)} + d_{kk}^{(k-1)})$$

$$d_{kj}^{(k)} = \min(d_{kj}^{(k-1)}, d_{kk}^{(k-1)} + d_{kj}^{(k-1)})$$

Seja $d_{kk}^{(k-1)} = 0$, é fácil verificar que $d_{ik}^{(k)} = d_{ik}^{(k-1)}$ e $d_{kj}^{(k)} = d_{kj}^{(k-1)}$

(k não pode ser um vértice intermédio no caminho de i para k, nem no caminho de k para j)

Floyd-Warshall(D,W)

$n = rows[W]$
$D = W$
for $k = 1$ **to** n **do**
  for $i = 1$ **to** n **do**
    for $j = 1$ **to** n **do**
      $d_{ij} = \min(d_{ij}, d_{ik} + d_{kj})$
    end for
  end for
end for
return D

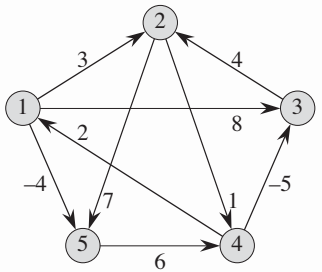
Floyd-Warshall(D,W)

$n = rows[W]$
$D = W$
for $k = 1$ **to** n **do**
  for $i = 1$ **to** n **do**
    for $j = 1$ **to** n **do**
      $d_{ij} = \min(d_{ij}, d_{ik} + d_{kj})$
    end for
  end for
end for
return D

Complexidade

- Tempo: $\Theta(n^3)$
- Espaço: $\Theta(n^2)$

Exemplo [CLRS, Fig 25.1]



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

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

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

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

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

Fecho Transitivo de um Grafo Dirigido

Dado um grafo $G = (V, E)$ dirigido, o fecho transitivo é definido por $G^* = (V, E^*)$ tal que:

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

Algoritmo Floyd-Warshall

Fecho Transitivo de um Grafo Dirigido

Dado um grafo $G = (V, E)$ dirigido, o fecho transitivo é definido por $G^* = (V, E^*)$ tal que:

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

Algoritmo

- Atribuir a cada arco peso 1 e utilizar algoritmo de Floyd-Warshall
- Se $d_{ij} \neq \infty$, então $(i, j) \in E^*$
- Complexidade: $\Theta(n^3)$