

# BCC204 - Teoria dos Grafos

Marco Antonio M. Carvalho

Departamento de Computação  
Instituto de Ciências Exatas e Biológicas  
Universidade Federal de Ouro Preto



## 1 Algoritmo de Bellman-Ford

## Fonte

Este material é baseado no livro

- ▶ Goldbarg, M., & Goldbarg, E. (2012). *Grafos: conceitos, algoritmos e aplicações*. Elsevier.

## Licença

Este material está licenciado sob a Creative Commons BY-NC-SA 4.0. Isto significa que o material pode ser compartilhado e adaptado, desde que seja atribuído o devido crédito, que o material não seja utilizado de forma comercial e que o material resultante seja distribuído de acordo com a mesma licença.

## Arestas de Peso Negativo

Para além das distâncias geográficas, caminhos mais curtos podem modelar diversas outras situações reais, incluindo aquelas que para serem modeladas necessitam de arestas cujo peso é negativo:

- ▶ Movimentações financeiras, nas quais é possível obter lucro ou prejuízo, principalmente quando há utilização de câmbio;
- ▶ Um taxista que recebe mais dinheiro do que gasta com combustível a cada viagem: se o táxi roda vazio, ele gasta mais do que recebe;
- ▶ Um entregador que necessita atravessar um pedágio e pode acabar pagando mais do que recebe para entregar encomendas;
- ▶ A energia gerada e consumida durante uma reação química.

## *Ford-Moore-Bellman?*

Alguns autores denominam o algoritmo de *Ford-Moore-Bellman*, em homenagem a outros três autores que propuseram o mesmo algoritmo em anos diferentes:

- ▶ Lester Ford (1956);
- ▶ Edward Moore (1957);
- ▶ Richard Bellman (1958).

## Princípio

Calcula caminhos mais curtos via programação dinâmica *bottom-up*.

Ao invés de fechar um vértice por iteração, como o algoritmo de Dijkstra, examina todos os vértices de um grafo **orientado** por iteração até que atualizações não sejam possíveis.

Em um grafo com  $n$  vértices, qualquer caminho possui no máximo  $n - 1$  arestas, portanto, cada vértice é examinado no máximo  $n - 1$  vezes.

Com esta estratégia, é possível calcular caminhos mínimos em grafos com arestas de **peso negativo**.

Assim como o algoritmo de *Dijkstra*, baseia-se no princípio de relaxação: uma aproximação da distância da origem até cada vértice é gradualmente atualizada por valores mais precisos até que a solução ótima seja atingida.

## Princípio

Se, em alguma iteração do algoritmo os caminhos até cada um dos vértices permanecerem inalterados, não haverá atualizações nas próximas iterações e o algoritmo pode terminar.

Entretanto, se houver atualizações na última iteração do algoritmo, é sinal de que há pelo menos um ciclo negativo no grafo, dado que algum caminho terá  $n$  arestas ou mais.

## Terminologia

- ▶  $\Gamma^-(i)$ : Conjunto de vértices antecessores do vértice atual;
- ▶  $dt[i]$ : Vetor que armazena a distância entre o vértice de origem e o vértice  $i$ ;
- ▶  $rot[i]$ : Vetor que armazena o índice do vértice anterior ao vértice  $i$ , no caminho cuja distância está armazenada em  $dt[i]$ ;
- ▶  $altera$ : variável *booleana* que indica se houve alguma atualização na iteração atual.



# Algoritmo de Bellman-Ford

**Entrada:** Grafo  $G = (V, E)$  e matriz de pesos  $D = \{d_{ij}\}$  para todos os arcos  $(i, j)$

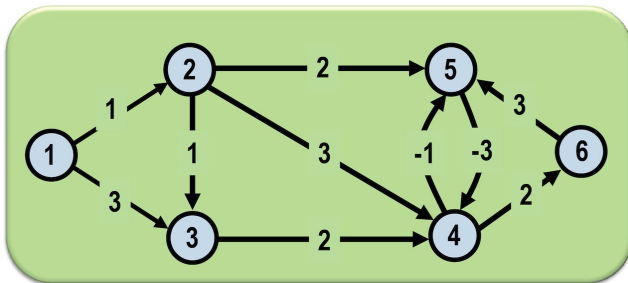
```
1  $dt[1] \leftarrow 0$ ;  $rot[1] \leftarrow \infty$ ; //considerando o vértice 1 como o inicial
2 para  $i \leftarrow 2$  até  $n$  faça
3     se  $\exists (1, i) \in E$  então  $rot[i] \leftarrow 1$ ;  $dt[i] \leftarrow d_{1i}$ ;
4     senão  $rot[i] \leftarrow 0$ ;  $dt[i] \leftarrow \infty$ ;
5 fim
6 para  $k \leftarrow 1$  até  $n-1$  faça
7     altera  $\leftarrow$  falso;
8     para  $i \leftarrow 2$  até  $n$  faça
9         para  $j \in \Gamma^-(i)$  faça
10             se  $dt[i] > dt[j] + d_{ji}$  então
11                  $dt[i] \leftarrow dt[j] + d_{ji}$ ;
12                  $rot[i] \leftarrow j$ ;
13                 altera  $\leftarrow$  verdadeiro; //indica que houve alteração
14             fim
15         fim
16     fim
17     se altera = falso então  $k \leftarrow n$ ;
18 fim
```

# Ciclos de Custo Negativo

## Bellman-Ford – Detecção

Em caminhos sem ciclos, o caminho mais longo consiste em  $n - 1$  arestas, ou iterações no laço principal do algoritmo.

Se na iteração  $n$  do algoritmo alguma atualização de distâncias for feita é detectado o ciclo.



Exemplo de ciclo negativo entre os vértices 4 e 5.

## Complexidade 1

- ▶ Após a inicialização, o laço **para** da linha 7 é repetido por no máximo  $n-1$  vezes;
- ▶ Em cada iteração, são calculados caminhos com  $k$  arestas entre a origem e os demais vértices do grafo;
- ▶ Para cada um dos  $n-1$  vértices, todos seus antecessores são examinados;
- ▶ O vértice original não é atualizado, logo,  $n-2$  antecessores são analisados no máximo;
- ▶ Logo, em uma implementação simples, a complexidade é  $O(n^3)$ .

## Complexidade 2

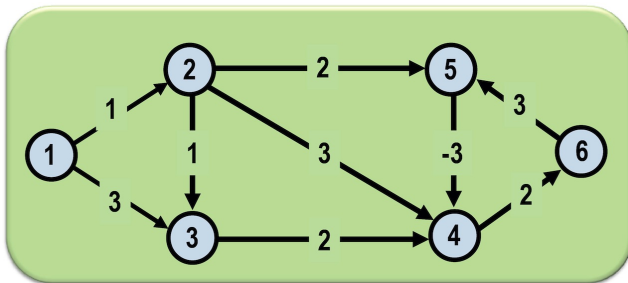
Em 1970, Jin Yen<sup>a</sup> propôs uma implementação deste método de complexidade  $O(nm)$  no pior caso:

- 1 Se  $dt[v]$  não se alterar desde a última vez em que  $\Gamma^+$  foi examinado, então não é necessário examinar seus arcos de saída novamente;
- 2 O comprimento do laço externo é reduzido de  $n - 1$  para  $n/2$  através de uma ordenação linear de vértices e posterior partição dos mesmos.

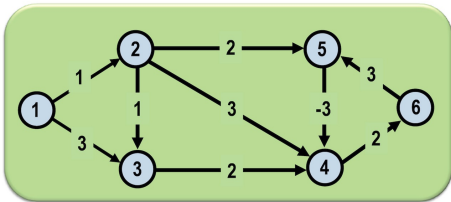
---

<sup>a</sup>Yen, Jin Y. (1970). "An algorithm for finding shortest routes from all source nodes to a given destination in general networks". Quarterly of Applied Mathematics 27: 526–530.

# Exemplo



# Exemplo

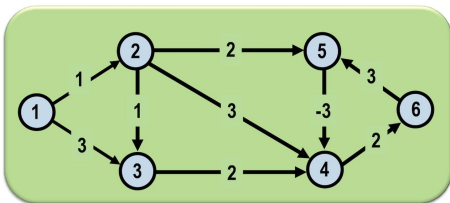


<i>dt</i>				
2	3	4	5	6
1	3	$\infty$	$\infty$	$\infty$

<i>rot</i>				
2	3	4	5	6
1	1	0	0	0

Vetores após a inicialização do algoritmo.

# Exemplo



$$i=2, \Gamma^-(i)=\{1\};$$

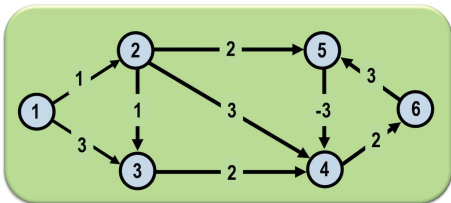
$$\triangleright j=1, dt[1]+d_{12} = 1$$

<i>dt</i>				
2	3	4	5	6
1	3	$\infty$	$\infty$	$\infty$

<i>rot</i>				
2	3	4	5	6
1	1	0	0	0

Iteração  $k=1$  (continua...)

# Exemplo



$$i=3, \Gamma^-(i)=\{1, 2\};$$

$$\triangleright j=1, dt[1]+d_{13} = 3$$

$$\triangleright j=2, dt[2]+d_{23} = \mathbf{2}$$

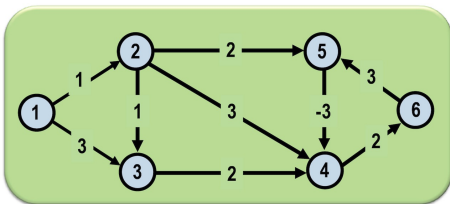
$dt$				
2	3	4	5	6
1	2	$\infty$	$\infty$	$\infty$

$rot$				
2	3	4	5	6
1	2	0	0	0

Iteração  $k=1$  (continua...)



# Exemplo



$i=4, \Gamma^-(i)=\{2, 3, 5\};$

►  $j=2, dt[2]+d_{24} = 4$

►  $j=3, dt[3]+d_{34} = 4$

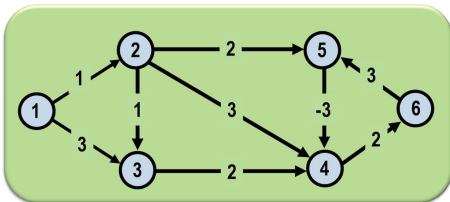
►  $j=5, dt[5]+d_{54} = \infty$

$dt$				
2	3	4	5	6
1	2	4	$\infty$	$\infty$

$rot$				
2	3	4	5	6
1	2	2	0	0

Iteração  $k=1$  (continua...)

# Exemplo



$i=5, \Gamma^-(i)=\{2, 6\};$

►  $j=2, dt[2]+d_{25} = 3$

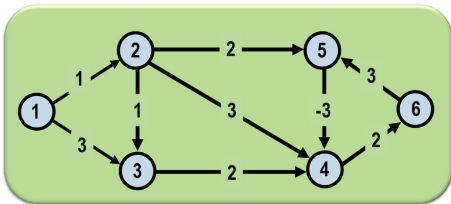
►  $j=6, dt[6]+d_{65} = \infty$

$dt$				
2	3	4	5	6
1	2	4	3	$\infty$

$rot$				
2	3	4	5	6
1	2	2	2	0

Iteração  $k=1$  (continua...)

# Exemplo



$$i=6, \Gamma^-(i)=\{4\};$$

$$\triangleright j=4, dt[4]+d_{46} = 6$$

$$dt$$

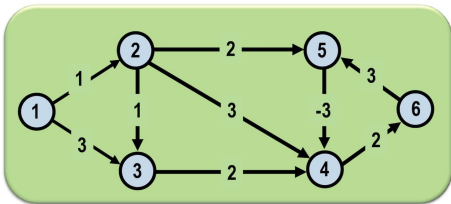
2	3	4	5	6
1	2	4	3	6

$$rot$$

2	3	4	5	6
1	2	2	2	4

Iteração  $k=1$  (final)

# Exemplo



$$i=2, \Gamma^-(i)=\{1\};$$

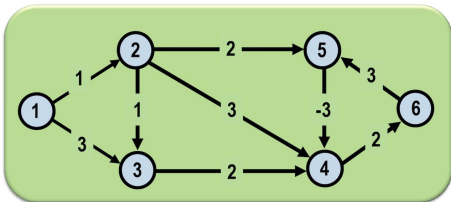
$$\triangleright j=1, dt[1]+d_{12} = 1$$

<i>dt</i>				
2	3	4	5	6
1	2	4	3	6

<i>rot</i>				
2	3	4	5	6
1	2	2	2	4

Iteração  $k=2$  (continua...)

# Exemplo



$$i=3, \Gamma^-(i)=\{1, 2\};$$

$$\triangleright j=1, dt[1]+d_{13} = 3$$

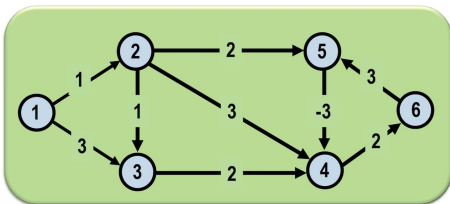
$$\triangleright j=2, dt[2]+d_{23} = 2$$

<i>dt</i>				
2	3	4	5	6
1	2	4	3	6

<i>rot</i>				
2	3	4	5	6
1	2	2	2	4

Iteração  $k=2$  (continua...)

# Exemplo



$i=4, \Gamma^-(i)=\{2, 3, 5\};$

▶  $j=2, dt[2]+d_{24} = 4$

▶  $j=3, dt[3]+d_{34} = 4$

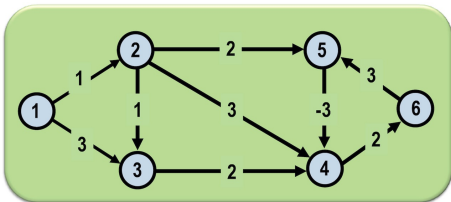
▶  $j=5, dt[5]+d_{54} = 0$

$dt$				
2	3	4	5	6
1	2	0	3	6

$rot$				
2	3	4	5	6
1	2	5	2	4

Iteração  $k=2$  (continua...)

# Exemplo



$$i=5, \Gamma^-(i)=\{2, 6\};$$

$$\triangleright j=2, dt[2]+d_{25} = 3$$

$$\triangleright j=6, dt[6]+d_{65} = 9$$

$$dt$$

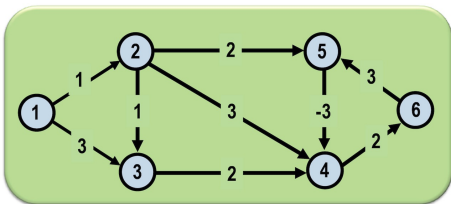
2	3	4	5	6
1	2	0	3	6

$$rot$$

2	3	4	5	6
1	2	5	2	4

Iteração  $k=2$  (continua...)

# Exemplo



$$i=6, \Gamma^-(i)=\{4\};$$

$$\triangleright j=4, dt[4]+d_{46} = 2$$

$$dt$$

2	3	4	5	6
1	2	0	3	2

$$rot$$

2	3	4	5	6
1	2	5	2	4

Iteração  $k=2$  (final)



## Final

Na próxima iteração, em que  $k=3$ , nenhuma alteração é realizada, e com isto, o algoritmo termina!

## Aplicação

Uma das aplicações do algoritmo é em protocolos de roteamento em redes de dados.

O algoritmo é distribuído entre os nós da rede, de maneira que cada nó calcula sua distância em relação aos demais, compartilhando seu resultado para uso pelos outros nós.

Entretanto, o algoritmo possui dificuldades relacionadas a escalabilidade e tolerância a falhas em nós da rede.

Além disso, quaisquer modificações na topologia da rede demoram a serem refletidas pelo algoritmo, dado que a atualização das distâncias é gradual.

# Dúvidas?

