# Capítulo XIV

# Caminhos Mais Curtos de um vértice a todos os vértices

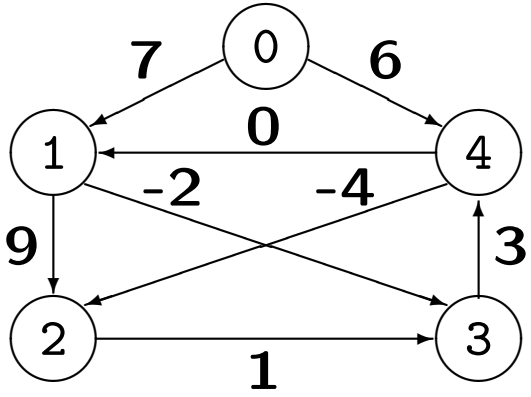
(num grafo orientado e pesado)

—

# Algoritmo de Bellman-Ford

# Problema

Dado um grafo **orientado** e **pesado** e um vértice $o$, como encontrar, para cada vértice $x$ para o qual há caminho a partir de $o$, um **caminho pesado mais curto de $o$ para $x$**?

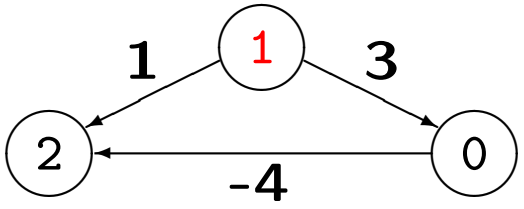



**Caminhos (não pesado e pesado) mais curtos de 0 para 3**

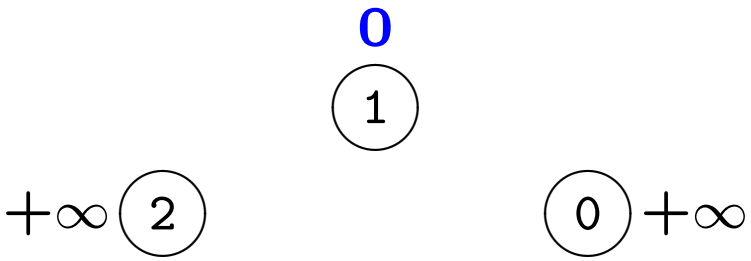
| | | | |
|---|---|---|---|
| Caminho não pesado: | 0, 1, 3 | Compr.: 2 | Compr. pesado: **5** |
| **Caminho pesado:** | 0, 4, 2, 3 | Compr.: 3 | Compr. pesado: **3** |

**Observação:** os pesos dos arcos podem ser negativos.

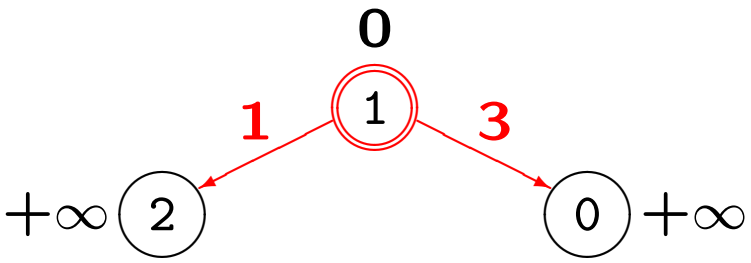
# Dijkstra não Funciona com Pesos Negativos

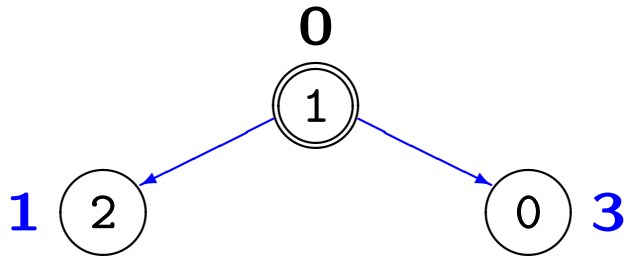



**Inicialização** origem 1

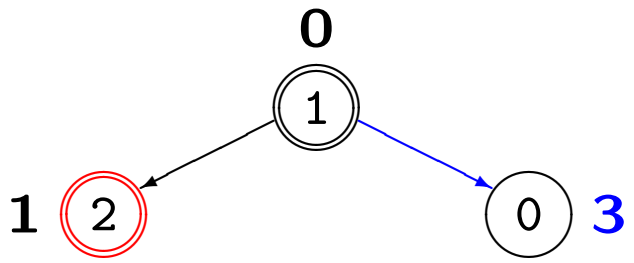



**Seleção de** 1



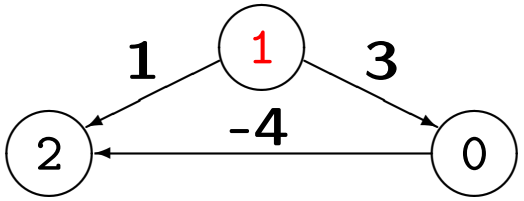

**Seleção de** 2




O comprimento do caminho mais curto de 1 para 2 não é **1**

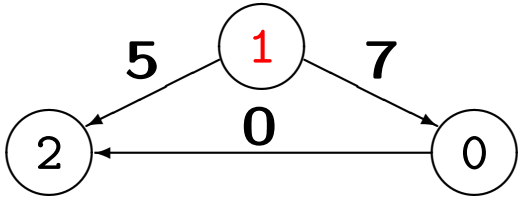
# Ideia que não Funciona

- Alterar os pesos dos arcos: se $m$ for o menor peso de um arco do grafo, adicionar $-m$ a todos os pesos dos arcos.
- Aplicar o algoritmo de Dijkstra.




Caminho mais curto de 1 para 2:

1 0 2

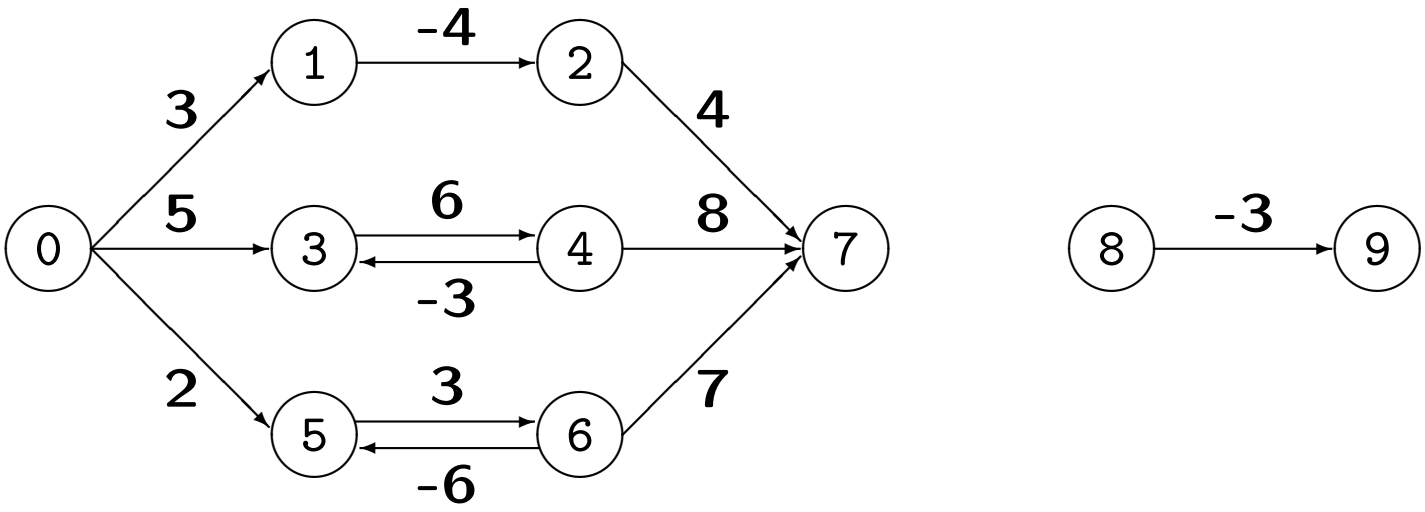



Caminho mais curto de 1 para 2:

1 2

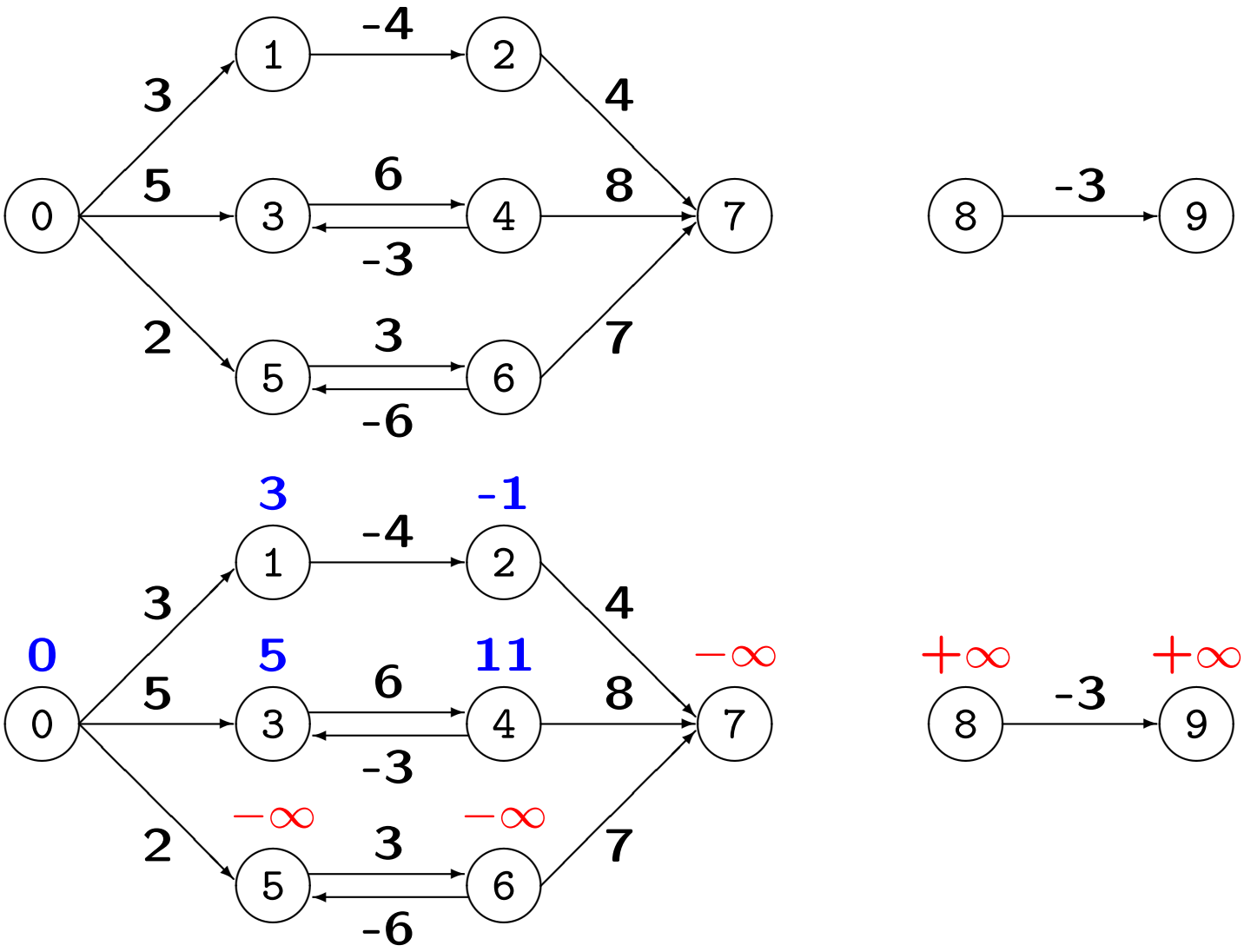
# Ciclos de Peso Negativo

## Comprimentos dos Caminhos Mais Curtos a partir de 0?



# Ciclos de Peso Negativo

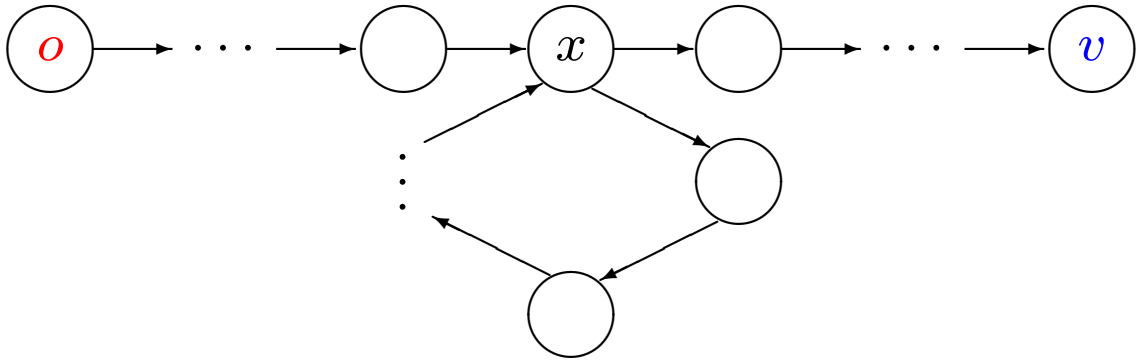
## Comprimentos dos Caminhos Mais Curtos a partir de 0?



# Observação

Se existe algum caminho de $o$ para $v$ e o grafo não tem ciclos de peso negativo acessíveis a partir de $o$, então:

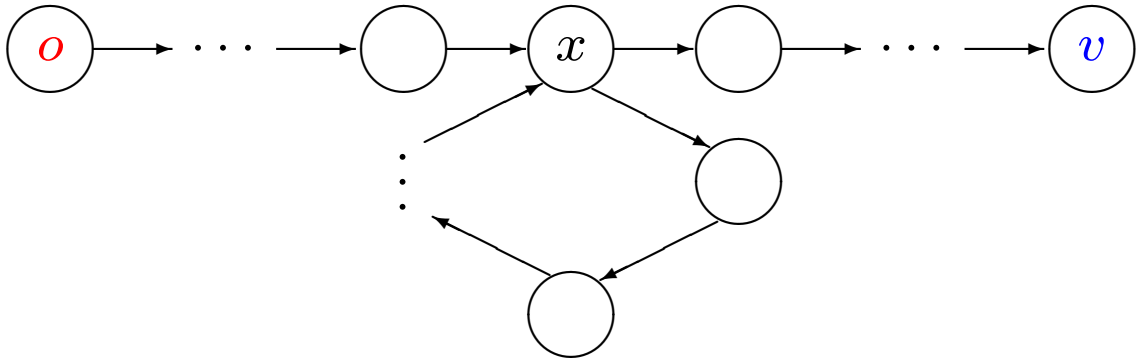
- Há um caminho mais curto de $o$ para $v$ que é **simples**.



# Observação

Se existe algum caminho de $o$ para $v$ e o grafo não tem ciclos de peso negativo acessíveis a partir de $o$, então:

- Há um caminho mais curto de $o$ para $v$ que é **simples**.




- Esse caminho tem, no máximo, $|V|$ vértices e $|V| - 1$ arcos.

# Observação

Se existe algum caminho de $o$ para $v$ e o grafo não tem ciclos de peso negativo acessíveis a partir de $o$, então:

- Há um caminho mais curto de $o$ para $v$ que é **simples**.
- Esse caminho tem, no máximo, $|V|$ vértices e $|V| - 1$ arcos.

# Primeiro Problema a Resolver

Para todo o vértice $v$,

descobrir o comprimento dos caminhos mais curtos de $o$ para $v$

que têm, no máximo, $|V| - 1$ arcos.

# Resolução do Primeiro Problema (1)

Se um caminho mais curto de $o$ para $v$ tem a forma:

$$o,\ w_1,\ w_2,\ \ldots,\ w_n,\ v \quad (\text{com } n \geq 0),$$

então

$$o,\ w_1,\ w_2,\ \ldots,\ w_n$$

é um caminho mais curto de $o$ para $w_n$.

# Resolução do Primeiro Problema (1)

Se um caminho mais curto de $o$ para $v$ tem a forma:

$$o,\ w_1,\ w_2,\ \ldots,\ w_n,\ v \quad (\text{com } n \geq 0),$$

então

$$o,\ w_1,\ w_2,\ \ldots,\ w_n$$

é um caminho mais curto de $o$ para $w_n$.

# Generalização do Primeiro Problema

Para todo o vértice $v$,

descobrir o comprimento dos caminhos mais curtos de $o$ para $v$

que têm, no máximo, $i$ arcos, para $i = 0, 1, \ldots, |V| - 1$:

$\mathcal{L}(v, i)$.

# Resolução do Primeiro Problema (2)

**Comprimento dos caminhos mais curtos de $o$ para $v$ que têm, no máximo, $i$ arcos (para $i \geq 0$):** $\mathcal{L}(v, i)$

- Se $i = 0$

# Resolução do Primeiro Problema (2)

**Comprimento dos caminhos mais curtos de $o$ para $v$ que têm, no máximo, $i$ arcos (para $i \geq 0$):** $\mathcal{L}(v, i)$

- Se $i = 0$ e $v = o$, $\mathcal{L}(v, i) = 0$;
- Se $i = 0$ e $v \neq o$, $\mathcal{L}(v, i) = +\infty$;

# Resolução do Primeiro Problema (2)

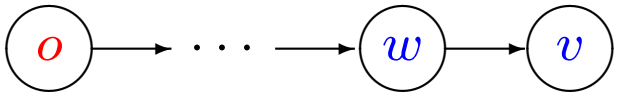
**Comprimento dos caminhos mais curtos de $o$ para $v$ que têm, no máximo, $i$ arcos (para $i \geq 0$):** $\mathcal{L}(v, i)$

- Se $i = 0$ e $v = o$, $\mathcal{L}(v, i) = 0$;
- Se $i = 0$ e $v \neq o$, $\mathcal{L}(v, i) = +\infty$;
- Se $i \geq 1$,
  - **ou** o caminho tem, no máximo, $i - 1$ arcos e o seu comprimento é $\mathcal{L}(v, i - 1)$;

# Resolução do Primeiro Problema (2)

**Comprimento dos caminhos mais curtos de $o$ para $v$ que têm, no máximo, $i$ arcos (para $i \geq 0$):** $\mathcal{L}(v, i)$

- Se $i = 0$ e $v = o$, $\mathcal{L}(v, i) = 0$;
- Se $i = 0$ e $v \neq o$, $\mathcal{L}(v, i) = +\infty$;
- Se $i \geq 1$,
  - **ou** o caminho tem, no máximo, $i - 1$ arcos e o seu comprimento é $\mathcal{L}(v, i - 1)$;
  - **ou** o caminho tem, no máximo, $i$ arcos, o último arco é $(w, v)$, para algum $(w, v) \in A$, e o seu compr. é $\mathcal{L}(w, i - 1) + \text{peso}(w, v)$.



# Resolução do Primeiro Problema (2)

**Comprimento dos caminhos mais curtos de $o$ para $v$ que têm, no máximo, $i$ arcos (para $i \geq 0$):** $\mathcal{L}(v,i)$

- Se $i = 0$ e $v = o$, $\mathcal{L}(v,i) = 0$;
- Se $i = 0$ e $v \neq o$, $\mathcal{L}(v,i) = +\infty$;
- Se $i \geq 1$,
  - **ou** o caminho tem, no máximo, $i-1$ arcos e o seu comprimento é $\mathcal{L}(v,i-1)$;
  - **ou** o caminho tem, no máximo, $i$ arcos, o último arco é $(w,v)$, para algum $(w,v) \in A$, e o seu compr. é $\mathcal{L}(w,i-1) + \text{peso}(w,v)$.

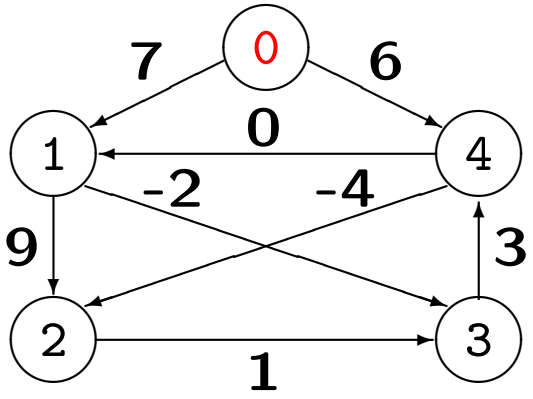
Portanto:

$$\mathcal{L}(v,i) = \min\left(\mathcal{L}(v,i-1), \min_{\{w|(w,v)\in A\}}\left(\mathcal{L}(w,i-1) + \text{peso}(w,v)\right)\right).$$

# Programação Dinâmica de $\mathcal{L}$ $(i = 0)$

$$\mathcal{L}(v,i) = \begin{cases} 0 & i = 0 \text{ e } v = o \\ +\infty & i = 0 \text{ e } v \neq o \\ \min\left(\mathcal{L}(v,i-1), \min\limits_{\{w|(w,v)\in A\}}\left(\mathcal{L}(w,i-1) + \text{peso}(w,v)\right)\right) & i \geq 1 \end{cases}$$

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 0 | | | | |
| 1 | $+\infty$ | | | | |
| 2 | $+\infty$ | | | | |
| 3 | $+\infty$ | | | | |
| 4 | $+\infty$ | | | | |



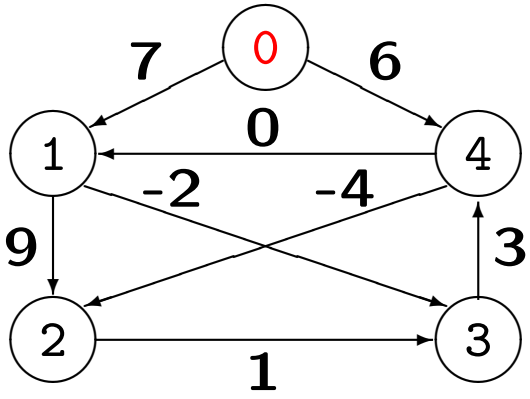

$\mathcal{L}(0,0) = 0$

$\mathcal{L}(1,0) = +\infty$

# Programação Dinâmica de $\mathcal{L}$ $(i = 1)$

$$\mathcal{L}(v,i) = \begin{cases} 0 & i = 0 \text{ e } v = o \\ +\infty & i = 0 \text{ e } v \neq o \\ \min\left(\mathcal{L}(v,i-1), \min\limits_{\{w|(w,v)\in A\}}\left(\mathcal{L}(w,i-1) + \text{peso}(w,v)\right)\right) & i \geq 1 \end{cases}$$

| | **0** | **1** | **2** | **3** | **4** |
|---|---|---|---|---|---|
| 0 | 0 | 0 | | | |
| 1 | $+\infty$ | 7 | | | |
| 2 | $+\infty$ | $+\infty$ | | | |
| 3 | $+\infty$ | $+\infty$ | | | |
| 4 | $+\infty$ | **6** | | | |

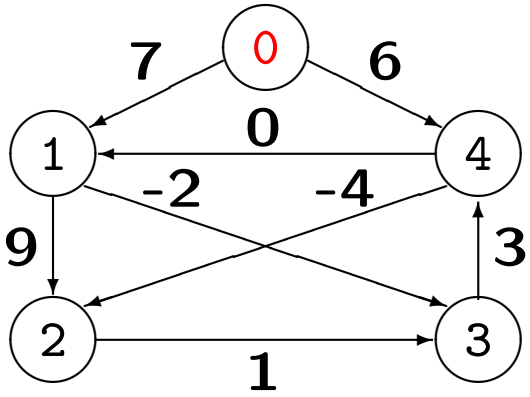



$$\mathcal{L}(4,1) = \min(\ \mathcal{L}(4,0),\ \mathcal{L}(0,0) + \text{peso}(0,4),\ \mathcal{L}(3,0) + \text{peso}(3,4)\ )$$
$$\min(\ +\infty,\ 0 + 6,\ +\infty + 3\ )$$

# Programação Dinâmica de $\mathcal{L}$ $(i = 2)$

$$\mathcal{L}(v,i) = \begin{cases} 0 & i = 0 \text{ e } v = o \\ +\infty & i = 0 \text{ e } v \neq o \\ \min\left(\mathcal{L}(v,i-1), \min\limits_{\{w|(w,v)\in A\}}\left(\mathcal{L}(w,i-1) + \text{peso}(w,v)\right)\right) & i \geq 1 \end{cases}$$

| | **0** | **1** | **2** | **3** | **4** |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | | |
| 1 | $+\infty$ | 7 | **6** | | |
| 2 | $+\infty$ | $+\infty$ | 2 | | |
| 3 | $+\infty$ | $+\infty$ | 5 | | |
| 4 | $+\infty$ | 6 | 6 | | |

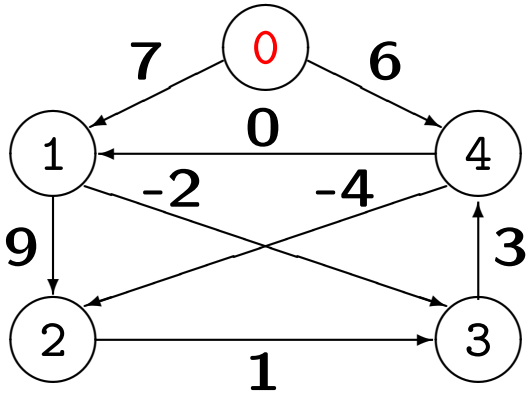



$$\mathcal{L}(1,2) = \min(\ \mathcal{L}(1,1),\ \mathcal{L}(0,1) + \text{peso}(0,1),\ \mathcal{L}(4,1) + \text{peso}(4,1)\ )$$
$$\min(\ 7,\ 0+7,\ 6+0\ )$$

# Programação Dinâmica de $\mathcal{L}$ ($i = 3$)

$$\mathcal{L}(v,i) = \begin{cases} 0 & i = 0 \text{ e } v = o \\ +\infty & i = 0 \text{ e } v \neq o \\ \min\left(\mathcal{L}(v,i-1), \min\limits_{\{w|(w,v)\in A\}}\left(\mathcal{L}(w,i-1) + \text{peso}(w,v)\right)\right) & i \geq 1 \end{cases}$$

| | **0** | **1** | **2** | **3** | **4** |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | |
| 1 | $+\infty$ | 7 | 6 | 6 | |
| 2 | $+\infty$ | $+\infty$ | 2 | 2 | |
| 3 | $+\infty$ | $+\infty$ | 5 | **3** | |
| 4 | $+\infty$ | 6 | 6 | 6 | |

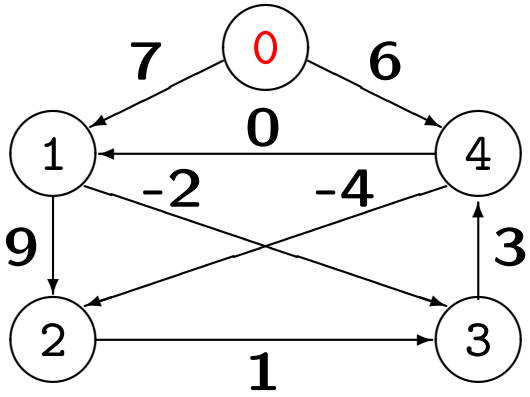



$$\mathcal{L}(3,3) = \min(\ \mathcal{L}(3,2),\ \mathcal{L}(1,2) + \text{peso}(1,3),\ \mathcal{L}(2,2) + \text{peso}(2,3)\ )$$
$$\min(\ 5,\ 6-2,\ 2+1\ )$$

# Programação Dinâmica de $\mathcal{L}$ $(i = 4)$

$$\mathcal{L}(v,i) = \begin{cases} 0 & i = 0 \text{ e } v = o \\ +\infty & i = 0 \text{ e } v \neq o \\ \min\left(\mathcal{L}(v,i-1), \min\limits_{\{w|(w,v)\in A\}}\left(\mathcal{L}(w,i-1) + \text{peso}(w,v)\right)\right) & i \geq 1 \end{cases}$$

| | **0** | **1** | **2** | **3** | **4** |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | $+\infty$ | 7 | 6 | 6 | 6 |
| 2 | $+\infty$ | $+\infty$ | 2 | 2 | 2 |
| 3 | $+\infty$ | $+\infty$ | 5 | 3 | **3** |
| 4 | $+\infty$ | 6 | 6 | 6 | 6 |




$$\mathcal{L}(3,4) = \min(\ \mathcal{L}(3,3),\ \mathcal{L}(1,3) + \text{peso}(1,3),\ \mathcal{L}(2,3) + \text{peso}(2,3)\ )$$
$$\min(\ 3,\ 6-2,\ 2+1\ )$$

# Programação Dinâmica de $\mathcal{L}$ (versão 1)

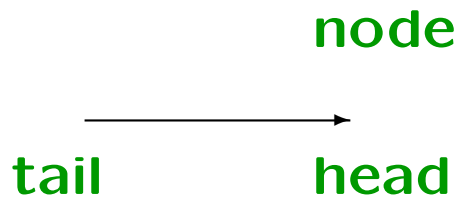
```
Pair<L[][], Node[]> L-DP1( Digraph<L> graph, Node origin ) {
    L[][] length = new L[ graph.numNodes() ][ graph.numNodes() ];
    Node[] via = new Node[ graph.numNodes() ];

    for every Node v in graph.nodes()
        length[v][0] = +∞;
    length[origin][0] = 0;
    via[origin] = origin;

    for ( int i = 1;   i < graph.numNodes();   i++ )
        compLengths1(graph, length, via, i);

    return new PairClass<>(length, via);
}
```

```
void compLengths1( Digraph<L> graph, L[][] len, Node[] via, int col ) {
    for every Node node in graph.nodes() {
        len[node][col] = len[node][col − 1];
        for every Edge<L> e in graph.inIncidentEdges(node) {
            Node tail = e.firstNode();
            if ( len[tail][col − 1] < +∞ ) {
                L newLen = len[tail][col − 1] + e.label();
                if ( newLen < len[node][col] ) {
                    len[node][col] = newLen;
                    via[node] = tail;
                }
            }
        }
    }
}
```



# Alteração à Implementação de $\mathcal{L}$

Em cada grande passo (cada execução de compLengths),

em vez de se percorrerem todos os arcos do grafo por grupos,

onde cada grupo tem o mesmo vértice destino,

percorrem-se todos os arcos do grafo por uma ordem qualquer.

No fim de cada grande passo, o conteúdo da matriz

**não depende** da ordem pela qual os arcos são percorridos.

# Programação Dinâmica de $\mathcal{L}$ (versão 2)

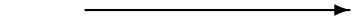
```
Pair<L[][], Node[]> L-DP2( Digraph<L> graph, Node origin ) {
    L[][] length = new L[ graph.numNodes() ][ graph.numNodes() ];
    Node[] via = new Node[ graph.numNodes() ];

    for every Node v in graph.nodes()
        length[v][0] = +∞;
    length[origin][0] = 0;
    via[origin] = origin;

    for ( int i = 1;   i < graph.numNodes();   i++ )
        compLengths2(graph, length, via, i);

    return new PairClass<>(length, via);
}
```

```
void compLengths2( Digraph<L> graph, L[][] len, Node[] via, int col ) {
    for every Node node in graph.nodes()
        len[node][col] = len[node][col − 1];
    for every Edge<L> e in graph.edges() {
        Node tail = e.firstNode();
        Node head = e.secondNode();
        if ( len[tail][col − 1] < +∞ ) {
            L newLen = len[tail][col − 1] + e.label();
            if ( newLen < len[head][col] ) {
                len[head][col] = newLen;
                via[head] = tail;
            }
        }
    }
}
```



# Algoritmo de Bellman-Ford [1958,1962]

Em cada grande passo (cada execução de compLengths),

percorrem-se todos os arcos do grafo por uma ordem qualquer.

Em vez de se guardarem os valores da função $\mathcal{L}$

numa matriz de $|V| \times |V|$

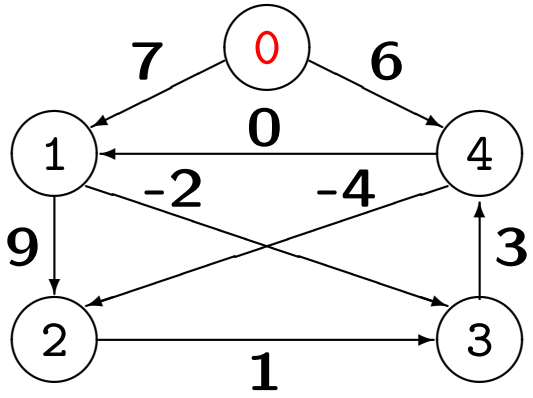
(vértices por número máximo de arcos do caminho + 1),

utiliza-se um vetor de $|V|$ posições (vértices).

No fim de cada grande passo, o conteúdo do vetor

**depende** da ordem pela qual os arcos são percorridos.

# Simulação dos Dois Algoritmos — Matriz




**Ordem Usada**

| | |
|---|---|
| (0,1) | (2,3) |
| (0,4) | (3,4) |
| (1,2) | (4,1) |
| (1,3) | (4,2) |

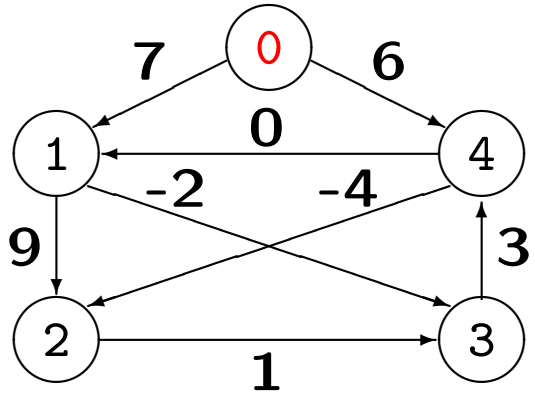
Matriz ($5 \times 5$)

| | **0** | **1** | **2** | **3** | **4** |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | $+\infty$ | 7 | 6 | 6 | 6 |
| 2 | $+\infty$ | $+\infty$ | 2 | 2 | 2 |
| 3 | $+\infty$ | $+\infty$ | 5 | 3 | 3 |
| 4 | $+\infty$ | 6 | 6 | 6 | 6 |

**1ª Iteração dos Arcos** (Coluna 1)

$$
\begin{aligned}
M[1][1] &= \min(+\infty, 0+7) = 7 \\
M[4][1] &= \min(+\infty, 0+6) = 6 \\
M[2][1] &= \min(+\infty, +\infty) = +\infty \\
M[3][1] &= \min(+\infty, +\infty) = +\infty \\
M[3][1] &= \min(+\infty, +\infty) = +\infty \\
M[4][1] &= \min(+\infty, +\infty) = +\infty \\
M[1][1] &= \min(+\infty, +\infty) = +\infty \\
M[2][1] &= \min(+\infty, +\infty) = +\infty
\end{aligned}
$$

# Simulação dos Dois Algoritmos — Vetor




**Ordem Usada**

| | |
|---|---|
| (0,1) | (2,3) |
| (0,4) | (3,4) |
| (1,2) | (4,1) |
| (1,3) | (4,2) |

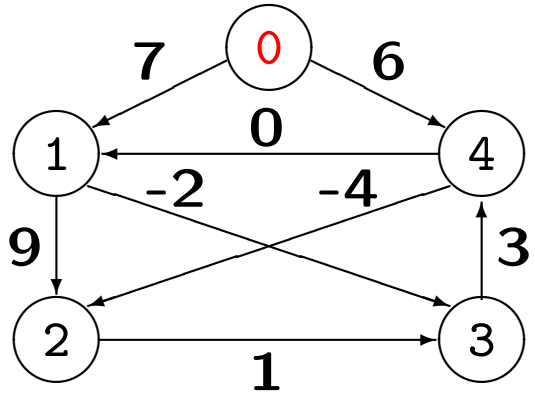
Vetor (compr. 5)

| | **0** | **1** | **2** | **3** | **4** |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | $+\infty$ | 6 | 6 | 6 | 6 |
| 2 | $+\infty$ | 2 | 2 | 2 | 2 |
| 3 | $+\infty$ | 5 | 3 | 3 | 3 |
| 4 | $+\infty$ | 6 | 6 | 6 | 6 |

**1ª Iteração dos Arcos**

$$
\begin{aligned}
V[1] &= \min(+\infty,\ 0+7) = 7\\
V[4] &= \min(+\infty,\ 0+6) = 6\\
V[2] &= \min(+\infty,\ 7+9) = 16\\
V[3] &= \min(+\infty,\ 7-2) = 5\\
V[3] &= \min(\ 5\ ,\ 16+1) = 5\\
V[4] &= \min(\ 6\ ,\ 5+3) = 6\\
V[1] &= \min(\ 7\ ,\ 6+0) = 6\\
V[2] &= \min(\ 16\ ,\ 6-4) = 2
\end{aligned}
$$

# Simulação dos Dois Algoritmos




**Ordem Usada**

| | |
|---|---|
| (0,1) | (2,3) |
| (0,4) | (3,4) |
| (1,2) | (4,1) |
| (1,3) | (4,2) |

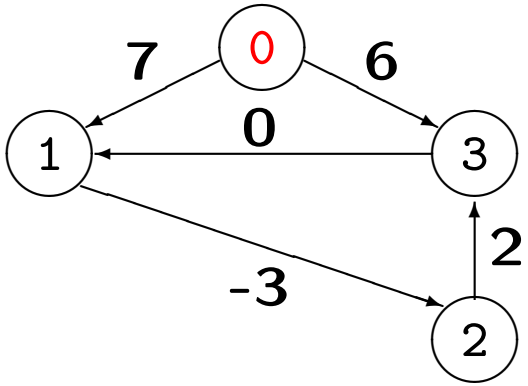
Matriz ($5 \times 5$)

| | **0** | **1** | **2** | **3** | **4** |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | $+\infty$ | 7 | 6 | 6 | 6 |
| 2 | $+\infty$ | $+\infty$ | 2 | 2 | 2 |
| 3 | $+\infty$ | $+\infty$ | 5 | 3 | 3 |
| 4 | $+\infty$ | 6 | 6 | 6 | 6 |

Vetor (compr. 5)

| | **0** | **1** | **2** | **3** | **4** |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | $+\infty$ | 6 | 6 | 6 | 6 |
| 2 | $+\infty$ | 2 | 2 | 2 | 2 |
| 3 | $+\infty$ | 5 | 3 | 3 | 3 |
| 4 | $+\infty$ | 6 | 6 | 6 | 6 |

# Simulação com Ciclos de Peso Negativo




**Ordem Usada**

| | |
|---|---|
| (0,1) | (2,3) |
| (0,3) | (3,1) |
| (1,2) | |

Matriz ($4 \times 4$)

| | **0** | **1** | **2** | **3** | **4** | **5** |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | $+\infty$ | 7 | 6 | 6 | 6 | 5 |
| 2 | $+\infty$ | $+\infty$ | 4 | 3 | 3 | 3 |
| 3 | $+\infty$ | 6 | 6 | 6 | 5 | 5 |

Vetor (compr. 4)

| | **0** | **1** | **2** | **3** | **4** | **5** |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | $+\infty$ | 6 | 5 | 4 | 3 | 2 |
| 2 | $+\infty$ | 4 | 3 | 2 | 1 | 0 |
| 3 | $+\infty$ | 6 | 5 | 4 | 3 | 2 |

# Caminhos Mais Curtos (1)

*(Single-source Shortest Paths)*

```
Pair<L[], Node[]> bellmanFord( Digraph<L> graph, Node origin )
    throws NegativeWeightCycleException {

    L[] length = new L[ graph.numNodes() ];
    Node[] via = new Node[ graph.numNodes() ];

    for every Node v in graph.nodes()
        length[v] = +∞;
    length[origin] = 0;
    via[origin] = origin;
```

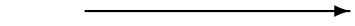
# Caminhos Mais Curtos (2)

```
    boolean changes = false;
    for ( int i = 1;   i < graph.numNodes();   i++ ) {
        changes = updateLengths(graph, length, via);
        if ( !changes )
            break;
    }

    // Negative-weight cycles detection.
    if ( changes   &&   updateLengths(graph, length, via) )
        throw new NegativeWeightCycleException();

    return new PairClass<>(length, via);
}
```

```
boolean updateLengths( Digraph<L> graph, L[] len, Node[] via ) {
    boolean changes = false;
    for every Edge<L> e in graph.edges() {
        Node tail = e.firstNode();
        Node head = e.secondNode();
        if ( len[tail] < +∞ ) {
            L newLen = len[tail] + e.label();
            if ( newLen < len[head] ) {
                len[head] = newLen;
                via[head] = tail;
                changes = true;
            }
        }
    }
    return changes;
}
```



tail head

# Complexidade Espacial de **bellmanFord**

**Caminhos Mais Curtos**

num grafo orientado e pesado

sem ciclos de peso negativo acessíveis da origem

ou

**Deteção de Ciclos de Peso Negativo Acessíveis da Origem**

num grafo orientado e pesado

| | |
|---|---|
| Vetor length | $\Theta(\|V\|)$ |
| Vetor via | $\Theta(\|V\|)$ |
| **TOTAL** | $\Theta(\|V\|)$ |

# Complexidade Temporal de **bellmanFord**

## Grafo em matriz de adjacências

| | |
|---|---|
| Criação de length e via | $\Theta(1)$ |
| Inicialização de length | $\Theta(\|V\|)$ |
| $\leq \|V\|$ execuções de **updateLengths** | $O(\|V\|^3)$ |
| • cada execução: $\Theta(\|V\|^2)$ | |
| **TOTAL** | $O(\|V\|^3)$ |

# Complexidade Temporal de **bellmanFord**

## Grafo em vetor de listas de incidências (antecessores ou sucessores)

| | |
|---|---|
| Criação de length e via | $\Theta(1)$ |
| Inicialização de length | $\Theta(\|V\|)$ |
| $\leq \|V\|$ execuções de **updateLengths** | $O(\|V\|^2 + \|V\| \times \|A\|)$ |
| • cada execução: $\Theta(\|V\| + \|A\|)$ | |
| **TOTAL** | $O(\|V\|^2 + \|V\| \times \|A\|)$ |

# Complexidade Temporal de **bellmanFord**

## Grafo em vetor ou lista de arcos (e $|V|$)

| | |
|---|---|
| Criação de length e via | $\Theta(1)$ |
| Inicialização de length | $\Theta(\|V\|)$ |
| $\leq \|V\|$ execuções de **updateLengths** | $O(\|V\| \times \|A\|)$ |
| • cada execução: $\Theta(\|A\|)$ | |
| **TOTAL** | $O(\|V\| \times \|A\|)$ |