# Capítulo X

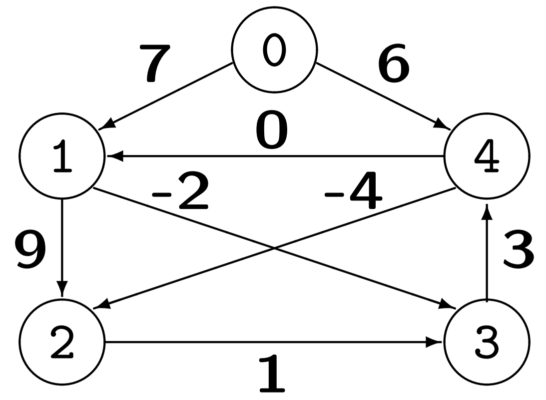
# Caminhos Mais Curtos de um vértice a todos os vértices

—

# 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 $x$**?



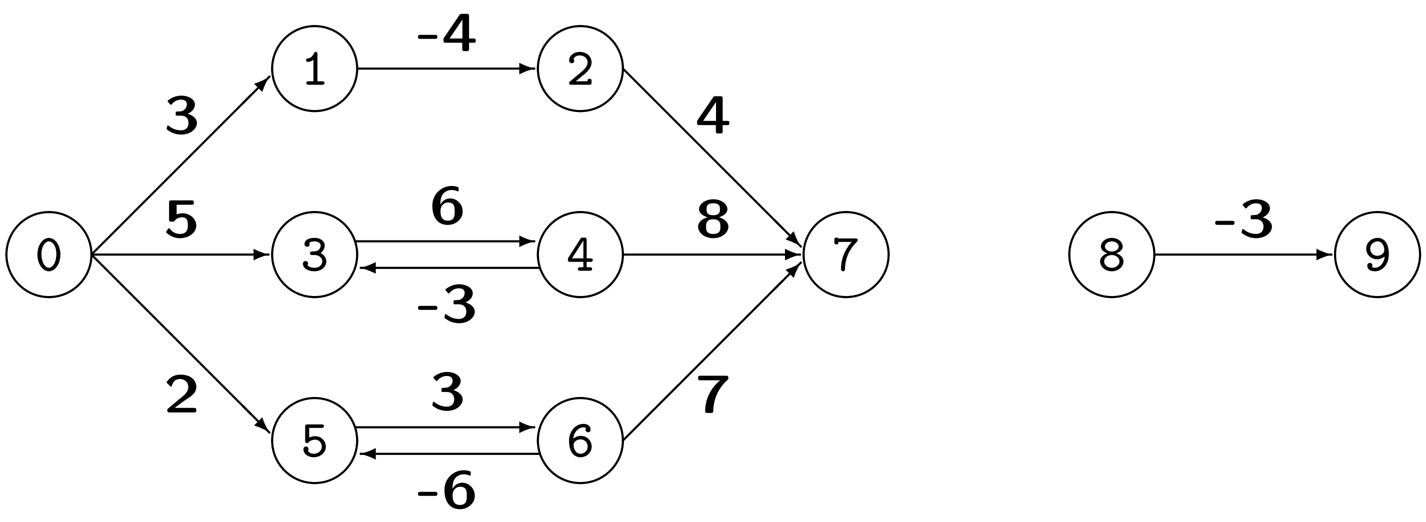

**Caminho mais curto de 0 para 3**

| | | | |
|---|---|---|---|
| **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.

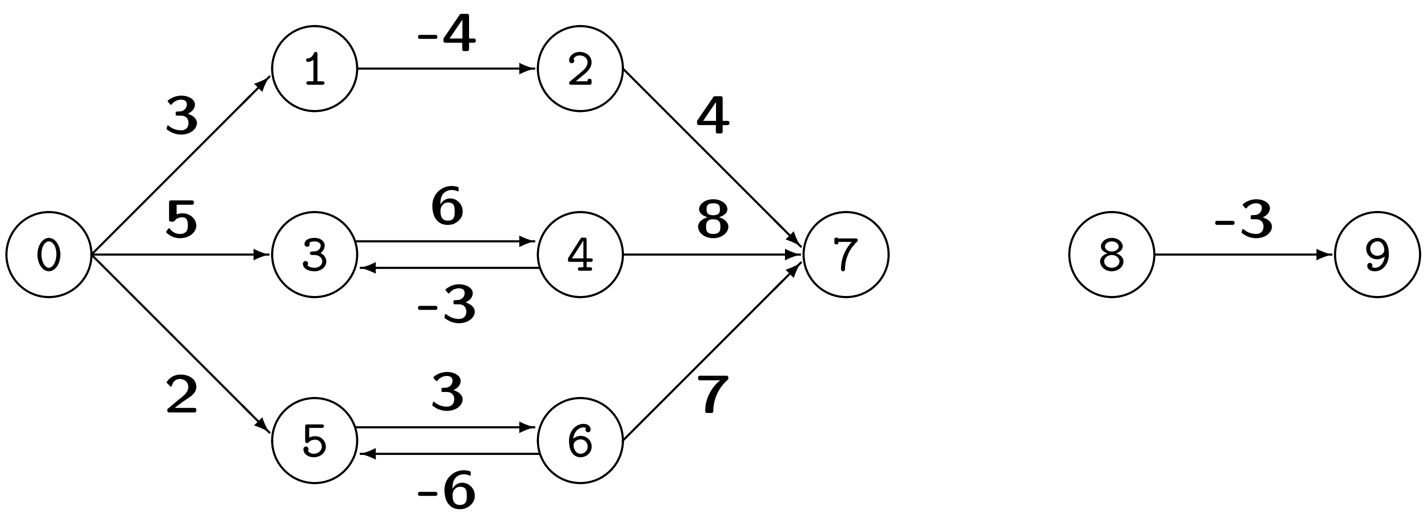
# Ciclos de Peso Negativo

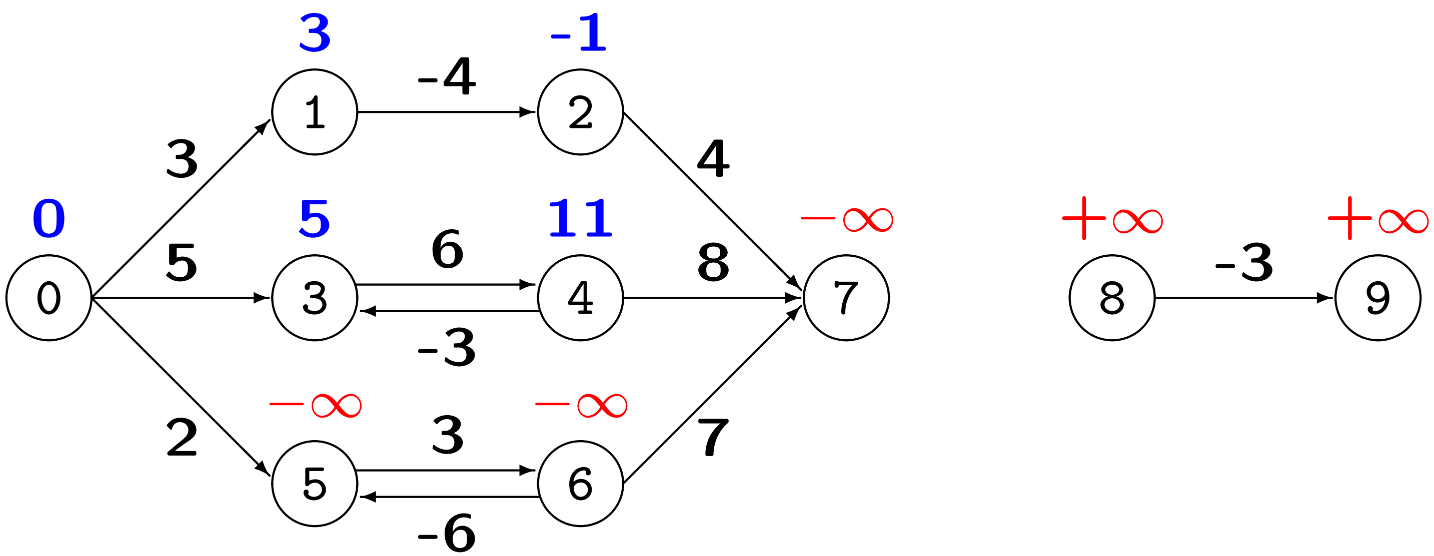
## Comprimentos dos Caminhos Mais Curtos a partir de 0?



# Ciclos de Peso Negativo

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



# Observações

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**; e
- esse caminho tem, no máximo, ...... vértices e ...... arcos.

# Observações

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**; e
- esse caminho tem, no máximo, $|V|$ vértices e $|V| - 1$ arcos.

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

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

então

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

é um caminho ......

# Observações

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**; e
- esse caminho tem, no máximo, $|V|$ vértices e $|V| - 1$ arcos.

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 $w_n$.

# 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.

# 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.

# Primeiro Problema que Será Resolvido

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

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

- Se $i = 0$ e $v = o$, $\quad \mathcal{L}(v,i) = 0$;
- Se $i = 0$ e $v \neq o$, $\quad \mathcal{L}(v,i) = +\infty$;
- Se $i > 0$,
  - **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 $\mathcal{L}(w,i-1) + \text{peso}(w,v)$.

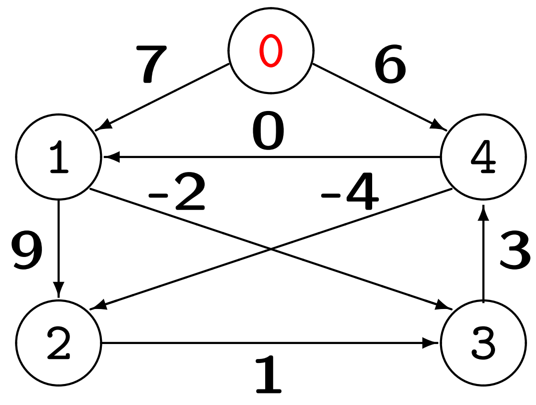
Portanto:

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

# 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 | $+\infty$ | | | | |

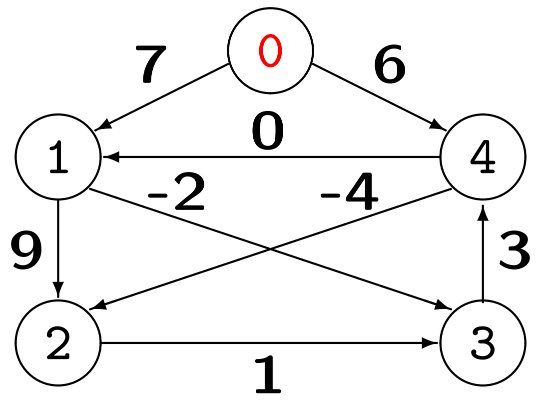



$$\mathcal{L}(0,0) = 0$$
$$\mathcal{L}(1,0) = +\infty$$

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

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

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

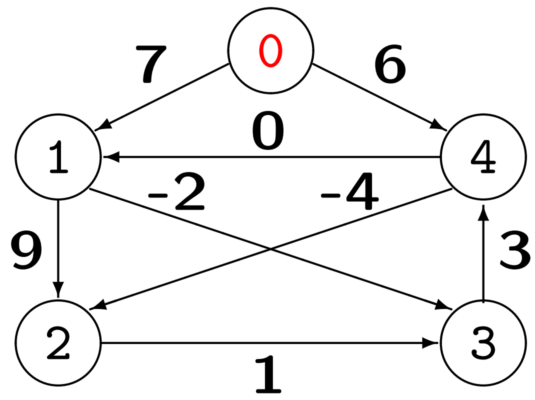



$$\begin{aligned} \mathcal{L}(4,1) &= \min( \ \mathcal{L}(4,0), \ \mathcal{L}(0,0) + \text{peso}(0,4), \ \mathcal{L}(3,0) + \text{peso}(3,4) \ ) \\ & \quad \min( \ +\infty, \ 0 + 6, \ +\infty + 3 \ ) \end{aligned}$$

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

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

| | **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 | | |



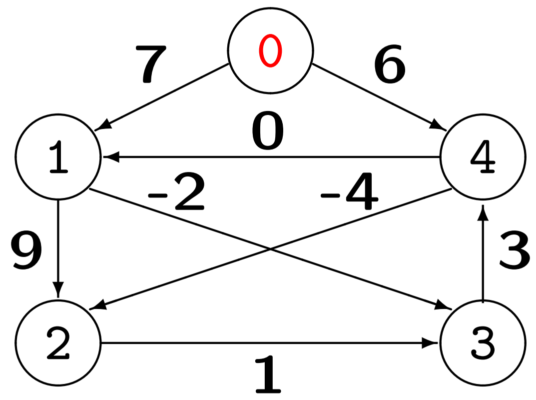

$$\mathcal{L}(1,2) = \min(\ \mathcal{L}(1,1),\ \mathcal{L}(0,1) + \text{peso}(0,1),\ \mathcal{L}(4,1) + \text{peso}(4,1)\ )$$

$$\min(\ 7,\ 0+7,\ 6+0\ )$$

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

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

| | **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 | |

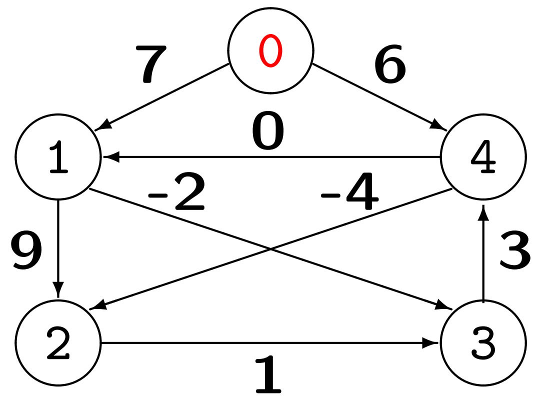



$$\begin{aligned} \mathcal{L}(3,3) \;=\; & \min(\;\; \mathcal{L}(3,2),\;\; \mathcal{L}(1,2) + \text{peso}(1,3),\;\; \mathcal{L}(2,2) + \text{peso}(2,3) \;\;) \\ & \min(\;\; 5,\;\; 6-2,\;\; 2+1 \;\;) \end{aligned}$$

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

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

| | **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 |




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

# Programação Dinâmica de $\mathcal{L}$

```
Pair<L[], Node[]> 𝓛-DP( 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++ )
        compLengths(graph, length, via, i);

    return new PairClass<L[], Node[]>(length, via);
}
```

```
void compLengths( Digraph<L> graph, L[] length, Node[] via, int col )
{
    for every Node node in graph.nodes()
    {
        length[node][col] = length[node][col − 1];
        for every Edge<L> e in graph.inIncidentEdges(node)
        {
            Node tail = e.endNodes()[0];
            if ( length[tail][col − 1] < +∞ )
            {
                L newLength = length[tail][col − 1] + e.label();
                if ( newLength < length[node][col] )
                {
                    length[node][col] = newLength;     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

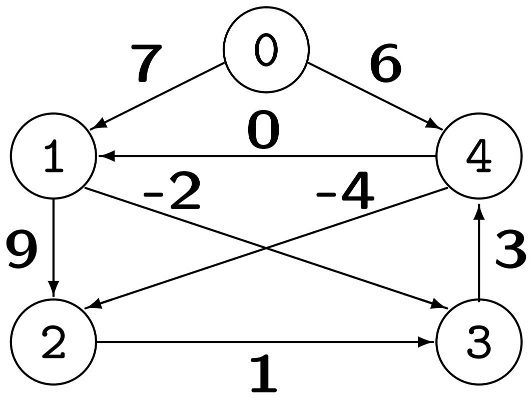
**não depende** da ordem pela qual os arcos são percorridos.

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

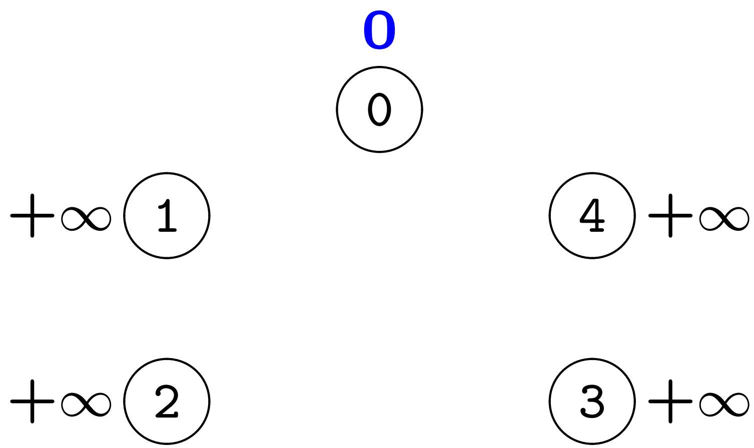
```
Pair<L[], Node[]> ℒ-DP-v2( Digraph<L> graph, Node origin )
{
    L[][] length = new L[ graph.numNodes() ][ graph.numNodes() ];
    Node[] via = new Node[ graph.numNodes() ];

    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++ )
        compLengths-v2(graph, length, via, i);

    return new PairClass<L[], Node[]>(length, via);
}
```

```
void compLengths-v2( Digraph<L> graph, L[] length, Node[] via,
    int col )
{
    for every Node v in graph.nodes()
        length[v][col] = length[v][col − 1];
    for every Edge<L> e in graph.edges()
    {
        Node[] endPoints = e.endNodes();
        Node tail = endPoints[0],     head = endPoints[1];
        if ( length[tail][col − 1] < +∞ )
        {
            L newLength = length[tail][col − 1] + e.label();
            if ( newLength < length[head][col] )
            {
                length[head][col] = newLength;     via[head] = tail;
            }
        }
    }
}
```

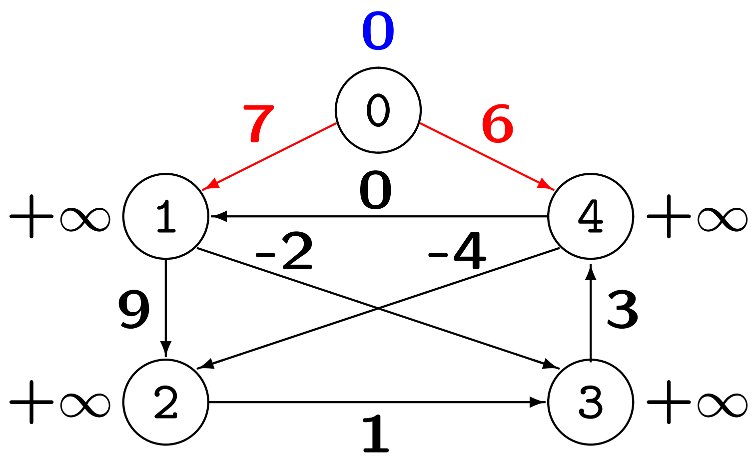
# Simulação do Algoritmo após 1 Passo ($i = 1$)

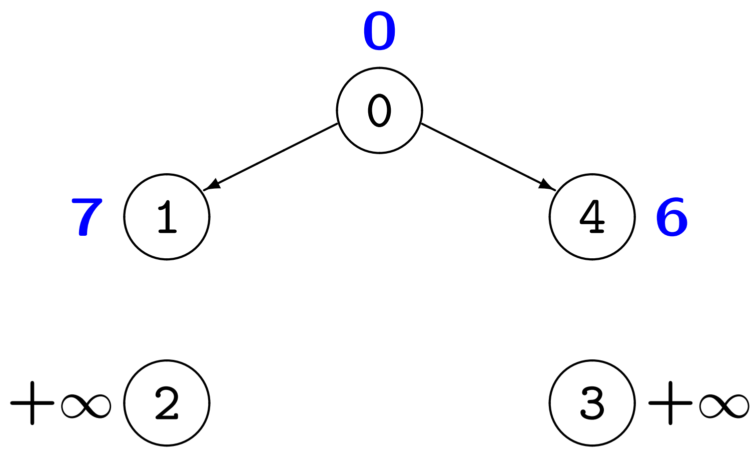



**ZERO ARCOS** origem 0

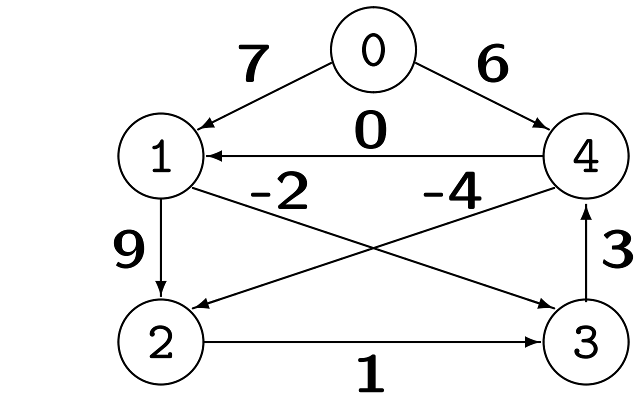



**Percorrer todos os Arcos**

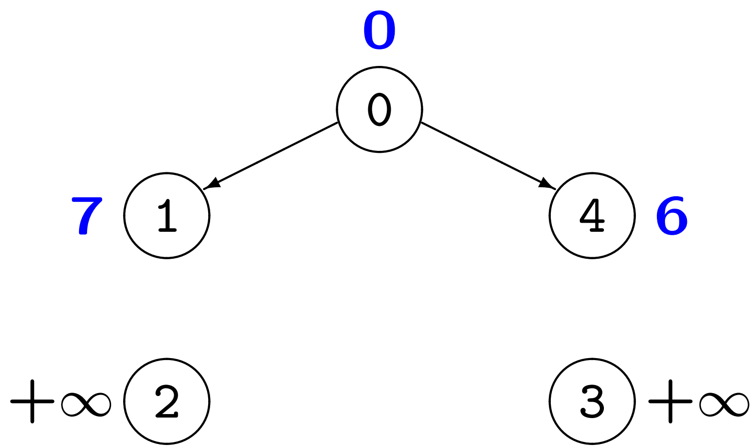


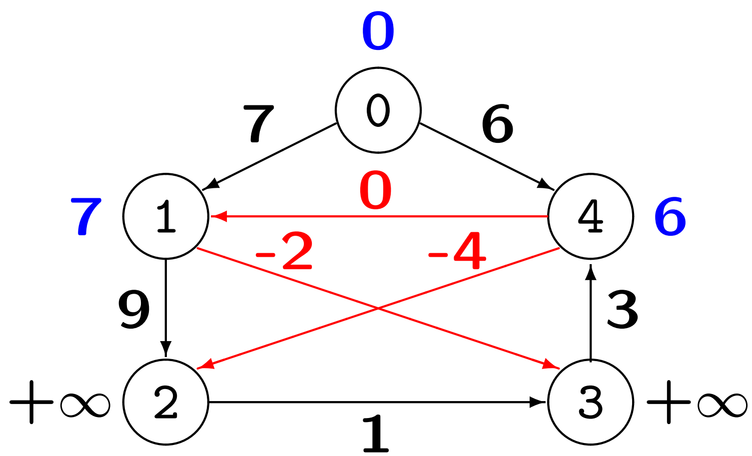
# Simulação do Algoritmo após 2 Passos ($i = 2$)

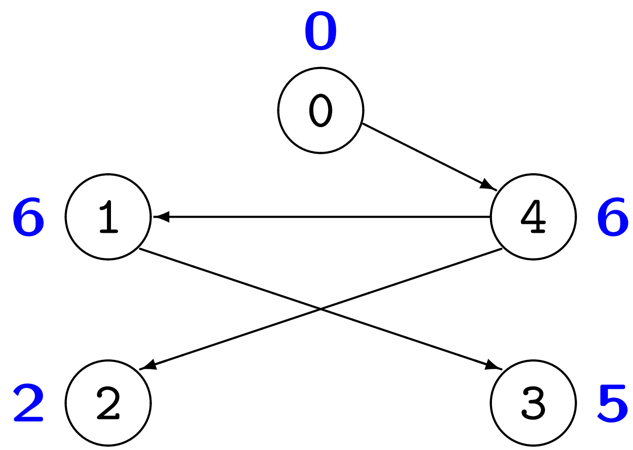



**Situação Corrente**

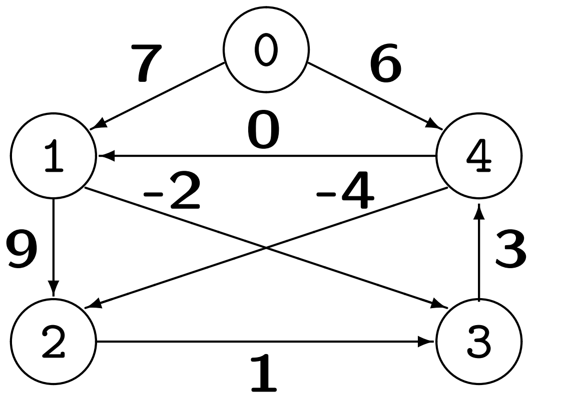



**Percorrer os Arcos**

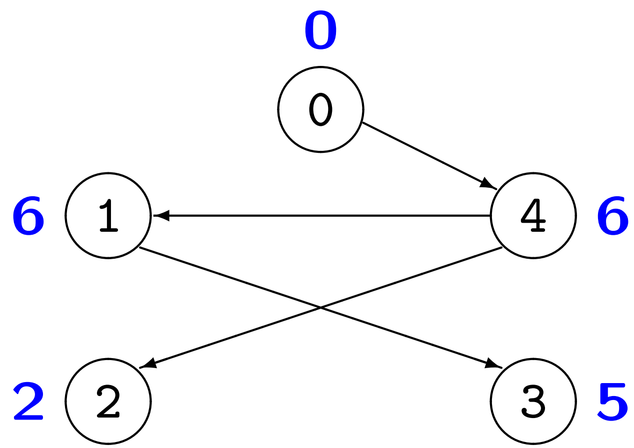


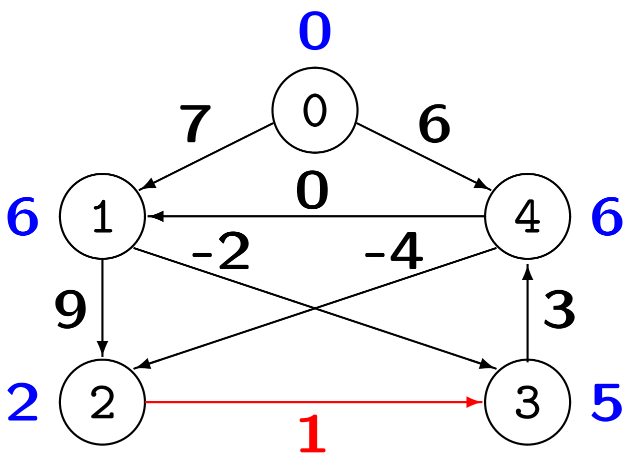
# Simulação do Algoritmo após 3 Passos ($i = 3$)

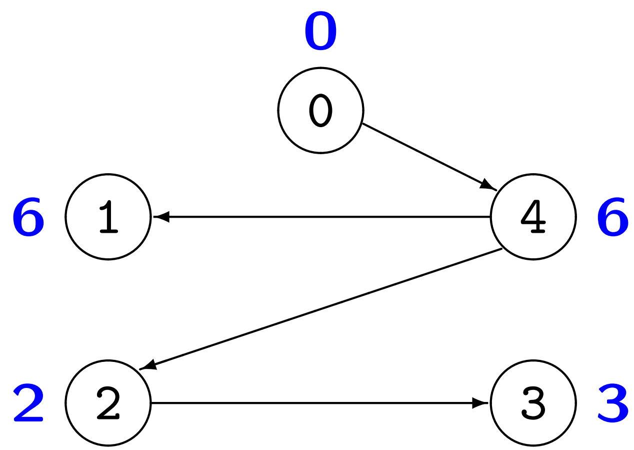



**Situação Corrente**

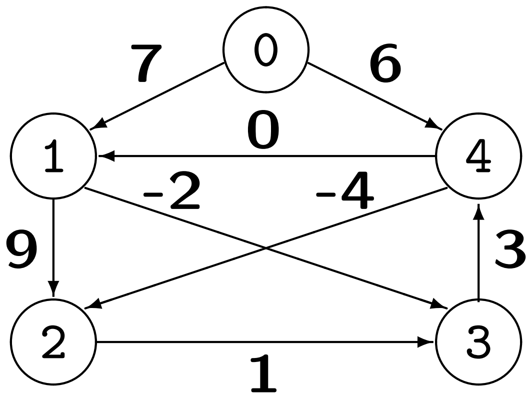



**Percorrer os Arcos**

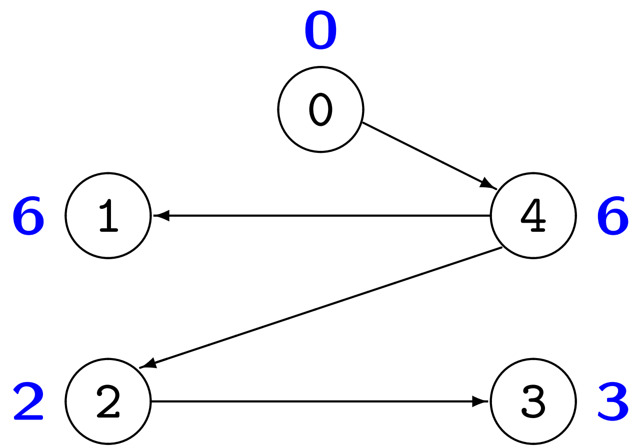


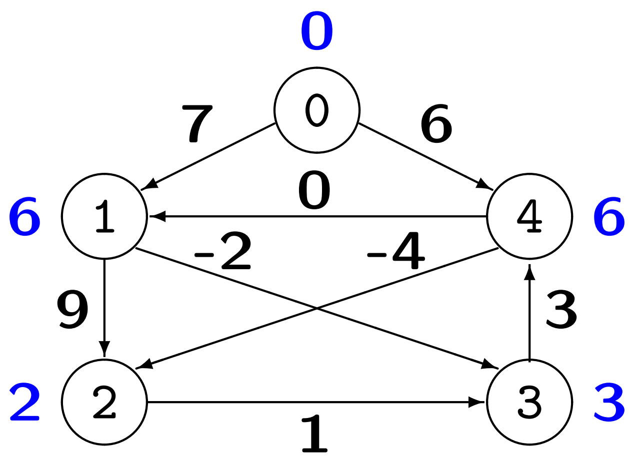
# Simulação do Algoritmo após 4 Passos ($i = 4$)

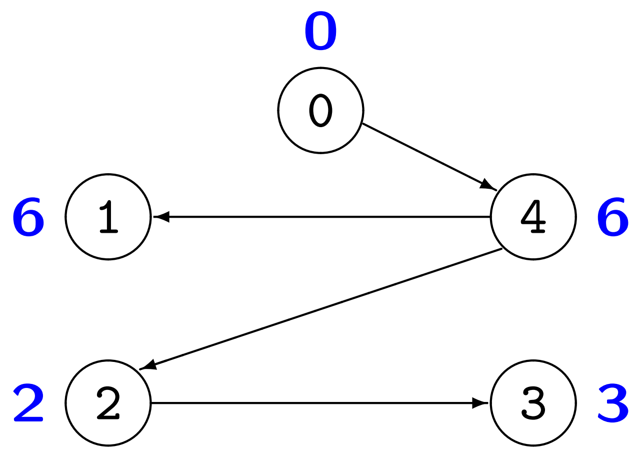



**Situação Corrente**




**Percorrer os Arcos**



# 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 $|V| \times |V|$

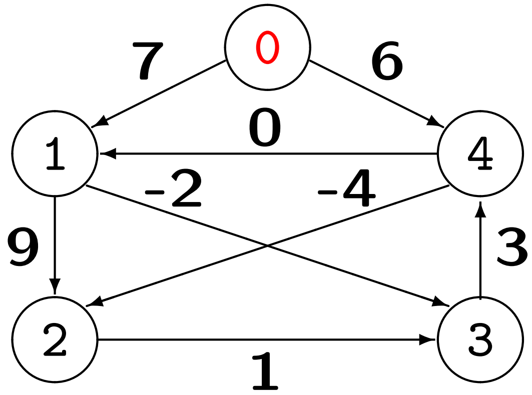
(vértices por número máximo de arcos do caminho $+ 1$),

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




**Ordem Unidimensional**

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

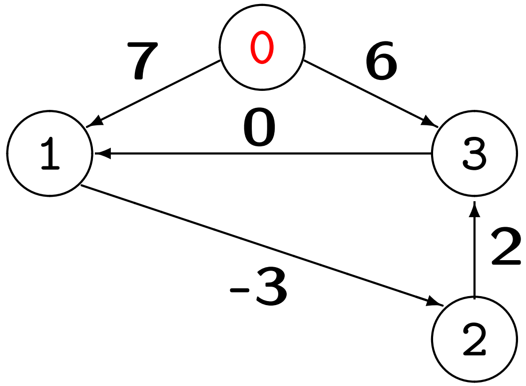
Matriz ($5 \times 5$)

| | **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 Unidimensional**

(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<L[], Node[]>(length, via);
}
```

```
boolean updateLengths( Digraph<L> graph, L[] length, Node[] via )
{
   boolean changes = false;
   for every Edge<L> e in graph.edges()
   {
      Node[] endPoints = e.endNodes();
      Node tail = endPoints[0],      head = endPoints[1];
      if ( length[tail] < +∞ )
      {
         L newLength = length[tail] + e.label();
         if ( newLength < length[head] )
         {
            length[head] = newLength;
            via[head] = tail;
            changes = true;
         }
      }
   }
   return changes;
}
```

# Complexidade do Algoritmo de Bellman-Ford

| **Implementação do Grafo** $(V, A)$ | **Caminhos Mais Curtos** (grafo orientado e pesado, sem ciclos de peso negativo) ou **Deteção de Ciclos de Peso Negativo** (grafo orientado e pesado) |
|---|---|
| Matriz de adjacências | $O(\lvert V\rvert^{3})$ |
| Vetor de Listas de incidências | $O(\lvert V\rvert \times (\lvert V\rvert + \lvert A\rvert))$ |
| Vetor ou Lista de arcos | $O(\lvert V\rvert \times \lvert A\rvert)$ |