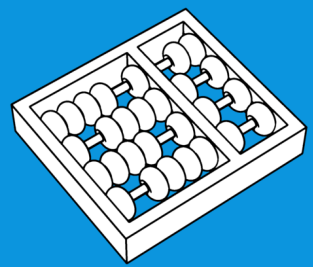
Caminhos mínimos

MC558 - Projeto e Análise de Algoritmos II

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

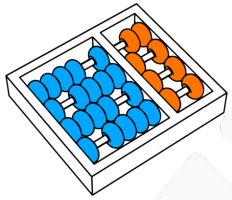
UNICAMP



“Sometimes the most productive thing you can do is relax.”

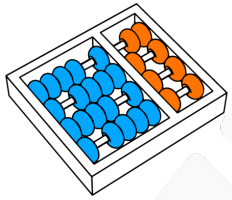
Mark Black

Aula anterior



Representando caminhos mínimos

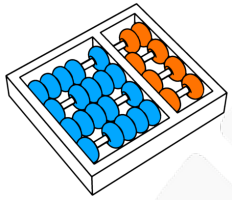
A saída é similar à da busca em largura a partir de **s**:



Representando caminhos mínimos

A saída é similar à da busca em largura a partir de **s**:

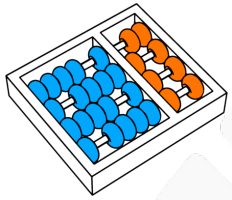
- Para cada **v** $\in$ **V[G]**, associamos um **PREDECESSOR** π[**v**].



Representando caminhos mínimos

A saída é similar à da busca em largura a partir de **s**:

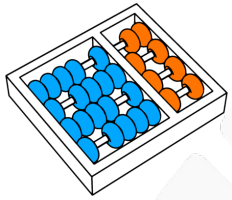
- Para cada **v** $\in$ **V[G]**, associamos um **PREDECESSOR** π[**v**].
- O vetor π induz uma **ÁRVORE DE CAMINHOS MÍNIMOS** com raiz em **s**.



Representando caminhos mínimos

A saída é similar à da busca em largura a partir de **s**:

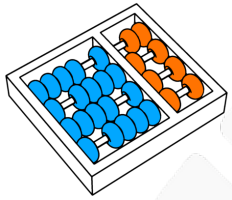
- Para cada **v** $\in$ **V[G]**, associamos um **PREDECESSOR** π[**v**].
- O vetor π induz uma **ÁRVORE DE CAMINHOS MÍNIMOS** com raiz em **s**.
- Um caminho de **s** a **v** na árvore é um caminho mínimo de **s** a **v** no grafo G.



Inicialização

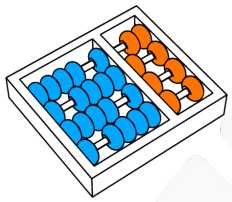
Algoritmo 1: INITIALIZE-SINGLE-SOURCE(G, s)

```
1 para cada v ∈ V[G]
2     d[v] ← ∞
3     π[v] ← NIL
4 d[s] ← 0
```



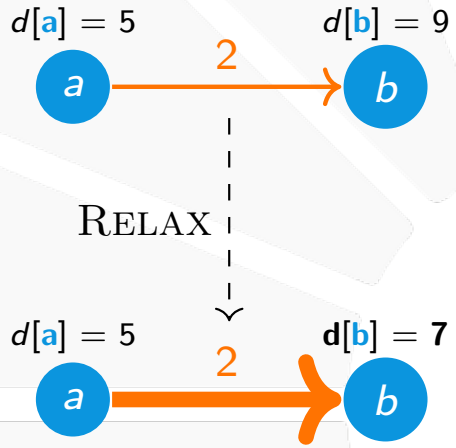
Relaxação

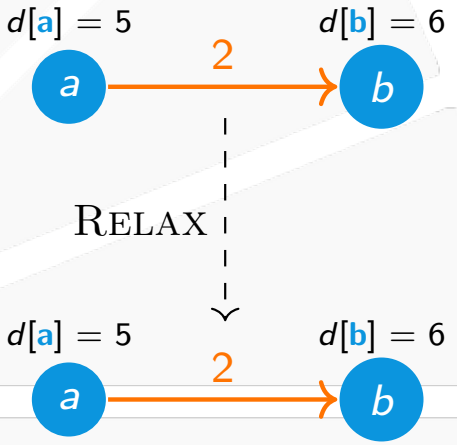
Tenta melhorar a estimativa $d[\mathbf{v}]$ examinando **(u,v)**.

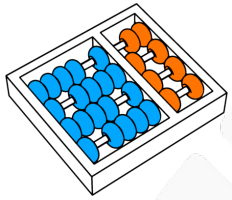


Relaxação

Tenta melhorar a estimativa $d[\mathbf{v}]$ examinando **(u,v)**.

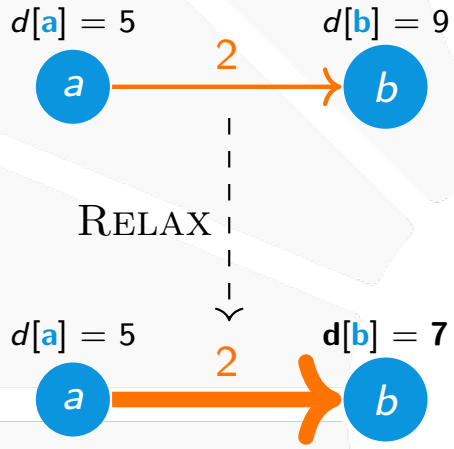


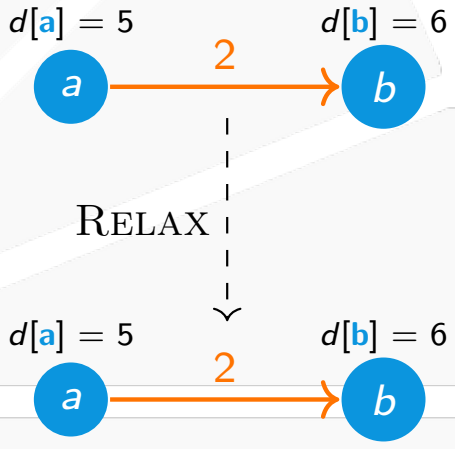




Relaxação

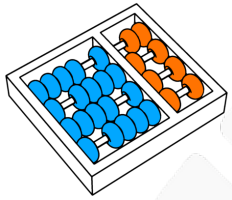
Tenta melhorar a estimativa $d[\mathbf{v}]$ examinando **(u,v)**.





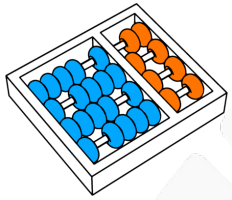
Algoritmo 4: RELAX(u, v, w)

```
1 se d[v] > d[u] + w(u, v)
2     d[v] ← d[u] + w(u, v)
3     π[v] ← u
```



Algoritmos baseados em relaxação

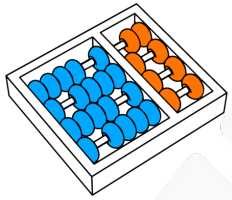
Características:



Algoritmos baseados em relaxação

Características:

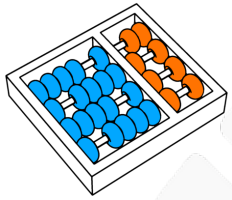
1. Inicializam d e π com uma sub-rotina INITIALIZE-SINGLE-SOURCE.



Algoritmos baseados em relaxação

Características:

1. Inicializam d e π com uma sub-rotina INITIALIZE-SINGLE-SOURCE.
2. Alteram d e π apenas com uma sub-rotina RELAX.

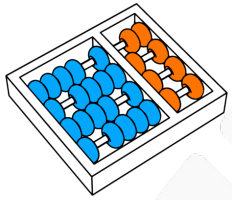


Algoritmos baseados em relaxação

Características:

1. Inicializam d e π com uma sub-rotina INITIALIZE-SINGLE-SOURCE.
2. Alteram d e π apenas com uma sub-rotina RELAX.

Esses algoritmos mantêm algumas invariantes:



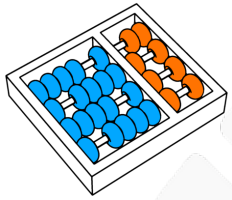
Algoritmos baseados em relaxação

Características:

1. Inicializam d e π com uma sub-rotina INITIALIZE-SINGLE-SOURCE.
2. Alteram d e π apenas com uma sub-rotina RELAX.

Esses algoritmos mantêm algumas invariantes:

- Existe um caminho de **s** a **v** com peso $d[\mathbf{v}]$.



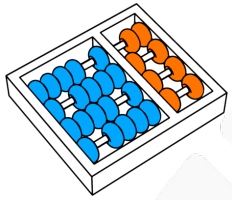
Algoritmos baseados em relaxação

Características:

1. Inicializam d e π com uma sub-rotina INITIALIZE-SINGLE-SOURCE.
2. Alteram d e π apenas com uma sub-rotina RELAX.

Esses algoritmos mantêm algumas invariantes:

- Existe um caminho de **s** a **v** com peso $d[\mathbf{v}]$.
- Esse caminho pode ser recuperado por meio π.



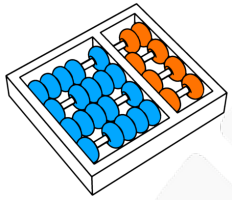
Algoritmos baseados em relaxação

Características:

1. Inicializam d e π com uma sub-rotina INITIALIZE-SINGLE-SOURCE.
2. Alteram d e π apenas com uma sub-rotina RELAX.

Esses algoritmos mantêm algumas invariantes:

- Existe um caminho de **s** a **v** com peso d[**v**].
- Esse caminho pode ser recuperado por meio π.
- Assim, d[**v**] é sempre **MAIOR OU IGUAL** a dist(**s**,**v**).



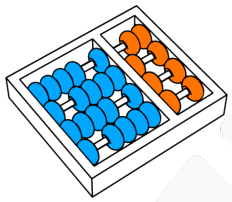
Algoritmos baseados em relaxação

Características:

1. Inicializam d e π com uma sub-rotina INITIALIZE-SINGLE-SOURCE.
2. Alteram d e π apenas com uma sub-rotina RELAX.

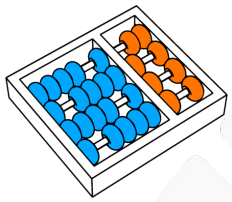
Esses algoritmos mantêm algumas invariantes:

- Existe um caminho de **s** a **v** com peso d[**v**].
- Esse caminho pode ser recuperado por meio π.
- Assim, d[**v**] é sempre **MAIOR OU IGUAL** a dist(**s**,**v**).
- Queremos que no final valha d[**v**] = dist(**s**,**v**).



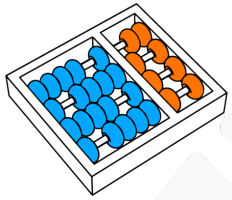
Propriedade de algoritmos baseados em relaxação

- **Limite superior:** Vale $d[\mathbf{v}] \geq \text{dist}(\mathbf{s},\mathbf{v})$ e, tão logo $d[\mathbf{v}]$ alcança $\text{dist}(\mathbf{s},\mathbf{v})$, nunca mais muda.



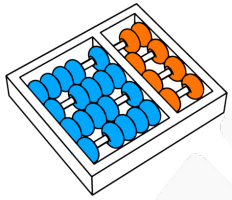
Propriedade de algoritmos baseados em relaxação

- **Limite superior:** Vale $d[\mathbf{v}] \geq \text{dist}(\mathbf{s},\mathbf{v})$ e, tão logo $d[\mathbf{v}]$ alcança $\text{dist}(\mathbf{s},\mathbf{v})$, nunca mais muda.
- **Inexistência de caminho:** Se não existe caminho de **s** a **v**, então $d[\mathbf{v}] = \infty$.



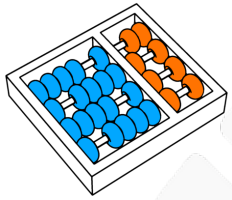
Propriedade de algoritmos baseados em relaxação

- **Limite superior:** Vale $d[\mathbf{v}] \geq \text{dist}(\mathbf{s},\mathbf{v})$ e, tão logo $d[\mathbf{v}]$ alcança $\text{dist}(\mathbf{s},\mathbf{v})$, nunca mais muda.
- **Inexistência de caminho:** Se não existe caminho de **s** a **v**, então $d[\mathbf{v}] = \infty$.
- **Subgrafo de predecessores:** Se $d[\mathbf{v}] < \infty$, então o subgrafo dos predecessores induzido por π é um caminho de peso $d[\mathbf{v}]$.



Propriedade de algoritmos baseados em relaxação

- **Limite superior:** Vale $d[\mathbf{v}] \geq \text{dist}(\mathbf{s},\mathbf{v})$ e, tão logo $d[\mathbf{v}]$ alcança $\text{dist}(\mathbf{s},\mathbf{v})$, nunca mais muda.
- **Inexistência de caminho:** Se não existe caminho de **s** a **v**, então $d[\mathbf{v}] = \infty$.
- **Subgrafo de predecessores:** Se $d[\mathbf{v}] < \infty$, então o subgrafo dos predecessores induzido por π é um caminho de peso $d[\mathbf{v}]$.
- **Convergência:** Se p é um caminho mínimo de **s** até **v** terminando com a aresta **(u,v)** e $d[\mathbf{u}] = \text{dist}(\mathbf{s},\mathbf{u})$, então ao relaxar **(u,v)**, $d[\mathbf{v}] = \text{dist}(\mathbf{s},\mathbf{v})$, que nunca mais muda.

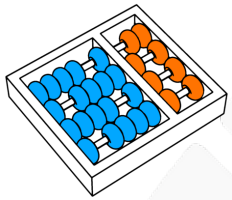


Propriedade de algoritmos baseados em relaxação

- **Limite superior:** Vale $d[v] \geq \text{dist}(s,v)$ e, tão logo $d[v]$ alcança $\text{dist}(s,v)$, nunca mais muda.
- **Inexistência de caminho:** Se não existe caminho de s a v, então $d[v] = \infty$.
- **Subgrafo de predecessores:** Se $d[v] < \infty$, então o subgrafo dos predecessores induzido por π é um caminho de peso $d[v]$.
- **Convergência:** Se p é um caminho mínimo de s até v terminando com a aresta (u,v) e $d[u] = \text{dist}(s,u)$, então ao relaxar (u,v), $d[v] = \text{dist}(s,v)$, que nunca mais muda.
- **Relaxamento de caminho:** Se $p = (v_0, v_1, \ldots, v_k)$ é um caminho mínimo de $s = v_0$ a v_k e relaxamos as arestas de p na ordem $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, então $d[v_k] = \text{dist}(s, v_k)$.

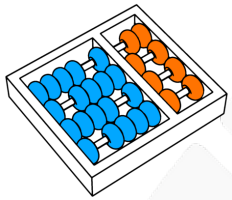
  A propriedade vale mesmo se tivermos realizado quaisquer outras relaxações durante a execução.

Algoritmo de Bellman-Ford



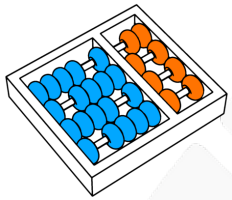
Arestas vs ciclos de peso negativo

- O algoritmo de Dijkstra resolve o Problema dos Caminhos Mínimos quando (G, w) **NÃO TEM ARESTAS DE PESO NEGATIVO**.



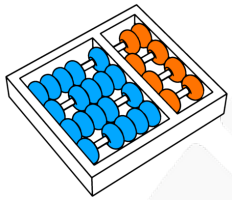
Arestas vs ciclos de peso negativo

- O algoritmo de Dijkstra resolve o Problema dos Caminhos Mínimos quando (G, w) **NÃO TEM ARESTAS DE PESO NEGATIVO**.
- Quando (G, w) possui arestas negativas, o algoritmo de Dijkstra não funciona.



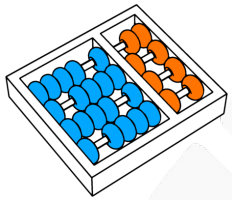
Arestas vs ciclos de peso negativo

- O algoritmo de Dijkstra resolve o Problema dos Caminhos Mínimos quando (G, w) **NÃO TEM ARESTAS DE PESO NEGATIVO**.
- Quando (G, w) possui arestas negativas, o algoritmo de Dijkstra não funciona.
- Uma das dificuldades com arestas negativas é a possível existência de **CICLOS DE PESO NEGATIVO** ou simplesmente ciclos negativos.



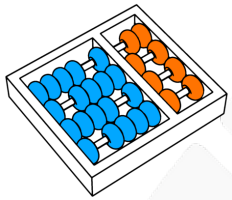
Ciclos negativos — uma dificuldade

- O Problema dos Caminhos Mínimos para instâncias com ciclos negativos é NP-**DIFÍCIL**.



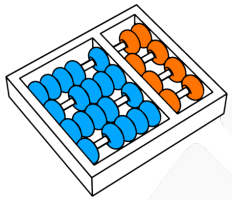
Ciclos negativos — uma dificuldade

- O Problema dos Caminhos Mínimos para instâncias com ciclos negativos é NP-**DIFÍCIL**.
  - Acreditamos que **NÃO** existem algoritmos eficientes para resolver problemas NP-difíceis.



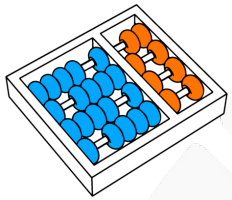
Ciclos negativos — uma dificuldade

- O Problema dos Caminhos Mínimos para instâncias com ciclos negativos é NP-**DIFÍCIL**.
  - Acreditamos que **NÃO** existem algoritmos eficientes para resolver problemas NP-difíceis.
  - Assim, vamos nos restringir ao Problema de Caminhos Mínimos **SEM CICLOS NEGATIVOS**.



Ciclos negativos — uma dificuldade

- O Problema dos Caminhos Mínimos para instâncias com ciclos negativos é NP-**DIFÍCIL**.
  - Acreditamos que **NÃO** existem algoritmos eficientes para resolver problemas NP-difíceis.
  - Assim, vamos nos restringir ao Problema de Caminhos Mínimos **SEM CICLOS NEGATIVOS**.
- Um algoritmo que resolve o problema restrito é o algoritmo de Bellman-Ford, que também é baseado em relaxação.

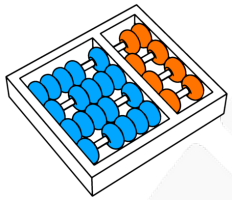


Definição do problema

Problema

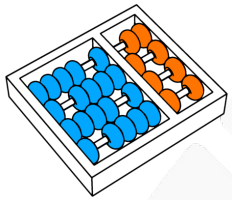
Entrada: *Um grafo direcionado* $G = (V, E)$, *uma função de peso* w *nas arestas e um vértice origem* s.

Saída: FALSE, *se existe um ciclo negativo atingível a partir de* s. *Caso contrário, além de* TRUE, *também devolve um vetor* d *com* $d[v] = \text{dist}(s,v)$ *para* $v \in V$ *e um vetor* π *definindo uma* ***ÁRVORE DE CAMINHOS MÍNIMOS***.



Ideia do algoritmo de Bellman-Ford

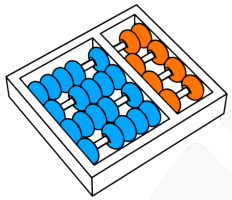
Relaxamento de caminho: Para **QUALQUER** caminho mínimo $(v_0, v_1, \ldots, v_k)$, queremos relaxar (v_0, v_1), (v_1, v_2), $\ldots$, (v_{k-1}, v_k) em ordem.



Ideia do algoritmo de Bellman-Ford

Relaxamento de caminho: Para **QUALQUER** caminho mínimo $(v_0, v_1, \ldots, v_k)$, queremos relaxar (v_0, v_1), (v_1, v_2), $\ldots$, (v_{k-1}, v_k) em ordem.

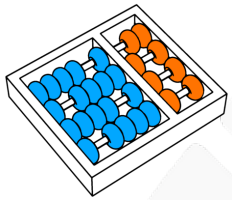
1. Executamos RELAX para todas as arestas:



Ideia do algoritmo de Bellman-Ford

Relaxamento de caminho: Para **QUALQUER** caminho mínimo $(v_0, v_1, \ldots, v_k)$, queremos relaxar (v_0, v_1), (v_1, v_2), $\ldots$, (v_{k-1}, v_k) em ordem.

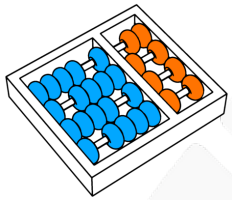
1. Executamos RELAX para todas as arestas:



Ideia do algoritmo de Bellman-Ford

Relaxamento de caminho: Para **QUALQUER** caminho mínimo $(v_0, v_1, \ldots, v_k)$, queremos relaxar (v_0, v_1), (v_1, v_2), $\ldots$, (v_{k-1}, v_k) em ordem.

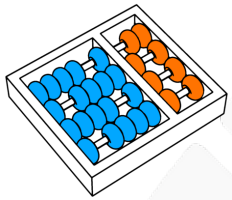
1. Executamos RELAX para todas as arestas:
   $\Rightarrow (v_0, v_1)$ relaxada.
2. Executamos novamente RELAX para todas as arestas:



Ideia do algoritmo de Bellman-Ford

Relaxamento de caminho: Para **QUALQUER** caminho mínimo $(v_0, v_1, \ldots, v_k)$, queremos relaxar (v_0, v_1), (v_1, v_2), $\ldots$, (v_{k-1}, v_k) em ordem.

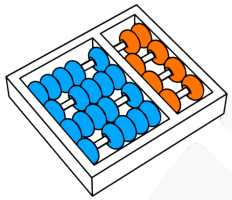
1. Executamos RELAX para todas as arestas:
   $\Rightarrow (v_0, v_1)$ relaxada.
2. Executamos novamente RELAX para todas as arestas:



Ideia do algoritmo de Bellman-Ford

Relaxamento de caminho: Para **QUALQUER** caminho mínimo $(v_0, v_1, \ldots, v_k)$, queremos relaxar (v_0, v_1), (v_1, v_2), $\ldots$, (v_{k-1}, v_k) em ordem.

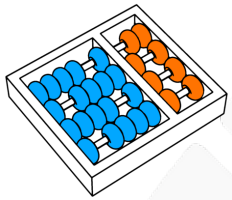
1. Executamos RELAX para todas as arestas:
   $\Rightarrow (v_0, v_1)$ relaxada.
2. Executamos novamente RELAX para todas as arestas:
   $\Rightarrow (v_0, v_1)$, (v_1, v_2) relaxadas em ordem.
3. Executamos novamente RELAX para todas as arestas:



Ideia do algoritmo de Bellman-Ford

Relaxamento de caminho: Para **QUALQUER** caminho mínimo $(v_0, v_1, \ldots, v_k)$, queremos relaxar (v_0, v_1), (v_1, v_2), $\ldots$, (v_{k-1}, v_k) em ordem.

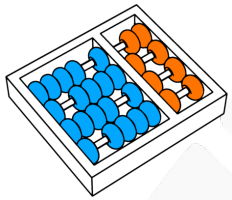
1. Executamos RELAX para todas as arestas:
   $\Rightarrow (v_0, v_1)$ relaxada.
2. Executamos novamente RELAX para todas as arestas:
   $\Rightarrow (v_0, v_1)$, (v_1, v_2) relaxadas em ordem.
3. Executamos novamente RELAX para todas as arestas:



Ideia do algoritmo de Bellman-Ford

Relaxamento de caminho: Para **QUALQUER** caminho mínimo $(v_0, v_1, \ldots, v_k)$, queremos relaxar $(v_0, v_1),\ (v_1, v_2),\ \ldots,\ (v_{k-1}, v_k)$ em ordem.

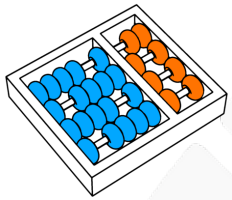
1. Executamos RELAX para todas as arestas:
   $\Rightarrow (v_0, v_1)$ relaxada.
2. Executamos novamente RELAX para todas as arestas:
   $\Rightarrow (v_0, v_1),\ (v_1, v_2)$ relaxadas em ordem.
3. Executamos novamente RELAX para todas as arestas:
   $\Rightarrow (v_0, v_1),\ (v_1, v_2),\ (v_2, v_3)$ relaxadas em ordem.
4. Repetimos esse passo até $|V| - 1$ vezes.



Ideia do algoritmo de Bellman-Ford

Relaxamento de caminho: Para **QUALQUER** caminho mínimo $(v_0, v_1, \ldots, v_k)$, queremos relaxar $(v_0, v_1),\ (v_1, v_2),\ \ldots,\ (v_{k-1}, v_k)$ em ordem.

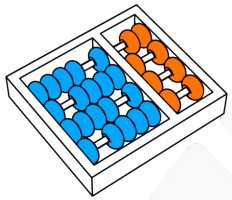
1. Executamos RELAX para todas as arestas:
   $\Rightarrow (v_0, v_1)$ relaxada.
2. Executamos novamente RELAX para todas as arestas:
   $\Rightarrow (v_0, v_1),\ (v_1, v_2)$ relaxadas em ordem.
3. Executamos novamente RELAX para todas as arestas:
   $\Rightarrow (v_0, v_1),\ (v_1, v_2),\ (v_2, v_3)$ relaxadas em ordem.
4. Repetimos esse passo até $|V| - 1$ vezes.



Ideia do algoritmo de Bellman-Ford

Relaxamento de caminho: Para **QUALQUER** caminho mínimo $(v_0, v_1, \ldots, v_k)$, queremos relaxar (v_0, v_1), (v_1, v_2), $\ldots$, (v_{k-1}, v_k) em ordem.

1. Executamos RELAX para todas as arestas:
   $\Rightarrow (v_0, v_1)$ relaxada.
2. Executamos novamente RELAX para todas as arestas:
   $\Rightarrow (v_0, v_1)$, (v_1, v_2) relaxadas em ordem.
3. Executamos novamente RELAX para todas as arestas:
   $\Rightarrow (v_0, v_1)$, (v_1, v_2), (v_2, v_3) relaxadas em ordem.
4. Repetimos esse passo até $|V| - 1$ vezes. Por quê?

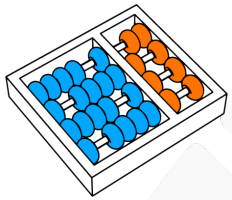


Ideia do algoritmo de Bellman-Ford

Relaxamento de caminho: Para **QUALQUER** caminho mínimo $(v_0, v_1, \ldots, v_k)$, queremos relaxar (v_0, v_1), (v_1, v_2), $\ldots$, (v_{k-1}, v_k) em ordem.

1. Executamos RELAX para todas as arestas:
   $\Rightarrow$ (v_0, v_1) relaxada.
2. Executamos novamente RELAX para todas as arestas:
   $\Rightarrow$ (v_0, v_1), (v_1, v_2) relaxadas em ordem.
3. Executamos novamente RELAX para todas as arestas:
   $\Rightarrow$ (v_0, v_1), (v_1, v_2), (v_2, v_3) relaxadas em ordem.
4. Repetimos esse passo até $|V| - 1$ vezes. Por quê?
   $\Rightarrow$ (v_0, v_1), (v_1, v_2), $\ldots$, (v_{k-1}, v_k) relaxadas em ordem.

Podemos verificar se o grafo contém **CICLOS NEGATIVOS** executando mais uma vez:



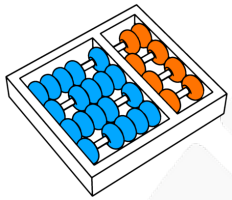
Ideia do algoritmo de Bellman-Ford

Relaxamento de caminho: Para **QUALQUER** caminho mínimo $(v_0, v_1, \ldots, v_k)$, queremos relaxar $(v_0, v_1),\ (v_1, v_2),\ \ldots,\ (v_{k-1}, v_k)$ em ordem.

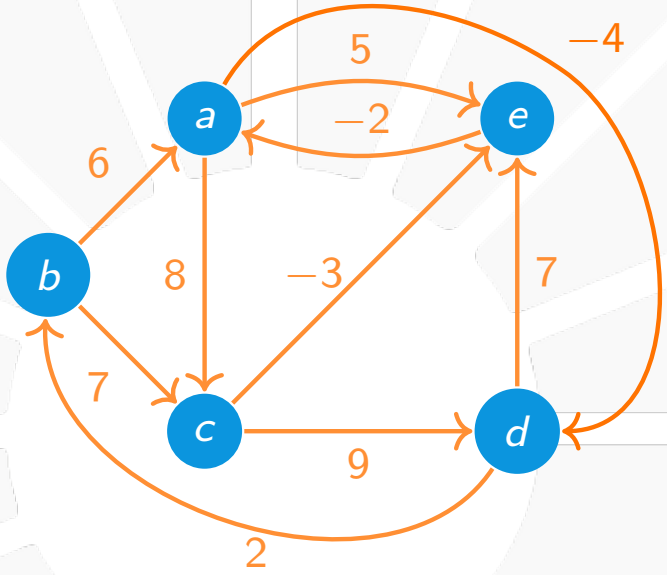
1. Executamos RELAX para todas as arestas:
   $\Rightarrow (v_0, v_1)$ relaxada.
2. Executamos novamente RELAX para todas as arestas:
   $\Rightarrow (v_0, v_1),\ (v_1, v_2)$ relaxadas em ordem.
3. Executamos novamente RELAX para todas as arestas:
   $\Rightarrow (v_0, v_1),\ (v_1, v_2),\ (v_2, v_3)$ relaxadas em ordem.
4. Repetimos esse passo até $|V| - 1$ vezes. Por quê?
   $\Rightarrow (v_0, v_1),\ (v_1, v_2),\ \ldots,\ (v_{k-1}, v_k)$ relaxadas em ordem.

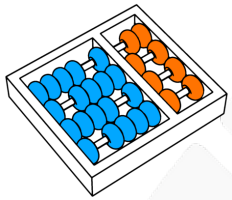
Podemos verificar se o grafo contém **CICLOS NEGATIVOS** executando mais uma vez:

- Se algum valor $d[v]$ diminuir, então há ciclo negativo.

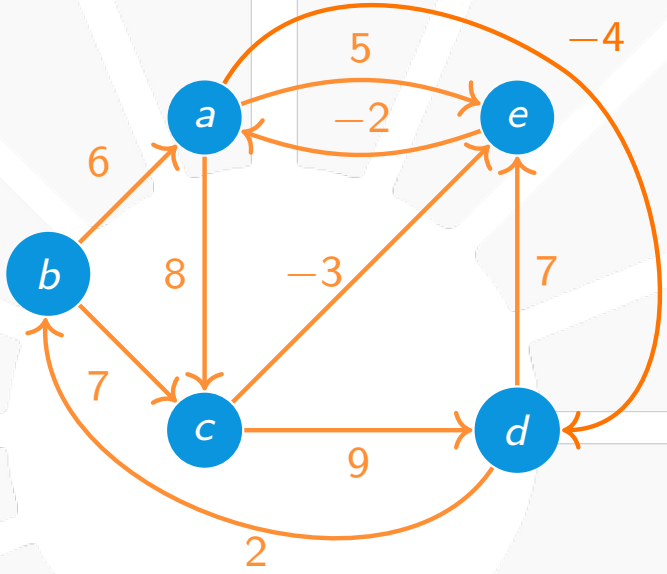


Exemplo



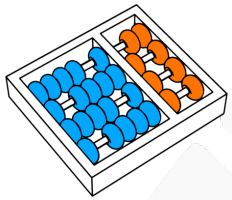


Exemplo

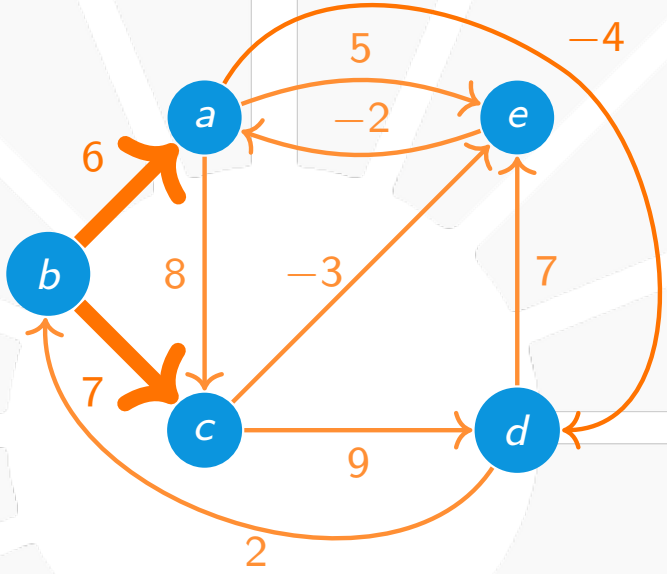


	a	b	c	d	e
d	∞	0	∞	∞	∞

Ordem: **(a,c),(a,d),(a,e),(c,d),(c,e),(d,b),(d,e),(e,a),(b,a),(b,c)**.

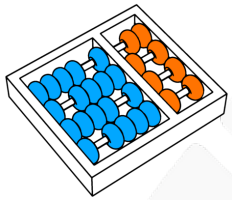


Exemplo

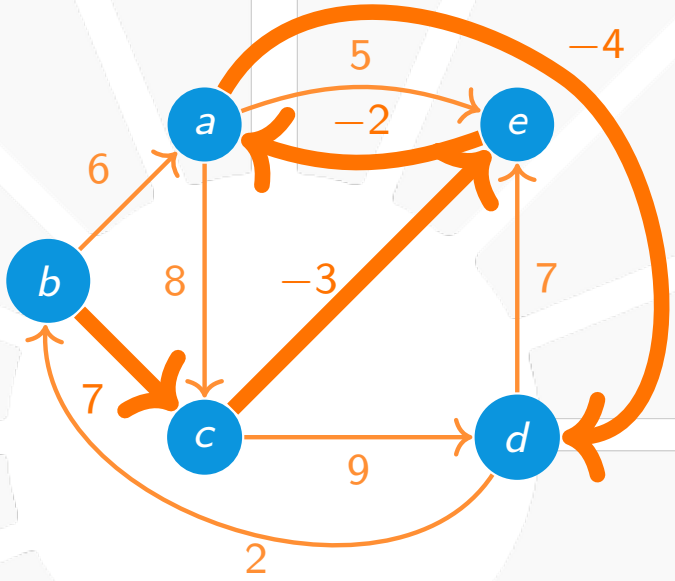


	a	b	c	d	e
d	6	0	7	∞	∞

Ordem: **(a,c),(a,d),(a,e),(c,d),(c,e),(d,b),(d,e),(e,a),(b,a),(b,c)**.

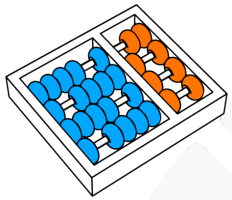


Exemplo

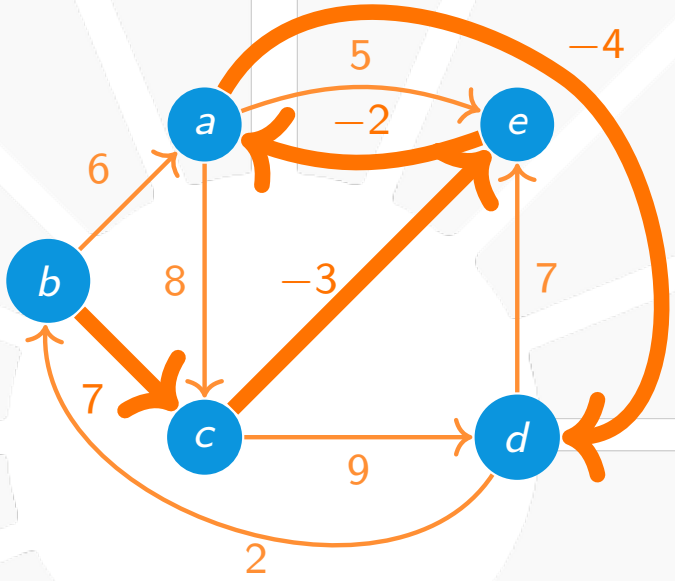


	a	b	c	d	e
d	2	0	7	2	4

Ordem: **(a,c),(a,d),(a,e),(c,d),(c,e),(d,b),(d,e),(e,a),(b,a),(b,c)**.

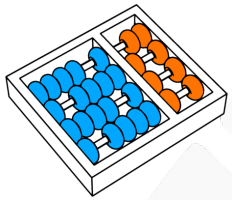


Exemplo



	a	b	c	d	e
d	2	0	7	−2	4

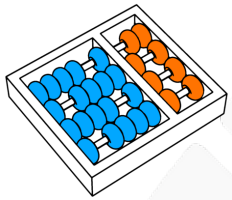
Ordem: (a,c),(a,d),(a,e),(c,d),(c,e),(d,b),(d,e),(e,a),(b,a),(b,c).



O algoritmo de Bellman-Ford

Algoritmo 5: BELLMAN-FORD(G, w, s)

```
1 INITIALIZE-SINGLE-SOURCE(G, s)
2 para i ← 1 até |V[G]| − 1
3 |   para cada (u, v) ∈ E[G]
4 |   |   RELAX(u, v, w)
5 para cada (u, v) ∈ E[G]
6 |   se d[v] > d[u] + w(u, v)
7 |   |   devolva FALSE
8 devolva TRUE, d, π
```

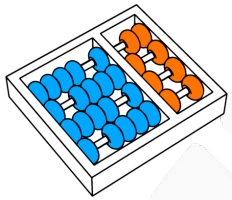


O algoritmo de Bellman-Ford

Algoritmo 6: BELLMAN-FORD(G, w, s)

```
1 INITIALIZE-SINGLE-SOURCE(G, s)
2 para i ← 1 até |V[G]| − 1
3 |   para cada (u, v) ∈ E[G]
4 |   |   RELAX(u, v, w)
5 para cada (u, v) ∈ E[G]
6 |   se d[v] > d[u] + w(u, v)
7 |   |   devolva FALSE
8 devolva TRUE, d, π
```

Complexidade de tempo: $O(VE)$

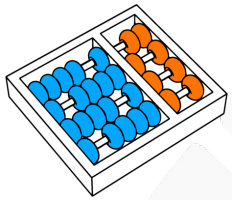


Correção do algoritmo Bellman-Ford

Teorema

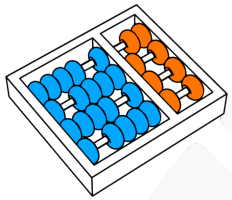
BELLMAN-FORD *devolve:*

- FALSE, *se existe um ciclo negativo atingível a partir de* ***s***.
- TRUE, *caso contrário; neste caso devolve também:*
  - *Um vetor* d *com* $d[\boldsymbol{v}] = \text{dist}(\boldsymbol{s},\boldsymbol{v})$ *para* $\boldsymbol{v} \in \boldsymbol{V}$.
  - *Um vetor* π *definindo uma* ***ÁRVORE DE CAMINHOS MÍNIMOS***.



Correção do algoritmo Bellman-Ford

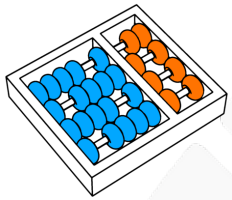
Primeiro, suponha que não há ciclos negativos atingíveis por **s**.



Correção do algoritmo Bellman-Ford

Primeiro, suponha que não há ciclos negativos atingíveis por **s**.

Considere **v** $\in$ **V[G]** e os valores de d e π após o primeiro laço:

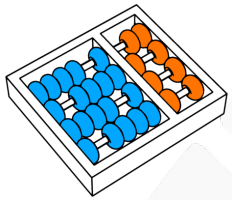


Correção do algoritmo Bellman-Ford

Primeiro, suponha que não há ciclos negativos atingíveis por **s**.

Considere **v** $\in$ **V[G]** e os valores de d e π após o primeiro laço:

- Se **v** não é atingível, $d[\mathbf{v}] = \infty$ (por **Inexistência de caminho**).

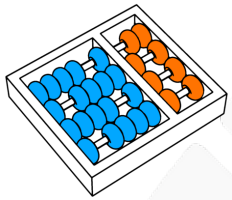


Correção do algoritmo Bellman-Ford

Primeiro, suponha que não há ciclos negativos atingíveis por **s**.

Considere $\mathbf{v} \in \mathbf{V[G]}$ e os valores de d e π após o primeiro laço:

- Se **v** não é atingível, $d[\mathbf{v}] = \infty$ (por **Inexistência de caminho**).
- Senão, existe caminho mínimo $(\mathbf{v_0}, \mathbf{v_1}, \ldots, \mathbf{v_k})$ de $\mathbf{s} = \mathbf{v_0}$ a $\mathbf{v} = \mathbf{v_k}$.

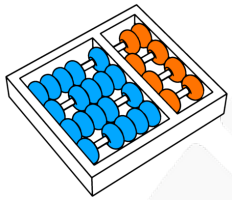


Correção do algoritmo Bellman-Ford

Primeiro, suponha que não há ciclos negativos atingíveis por $\mathbf{s}$.

Considere $\mathbf{v} \in \mathbf{V[G]}$ e os valores de d e π após o primeiro laço:

- Se $\mathbf{v}$ não é atingível, $d[\mathbf{v}] = \infty$ (por **Inexistência de caminho**).
- Senão, existe caminho mínimo $(\mathbf{v_0}, \mathbf{v_1}, \ldots, \mathbf{v_k})$ de $\mathbf{s} = \mathbf{v_0}$ a $\mathbf{v} = \mathbf{v_k}$.
- Como $k \leq |V| - 1$, então $(\mathbf{v_0}, \mathbf{v_1}),\ (\mathbf{v_1}, \mathbf{v_2}),\ \ldots,\ (\mathbf{v_{k-1}}, \mathbf{v_k})$ foram relaxadas **NESTA ORDEM**.

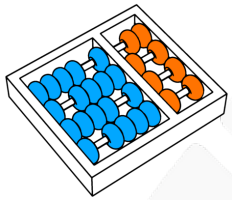


Correção do algoritmo Bellman-Ford

Primeiro, suponha que não há ciclos negativos atingíveis por **s**.

Considere $\mathbf{v} \in \mathbf{V[G]}$ e os valores de d e π após o primeiro laço:

- Se **v** não é atingível, $d[\mathbf{v}] = \infty$ (por **Inexistência de caminho**).
- Senão, existe caminho mínimo $(\mathbf{v_0}, \mathbf{v_1}, \ldots, \mathbf{v_k})$ de $\mathbf{s} = \mathbf{v_0}$ a $\mathbf{v} = \mathbf{v_k}$.
- Como $k \leq |V| - 1$, então $(\mathbf{v_0}, \mathbf{v_1}),\ (\mathbf{v_1}, \mathbf{v_2}),\ \ldots,\ (\mathbf{v_{k-1}}, \mathbf{v_k})$ foram relaxadas **NESTA ORDEM**.
- Por **Relaxamento de caminho**, $d[\mathbf{v}] = \text{dist}(\mathbf{s},\mathbf{v})$.

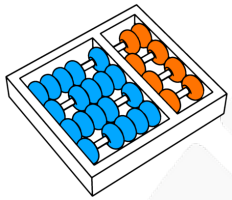


Correção do algoritmo Bellman-Ford

Primeiro, suponha que não há ciclos negativos atingíveis por **s**.

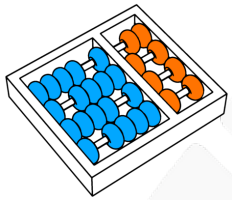
Considere $\mathbf{v} \in \mathbf{V[G]}$ e os valores de d e π após o primeiro laço:

- Se **v** não é atingível, $d[\mathbf{v}] = \infty$ (por **Inexistência de caminho**).
- Senão, existe caminho mínimo $(\mathbf{v_0}, \mathbf{v_1}, \ldots, \mathbf{v_k})$ de $\mathbf{s} = \mathbf{v_0}$ a $\mathbf{v} = \mathbf{v_k}$.
- Como $k \leq |V| - 1$, então $(\mathbf{v_0}, \mathbf{v_1}),\ (\mathbf{v_1}, \mathbf{v_2}),\ \ldots,\ (\mathbf{v_{k-1}}, \mathbf{v_k})$ foram relaxadas **NESTA ORDEM**.
- Por **Relaxamento de caminho**, $d[\mathbf{v}] = \text{dist}(\mathbf{s},\mathbf{v})$.
- Também, por **Sugrafo de predecessores**: π induz um caminho mínimo de **s** a **v**.



Correção do algoritmo Bellman-Ford

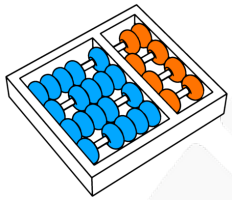
Também temos que mostrar que nesse caso BELLMAN-FORD devolve TRUE.



Correção do algoritmo Bellman-Ford

Também temos que mostrar que nesse caso BELLMAN-FORD devolve TRUE.

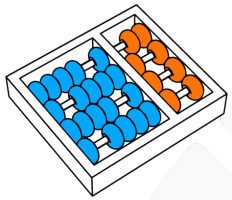
- Considere d imediatamente após o primeiro laço.



Correção do algoritmo Bellman-Ford

Também temos que mostrar que nesse caso BELLMAN-FORD devolve TRUE.

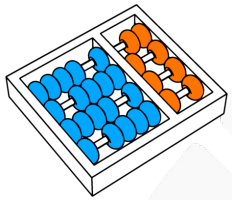
- Considere d imediatamente após o primeiro laço.
- Nesse instante, $d[\mathbf{v}] = \text{dist}(\mathbf{s},\mathbf{v})$ para todo vértice **v**.



Correção do algoritmo Bellman-Ford

Também temos que mostrar que nesse caso BELLMAN-FORD devolve TRUE.

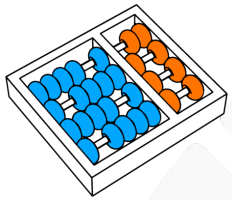
- Considere d imediatamente após o primeiro laço.
- Nesse instante, $d[\mathbf{v}] = \text{dist}(\mathbf{s},\mathbf{v})$ para todo vértice **v**.
- Por **Convergência**, sabemos que d nunca mais muda.



Correção do algoritmo Bellman-Ford

Também temos que mostrar que nesse caso BELLMAN-FORD devolve TRUE.

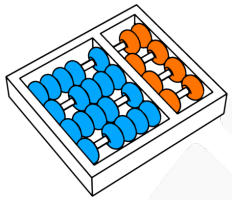
- Considere d imediatamente após o primeiro laço.
- Nesse instante, $d[\mathbf{v}] = \text{dist}(\mathbf{s},\mathbf{v})$ para todo vértice **v**.
- Por **Convergência**, sabemos que d nunca mais muda.
- Portanto, o teste da linha 6 falha sempre.



Correção do algoritmo Bellman-Ford

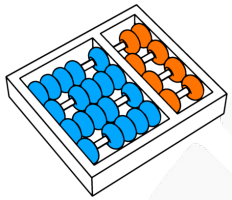
Também temos que mostrar que nesse caso BELLMAN-FORD devolve TRUE.

- Considere d imediatamente após o primeiro laço.
- Nesse instante, $d[\mathbf{v}] = \text{dist}(\mathbf{s},\mathbf{v})$ para todo vértice **v**.
- Por **Convergência**, sabemos que d nunca mais muda.
- Portanto, o teste da linha 6 falha sempre.
- Concluindo que o algoritmo devolve TRUE.



Correção do algoritmo Bellman-Ford

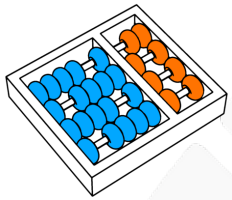
Suponha agora que (G, w) contenha **CICLO NEGATIVO** alcançável por **s**.



Correção do algoritmo Bellman-Ford

Suponha agora que (G, w) contenha **CICLO NEGATIVO** alcançável por **s**.

Queremos mostrar que o algoritmo devolve FALSE:



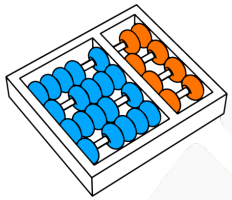
Correção do algoritmo Bellman-Ford

Suponha agora que (G, w) contenha **CICLO NEGATIVO** alcançável por $\mathbf{s}$.

Queremos mostrar que o algoritmo devolve FALSE:

- Seja $C = (\mathbf{v_0}, \mathbf{v_1}, \ldots, \mathbf{v_k} = \mathbf{v_0})$ um ciclo tal que

$$w(C) = \sum_{i=1}^{k} w(\mathbf{v_{i-1}}, \mathbf{v_i}) < 0.$$



Correção do algoritmo Bellman-Ford

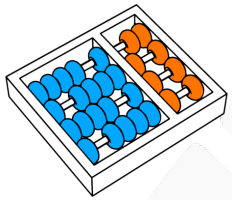
Suponha agora que (G, w) contenha **CICLO NEGATIVO** alcançável por $\mathbf{s}$.

Queremos mostrar que o algoritmo devolve FALSE:

- Seja $C = (\mathbf{v_0}, \mathbf{v_1}, \ldots, \mathbf{v_k} = \mathbf{v_0})$ um ciclo tal que

$$w(C) = \sum_{i=1}^{k} w(\mathbf{v_{i-1}}, \mathbf{v_i}) < 0.$$

- Suponha, por contradição que o algoritmo devolve TRUE.



Correção do algoritmo Bellman-Ford

Suponha agora que (G, w) contenha **CICLO NEGATIVO** alcançável por $\mathbf{s}$.

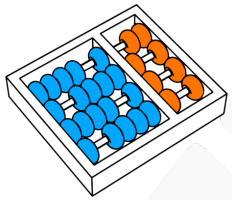
Queremos mostrar que o algoritmo devolve FALSE:

- Seja $C = (\mathbf{v_0}, \mathbf{v_1}, \ldots, \mathbf{v_k} = \mathbf{v_0})$ um ciclo tal que

$$w(C) = \sum_{i=1}^{k} w(\mathbf{v_{i-1}}, \mathbf{v_i}) < 0.$$

- Suponha, por contradição que o algoritmo devolve TRUE.
- Como relaxamos cada aresta $(\mathbf{v_{i-1}}, \mathbf{v_i})$:

$$d[\mathbf{v_i}] \leq d[\mathbf{v_{i-1}}] + w(\mathbf{v_{i-1}}, \mathbf{v_i}).$$

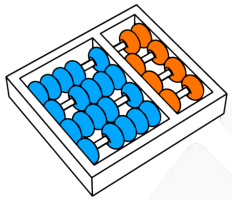


Correção do algoritmo Bellman-Ford

- Somando as desigualdades anteriores para cada aresta do ciclo, temos:

$$\sum_{i=1}^{k} d[v_i] \leq \sum_{i=1}^{k} (d[v_{i-1}] + w(v_{i-1}, v_i))$$

$$= \sum_{i=1}^{k} d[v_{i-1}] + \sum_{i=1}^{k} w(v_{i-1}, v_i).$$



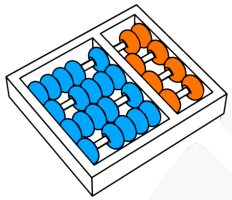
Correção do algoritmo Bellman-Ford

- Somando as desigualdades anteriores para cada aresta do ciclo, temos:

$$\sum_{i=1}^{k} d[v_i] \leq \sum_{i=1}^{k} (d[v_{i-1}] + w(v_{i-1}, v_i))$$

$$= \sum_{i=1}^{k} d[v_{i-1}] + \sum_{i=1}^{k} w(v_{i-1}, v_i).$$

- Como $v_0 = v_k$, temos que $\sum_{i=1}^{k} d[v_i] = \sum_{i=1}^{k} d[v_{i-1}]$.



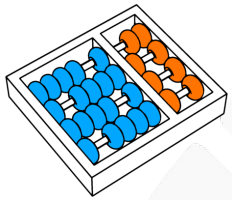
Correção do algoritmo Bellman-Ford

- Somando as desigualdades anteriores para cada aresta do ciclo, temos:

$$\sum_{i=1}^{k} d[v_i] \leq \sum_{i=1}^{k} (d[v_{i-1}] + w(v_{i-1}, v_i))$$

$$= \sum_{i=1}^{k} d[v_{i-1}] + \sum_{i=1}^{k} w(v_{i-1}, v_i).$$

- Como $v_0 = v_k$, temos que $\sum_{i=1}^{k} d[v_i] = \sum_{i=1}^{k} d[v_{i-1}]$.
- Logo, $0 \leq \sum_{i=1}^{k} w(v_{i-1}, v_i) = w(C)$.



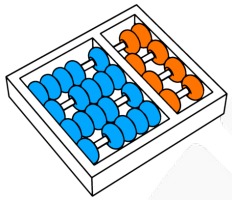
Correção do algoritmo Bellman-Ford

- Somando as desigualdades anteriores para cada aresta do ciclo, temos:

$$\sum_{i=1}^{k} d[v_i] \leq \sum_{i=1}^{k} (d[v_{i-1}] + w(v_{i-1}, v_i))$$

$$= \sum_{i=1}^{k} d[v_{i-1}] + \sum_{i=1}^{k} w(v_{i-1}, v_i).$$

- Como $v_0 = v_k$, temos que $\sum_{i=1}^{k} d[v_i] = \sum_{i=1}^{k} d[v_{i-1}]$.
- Logo, $0 \leq \sum_{i=1}^{k} w(v_{i-1}, v_i) = w(C)$.
- Mas isso é uma contradição, pois C é ciclo negativo.



Correção do algoritmo Bellman-Ford

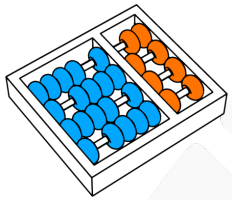
- Somando as desigualdades anteriores para cada aresta do ciclo, temos:

$$\sum_{i=1}^{k} d[v_i] \leq \sum_{i=1}^{k} (d[v_{i-1}] + w(v_{i-1}, v_i))$$

$$= \sum_{i=1}^{k} d[v_{i-1}] + \sum_{i=1}^{k} w(v_{i-1}, v_i).$$

- Como $v_0 = v_k$, temos que $\sum_{i=1}^{k} d[v_i] = \sum_{i=1}^{k} d[v_{i-1}]$.
- Logo, $0 \leq \sum_{i=1}^{k} w(v_{i-1}, v_i) = w(C)$.
- Mas isso é uma contradição, pois C é ciclo negativo.
- Concluindo que, nesse caso, o algoritmo devolve FALSE.

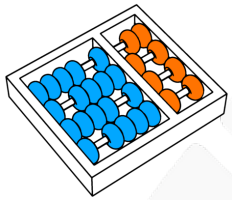
Sistemas de restrições de diferenças



Exemplo

Uma aplicação de caminhos mínimos é encontrar $x_1, x_2, \ldots, x_n$ que satisfaçam:

$$
\begin{array}{rcl}
x_1 - x_2 & \leq & 0 \\
x_1 - x_5 & \leq & -1 \\
x_2 - x_5 & \leq & 1 \\
x_3 - x_1 & \leq & 5 \\
x_4 - x_1 & \leq & 4 \\
x_4 - x_3 & \leq & -1 \\
x_5 - x_3 & \leq & -3 \\
x_5 - x_4 & \leq & -3
\end{array}
$$

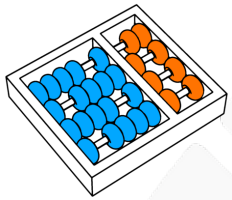


Exemplo

Uma aplicação de caminhos mínimos é encontrar $x_1, x_2, \ldots, x_n$ que satisfaçam:

$$
\begin{array}{rcl}
x_1 - x_2 & \leq & 0 \\
x_1 - x_5 & \leq & -1 \\
x_2 - x_5 & \leq & 1 \\
x_3 - x_1 & \leq & 5 \\
x_4 - x_1 & \leq & 4 \\
x_4 - x_3 & \leq & -1 \\
x_5 - x_3 & \leq & -3 \\
x_5 - x_4 & \leq & -3
\end{array}
$$

Onde:



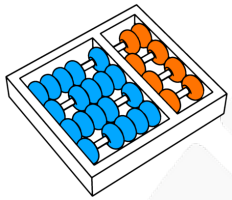
Exemplo

Uma aplicação de caminhos mínimos é encontrar $x_1, x_2, \ldots, x_n$ que satisfaçam:

$$\begin{aligned}
x_1 - x_2 &\leq 0 \\
x_1 - x_5 &\leq -1 \\
x_2 - x_5 &\leq 1 \\
x_3 - x_1 &\leq 5 \\
x_4 - x_1 &\leq 4 \\
x_4 - x_3 &\leq -1 \\
x_5 - x_3 &\leq -3 \\
x_5 - x_4 &\leq -3
\end{aligned}$$

Onde:

- x_i representa a hora do evento i



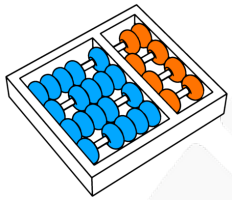
Exemplo

Uma aplicação de caminhos mínimos é encontrar $x_1, x_2, \ldots, x_n$ que satisfaçam:

$$\begin{array}{rcl} x_1 - x_2 & \leq & 0 \\ x_1 - x_5 & \leq & -1 \\ x_2 - x_5 & \leq & 1 \\ x_3 - x_1 & \leq & 5 \\ x_4 - x_1 & \leq & 4 \\ x_4 - x_3 & \leq & -1 \\ x_5 - x_3 & \leq & -3 \\ x_5 - x_4 & \leq & -3 \end{array}$$

Onde:

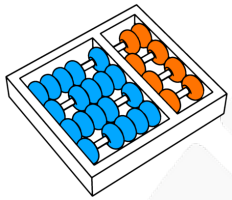
- x_i representa a hora do evento i
- $x_j - x_i \leq b_k$ significa que deve haver um intervalo de pelo menos b_k horas entre eventos i e j



Exemplo

Podemos reescrever as restrições de forma matricial:

$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} \leq \begin{pmatrix} 0 \\ -1 \\ 1 \\ 5 \\ 4 \\ -1 \\ -3 \\ -3 \end{pmatrix}$$



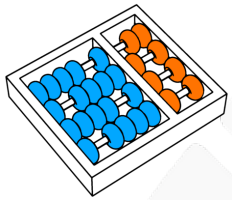
Exemplo

Podemos reescrever as restrições de forma matricial:

$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} \leq \begin{pmatrix} 0 \\ -1 \\ 1 \\ 5 \\ 4 \\ -1 \\ -3 \\ -3 \end{pmatrix}$$

Algumas soluções:

- $x = (-5, -3, 0, -1, -4)$,



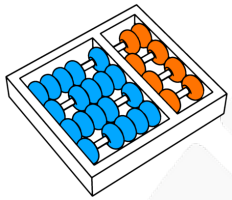
Exemplo

Podemos reescrever as restrições de forma matricial:

$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} \leq \begin{pmatrix} 0 \\ -1 \\ 1 \\ 5 \\ 4 \\ -1 \\ -3 \\ -3 \end{pmatrix}$$

Algumas soluções:

- $x = (-5, -3, 0, -1, -4)$,
- $x' = (0, 2, 5, 4, 1)$,



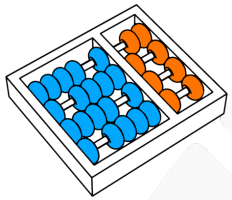
Exemplo

Podemos reescrever as restrições de forma matricial:

$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} \leq \begin{pmatrix} 0 \\ -1 \\ 1 \\ 5 \\ 4 \\ -1 \\ -3 \\ -3 \end{pmatrix}$$

Algumas soluções:

- $x = (-5, -3, 0, -1, -4)$,
- $x' = (0, 2, 5, 4, 1)$,...



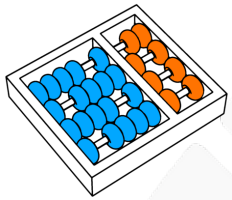
Soluções de sistemas de restrições de diferenças

Lema

Seja $x = (x_1, \ldots, x_n)$ *uma solução de um sistema de restrições de diferença* $Ax \leq b$ *e* d *uma constante. Então*

$$x + d = (x_1 + d, \ldots, x_n + d)$$

também é uma solução de $Ax \leq b$.



Soluções de sistemas de restrições de diferenças

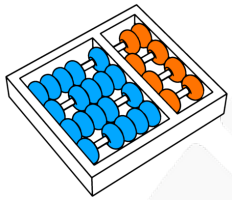
Lema

Seja $x = (x_1, \ldots, x_n)$ uma solução de um sistema de restrições de diferença $Ax \leq b$ e d uma constante. Então

$$x + d = (x_1 + d, \ldots, x_n + d)$$

também é uma solução de $Ax \leq b$.

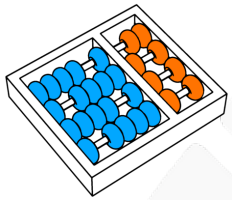
Mostraremos a seguir como encontrar uma solução de um sistema $Ax \leq b$ de restrições de diferença resolvendo um problema de caminhos mínimos.



Grafo de restrições

Construímos o chamado **GRAFO DE RESTRIÇÕES** fazendo o seguinte:

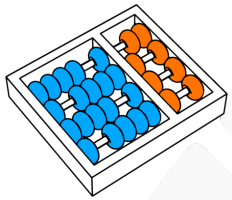
1. Primeiro criamos um grafo em que:



Grafo de restrições

Construímos o chamado **GRAFO DE RESTRIÇÕES** fazendo o seguinte:

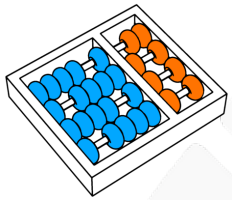
1. Primeiro criamos um grafo em que:
   - Cada vértice $\mathbf{v_i}$ corresponde a uma variável x_i.



Grafo de restrições

Construímos o chamado **GRAFO DE RESTRIÇÕES** fazendo o seguinte:

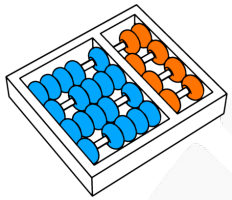
1. Primeiro criamos um grafo em que:
   - Cada vértice $\mathbf{v_i}$ corresponde a uma variável x_i.
   - Cada aresta $(\mathbf{v_i}, \mathbf{v_j})$ tem peso b_k e corresponde a uma restrição $x_j - x_i \leq b_k$.



Grafo de restrições

Construímos o chamado **GRAFO DE RESTRIÇÕES** fazendo o seguinte:

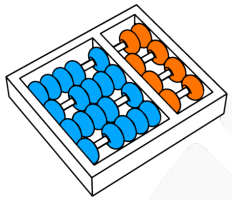
1. Primeiro criamos um grafo em que:
   - Cada vértice $\mathbf{v_i}$ corresponde a uma variável x_i.
   - Cada aresta $(\mathbf{v_i}, \mathbf{v_j})$ tem peso b_k e corresponde a uma restrição $x_j - x_i \leq b_k$.
   - Logo, a matriz A do sistema de restrições corresponde à transposta matriz de incidência desse grafo obtido



Grafo de restrições

Construímos o chamado **GRAFO DE RESTRIÇÕES** fazendo o seguinte:

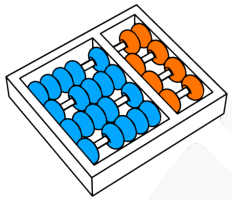
1. Primeiro criamos um grafo em que:
   - Cada vértice $\mathbf{v_i}$ corresponde a uma variável x_i.
   - Cada aresta $(\mathbf{v_i}, \mathbf{v_j})$ tem peso b_k e corresponde a uma restrição $x_j - x_i \leq b_k$.
   - Logo, a matriz A do sistema de restrições corresponde à transposta matriz de incidência desse grafo obtido
2. Finalmente, adicionamos um vértice especial $\mathbf{v_0}$ e uma aresta de $\mathbf{v_0}$ a cada outro vértice $\mathbf{v_i}$ com peso 0.



Grafo de restrições

Formalmente, dado o sistema $Ax \leq b$ de restrições de diferença, construímos o grafo $G = (\mathbf{V}, \mathbf{E})$ tal que:

- $\mathbf{V} = \{\mathbf{v_0}, \mathbf{v_1}, \ldots, \mathbf{v_n}\}$.
- $\mathbf{E} = \{(\mathbf{v_0}, \mathbf{v_1}), \ldots, (\mathbf{v_0}, \mathbf{v_n})\} \cup \{(\mathbf{v_i}, \mathbf{v_j}) : \mathbf{x_j} - \mathbf{x_i} \leq \mathbf{b_k} \text{ é restrição}\}$



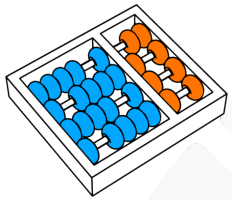
Grafo de restrições

Formalmente, dado o sistema $Ax \leq b$ de restrições de diferença, construímos o grafo $G = (\mathbf{V}, \mathbf{E})$ tal que:

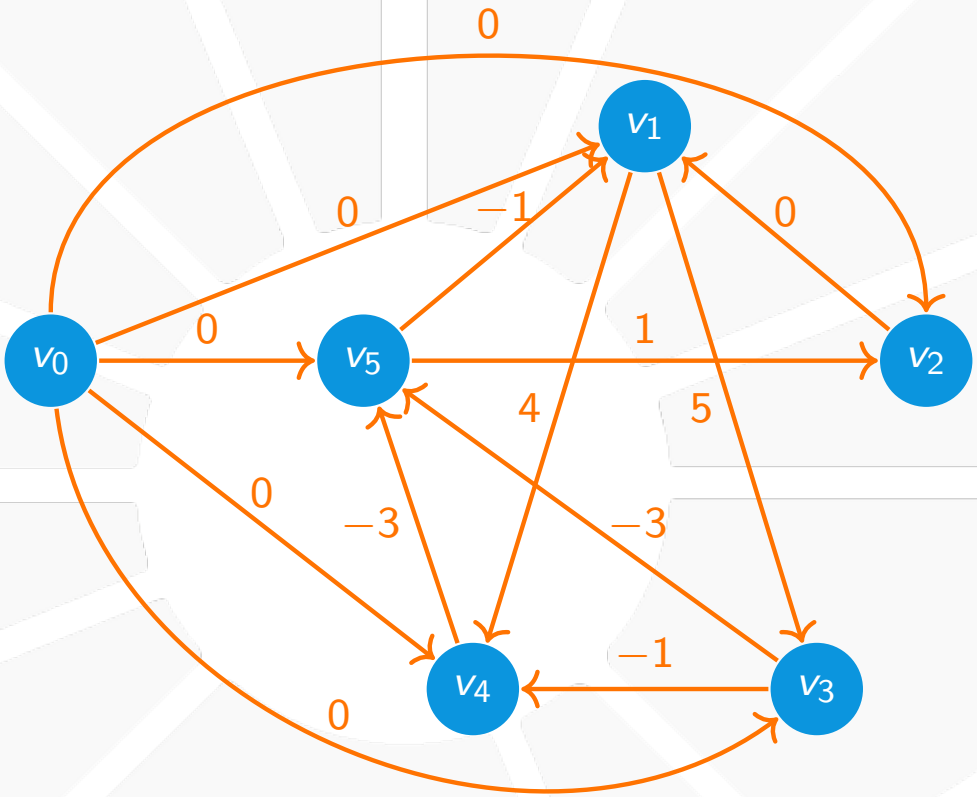
- $\mathbf{V} = \{\mathbf{v_0}, \mathbf{v_1}, \ldots, \mathbf{v_n}\}$.
- $\mathbf{E} = \{(\mathbf{v_0}, \mathbf{v_1}), \ldots, (\mathbf{v_0}, \mathbf{v_n})\} \cup \{(\mathbf{v_i}, \mathbf{v_j}) : \mathbf{x_j} - \mathbf{x_i} \leq \mathbf{b_k}$ é restrição$\}$

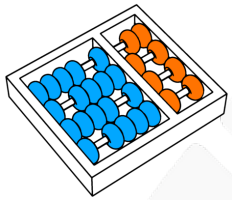
E associamos pesos:

- $w(\mathbf{v_i}, \mathbf{v_j}) = \begin{cases} b_k & \text{se } x_j - x_i \leq b_k \text{ é restrição} \\ 0 & \text{se } i = 0 \end{cases}$

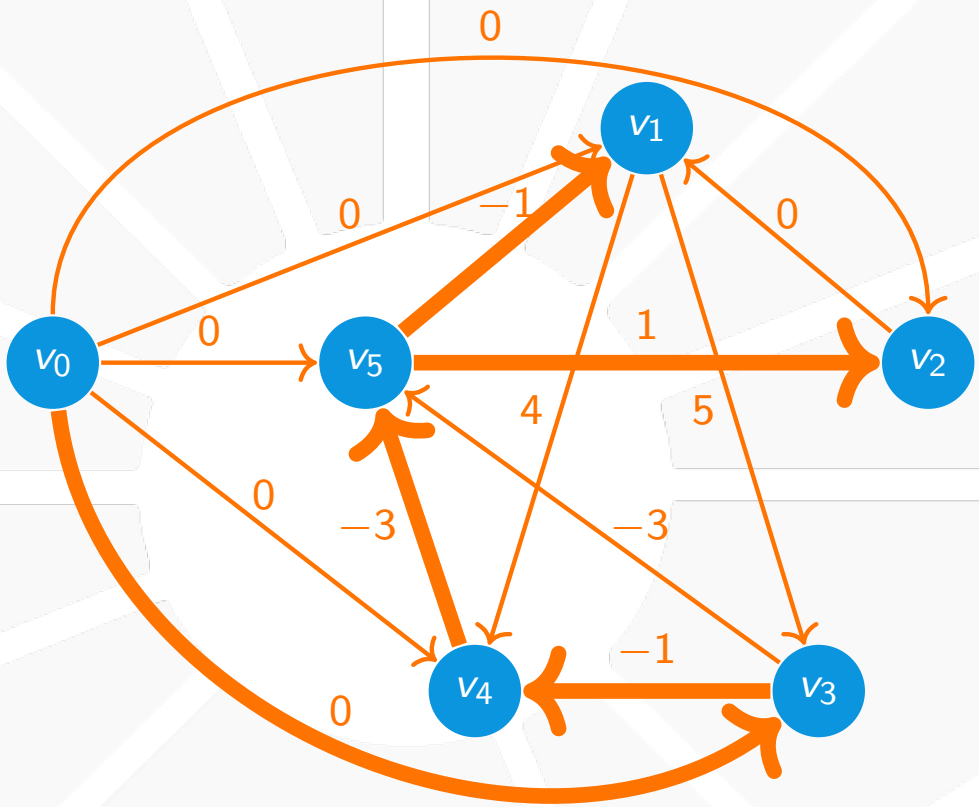


Grafo de restrições

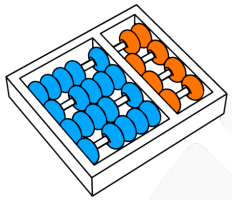




Grafo de restrições



Uma solução viável é $\mathbf{x} = (-\mathbf{5}, -\mathbf{3}, \mathbf{0}, -\mathbf{1}, -\mathbf{4})$.



Grafo de restrições

Teorema

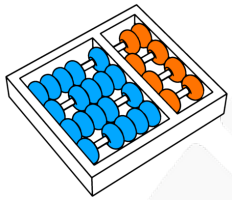
Seja $Ax \leq b$ um sistema de restrições de diferença e $G = (V, E)$ o grafo de restrições associado. Então:

- *Se G não contém ciclos negativos,*

$$x = (\text{dist}(v_0, v_1), \text{dist}(v_0, v_2), \ldots, \text{dist}(v_0, v_n))$$

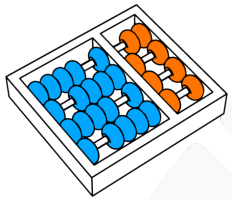
  é uma solução viável do sistema.

- *Se G contém ciclos negativos, o sistema não possui solução. viável.*



Resolvendo um sistema de restrições de diferença

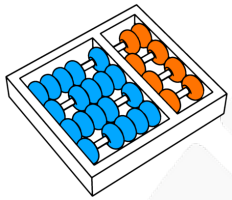
Para **RESOLVER O SISTEMA**, basta resolver caminhos mínimos:



Resolvendo um sistema de restrições de diferença

Para **RESOLVER O SISTEMA**, basta resolver caminhos mínimos:

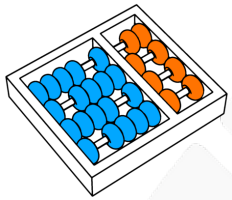
- Executamo BELLMAN-FORD a partir de v_0 no grafo de restrições G.



Resolvendo um sistema de restrições de diferença

Para **RESOLVER O SISTEMA**, basta resolver caminhos mínimos:

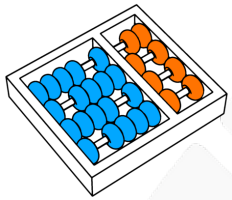
- Executamo BELLMAN-FORD a partir de v_0 no grafo de restrições G.
- Como todo vértice é atingível de v_0, se houver ciclo negativo, o algoritmo devolve FALSE.



Resolvendo um sistema de restrições de diferença

Para **RESOLVER O SISTEMA**, basta resolver caminhos mínimos:

- Executamo BELLMAN-FORD a partir de v_0 no grafo de restrições G.
- Como todo vértice é atingível de v_0, se houver ciclo negativo, o algoritmo devolve FALSE.
- Se não existir ciclo negativo, então obtemos a solução $x = (d[v_1], d[v_2], \ldots, d[v_n])$.

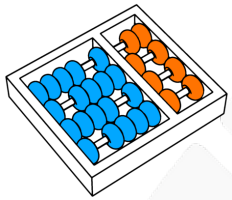


Resolvendo um sistema de restrições de diferença

Para **RESOLVER O SISTEMA**, basta resolver caminhos mínimos:

- Executamo BELLMAN-FORD a partir de $\mathbf{v_0}$ no grafo de restrições G.
- Como todo vértice é atingível de $\mathbf{v_0}$, se houver ciclo negativo, o algoritmo devolve FALSE.
- Se não existir ciclo negativo, então obtemos a solução $x = (d[\mathbf{v_1}], d[\mathbf{v_2}], \ldots, d[\mathbf{v_n}])$.

O **TEMPO DE EXECUÇÃO** é calculado como segue:



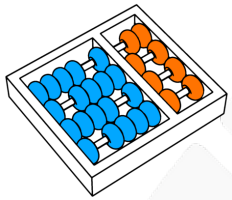
Resolvendo um sistema de restrições de diferença

Para **RESOLVER O SISTEMA**, basta resolver caminhos mínimos:

- Executamo BELLMAN-FORD a partir de $\mathbf{v_0}$ no grafo de restrições G.
- Como todo vértice é atingível de $\mathbf{v_0}$, se houver ciclo negativo, o algoritmo devolve FALSE.
- Se não existir ciclo negativo, então obtemos a solução $x = (d[\mathbf{v_1}], d[\mathbf{v_2}], \ldots, d[\mathbf{v_n}])$.

O **TEMPO DE EXECUÇÃO** é calculado como segue:

- A matriz A tem dimensões $m \times n$.



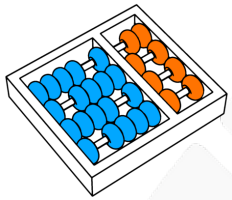
Resolvendo um sistema de restrições de diferença

Para **RESOLVER O SISTEMA**, basta resolver caminhos mínimos:

- Executamo BELLMAN-FORD a partir de $\mathbf{v_0}$ no grafo de restrições G.
- Como todo vértice é atingível de $\mathbf{v_0}$, se houver ciclo negativo, o algoritmo devolve FALSE.
- Se não existir ciclo negativo, então obtemos a solução $x = (d[\mathbf{v_1}], d[\mathbf{v_2}], \ldots, d[\mathbf{v_n}])$.

O **TEMPO DE EXECUÇÃO** é calculado como segue:

- A matriz A tem dimensões $m \times n$.
- Então, G possui $n + 1$ vértices e $n + m$ arestas.



Resolvendo um sistema de restrições de diferença

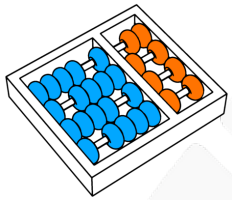
Para **RESOLVER O SISTEMA**, basta resolver caminhos mínimos:

- Executamo BELLMAN-FORD a partir de $\mathbf{v_0}$ no grafo de restrições G.
- Como todo vértice é atingível de $\mathbf{v_0}$, se houver ciclo negativo, o algoritmo devolve FALSE.
- Se não existir ciclo negativo, então obtemos a solução $x = (d[\mathbf{v_1}], d[\mathbf{v_2}], \ldots, d[\mathbf{v_n}])$.

O **TEMPO DE EXECUÇÃO** é calculado como segue:

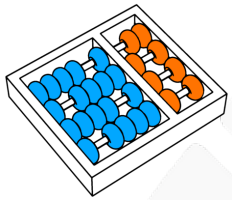
- A matriz A tem dimensões $m \times n$.
- Então, G possui $n + 1$ vértices e $n + m$ arestas.
- Logo, o tempo de execução de BELLMAN-FORD em G é $O((n + 1)(n + m)) = O(n^2 + nm)$.

Caminhos mínimos entre todos os pares de vértices



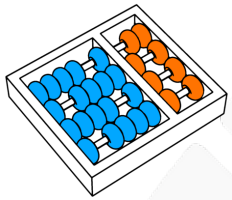
Novo problema

Dado grafo ponderado (G, w) sem ciclos negativos, queremos encontrar um caminho mínimo de **u** a **v** para **TODO** par de vértices **u**, **v**.



Algoritmos para grafos esparsos

Podemos executar $|V|$ vezes um algoritmo de Caminhos Mínimos com Mesma Origem:

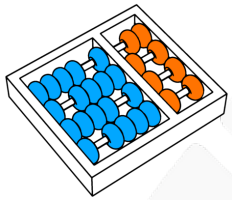


Algoritmos para grafos esparsos

Podemos executar $|V|$ vezes um algoritmo de Caminhos Mínimos com Mesma Origem:

- Se (G, w) não possui arestas negativas, usamos DIJKSTRA:

Tipo de fila	Uma vez	$\|V\|$ vezes
Heap	$O(E \log V)$	$O(VE \log V)$
Fibonacci	$O(V \log V + E)$	$O(V^2 \log V + VE)$



Algoritmos para grafos esparsos

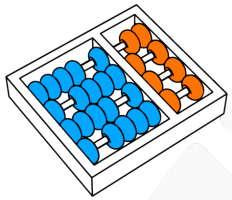
Podemos executar $|V|$ vezes um algoritmo de Caminhos Mínimos com Mesma Origem:

- Se (G, w) não possui arestas negativas, usamos Dijkstra:

Tipo de fila	Uma vez	$\|V\|$ vezes
Heap	$O(E \log V)$	$O(VE \log V)$
Fibonacci	$O(V \log V + E)$	$O(V^2 \log V + VE)$

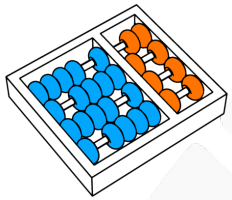
- Se (G, w) possui arestas negativas, usamos Bellman-Ford:

Uma vez	$\|V\|$ vezes
$O(VE)$	$O(V^2 E)$



O algoritmo de Floyd-Warshall

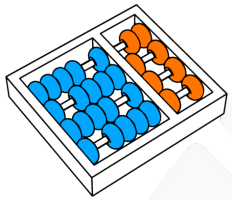
Floyd-Warshall é um algoritmo que resolve o problema diretamente e é melhor se G for **DENSO**.



O algoritmo de Floyd-Warshall

Floyd-Warshall é um algoritmo que resolve o problema diretamente e é melhor se G for **DENSO**.

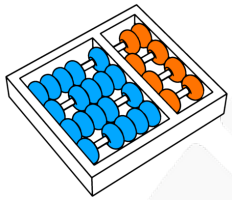
- Um grafo é denso se $|E| = \omega(V^2)$.



O algoritmo de Floyd-Warshall

Floyd-Warshall é um algoritmo que resolve o problema diretamente e é melhor se G for **DENSO**.

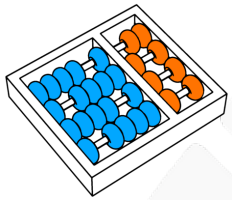
- Um grafo é denso se $|E| = \omega(V^2)$.
- É basado em programação dinâmica.



O algoritmo de Floyd-Warshall

Floyd-Warshall é um algoritmo que resolve o problema diretamente e é melhor se G for **DENSO**.

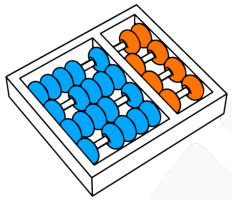
- Um grafo é denso se $|E| = \omega(V^2)$.
- É basado em programação dinâmica.
- Resolve o problema em tempo $O(V^3)$.



O algoritmo de Floyd-Warshall

Floyd-Warshall é um algoritmo que resolve o problema diretamente e é melhor se G for **DENSO**.

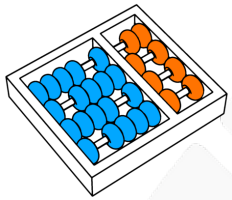
- Um grafo é denso se $|E| = \omega(V^2)$.
- É basado em programação dinâmica.
- Resolve o problema em tempo $O(V^3)$.
- Supomos que G é completo:



O algoritmo de Floyd-Warshall

Floyd-Warshall é um algoritmo que resolve o problema diretamente e é melhor se G for **DENSO**.

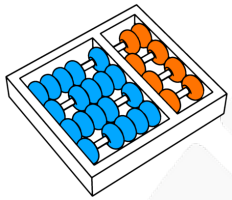
- Um grafo é denso se $|E| = \omega(V^2)$.
- É basado em programação dinâmica.
- Resolve o problema em tempo $O(V^3)$.
- Supomos que G é completo:



O algoritmo de Floyd-Warshall

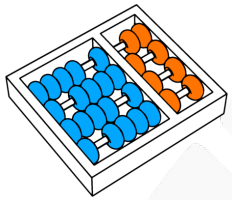
Floyd-Warshall é um algoritmo que resolve o problema diretamente e é melhor se G for **DENSO**.

- Um grafo é denso se $|E| = \omega(V^2)$.
- É basado em programação dinâmica.
- Resolve o problema em tempo $O(V^3)$.
- Supomos que G é completo:
  $\Rightarrow$ Se (i, j) não é aresta, definimos $w(i, j) = \infty$



Subproblema

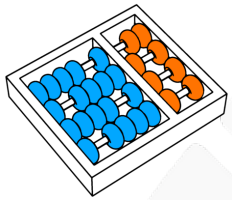
Para simplificar a discussão, suponha que $\mathbf{V = \{1, 2, \ldots, n\}}$.



Subproblema

Para simplificar a discussão, suponha que $\mathbf{V = \{1, 2, \ldots, n\}}$.

Considere um caminho $P = (\mathbf{v_1, v_2, \ldots, v_{l-1}, v_l})$:

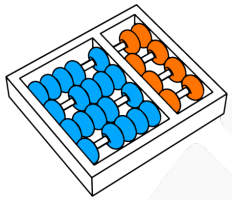


Subproblema

Para simplificar a discussão, suponha que $\mathbf{V = \{1, 2, \ldots, n\}}$.

Considere um caminho $P = (\mathbf{v_1, v_2, \ldots, v_{l-1}, v_l})$:

- Os **VÉRTICES INTERNOS** de P são $\{\mathbf{v_2, \ldots, v_{l-1}}\}$.

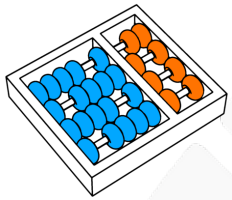


Subproblema

Para simplificar a discussão, suponha que $\mathbf{V = \{1, 2, \ldots, n\}}$.

Considere um caminho $P = (\mathbf{v_1, v_2, \ldots, v_{l-1}, v_l})$:

- Os **VÉRTICES INTERNOS** de P são $\{\mathbf{v_2, \ldots, v_{l-1}}\}$.
- P é chamado k-**INTERNO** se $\{\mathbf{v_2, \ldots, v_{l-1}}\} \subseteq \{\mathbf{1, \ldots, k}\}$.



Subproblema

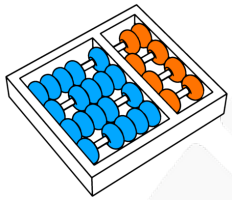
Para simplificar a discussão, suponha que $\mathbf{V = \{1, 2, \ldots, n\}}$.

Considere um caminho $P = (\mathbf{v_1, v_2, \ldots, v_{l-1}, v_l})$:

- Os **VÉRTICES INTERNOS** de P são $\{\mathbf{v_2, \ldots, v_{l-1}}\}$.
- P é chamado k-**INTERNO** se $\{\mathbf{v_2, \ldots, v_{l-1}}\} \subseteq \{\mathbf{1, \ldots, k}\}$.

Problema (Subproblema ótimo)

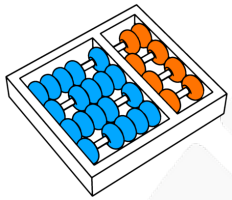
Sejam $\mathbf{i}$ *e* $\mathbf{j}$ *vértices de* G *e* k *um inteiro com* $k \geq 0$. *Dentre todos os caminhos* k*-internos de* $\mathbf{i}$ *até* $\mathbf{j}$, *encontre algum que tenha custo mínimo.*



Estrutura de um caminho mínimo

O algoritmo de Floyd-Warshall explora a relação entre:

- Um caminho k-interno P de **i** até **j** que tem custo mínimo
- e outros caminhos $(k-1)$-internos.

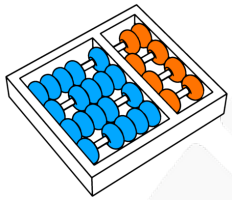


Estrutura de um caminho mínimo

O algoritmo de Floyd-Warshall explora a relação entre:

- Um caminho k-interno P de **i** até **j** que tem custo mínimo
- e outros caminhos $(k-1)$-internos.

Caso 1: Se **k** não é um vértice interno de P:



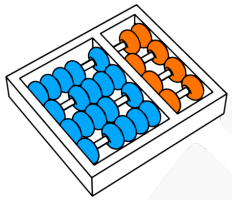
Estrutura de um caminho mínimo

O algoritmo de Floyd-Warshall explora a relação entre:

- Um caminho k-interno P de **i** até **j** que tem custo mínimo
- e outros caminhos $(k-1)$-internos.

Caso 1: Se **k** não é um vértice interno de P:

- Todos os vértices internos de P estão em $\{\mathbf{1}, \ldots, \mathbf{k-1}\}$.



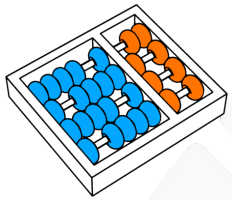
Estrutura de um caminho mínimo

O algoritmo de Floyd-Warshall explora a relação entre:

- Um caminho k-interno P de **i** até **j** que tem custo mínimo
- e outros caminhos $(k-1)$-internos.

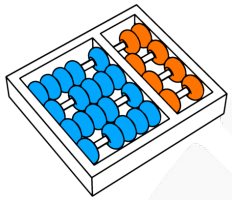
Caso 1: Se **k** não é um vértice interno de P:

- Todos os vértices internos de P estão em $\{\mathbf{1}, \ldots, \mathbf{k-1}\}$.
- Então, P é um caminho $(k-1)$-interno de custo mínimo.



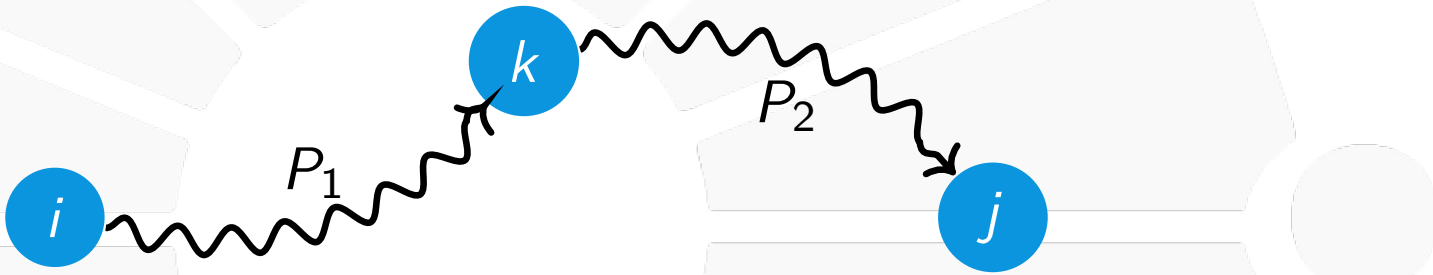
Estrutura de um caminho mínimo

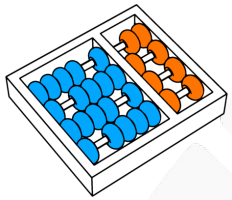
Caso 2: Se **k** é um vértice interno de P, então P pode ser dividido em dois caminhos P_1 (com início em **i** e fim em **k**) e P_2 (com início em **k** e fim em **j**).



Estrutura de um caminho mínimo

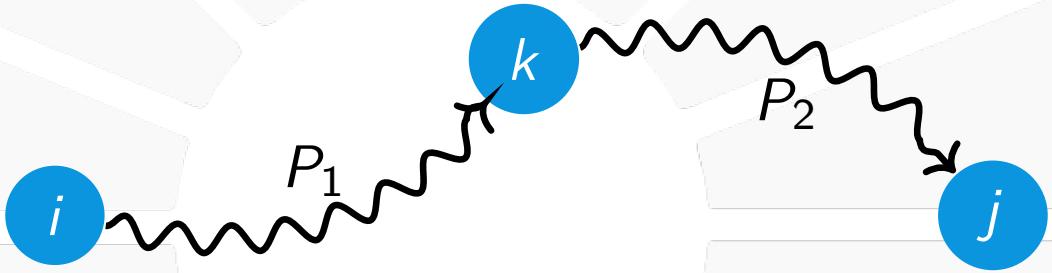
Caso 2: Se **k** é um vértice interno de P, então P pode ser dividido em dois caminhos P_1 (com início em **i** e fim em **k**) e P_2 (com início em **k** e fim em **j**).



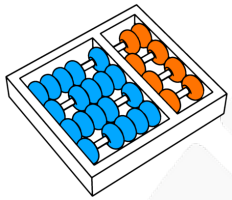


Estrutura de um caminho mínimo

Caso 2: Se **k** é um vértice interno de P, então P pode ser dividido em dois caminhos P_1 (com início em **i** e fim em **k**) e P_2 (com início em **k** e fim em **j**).

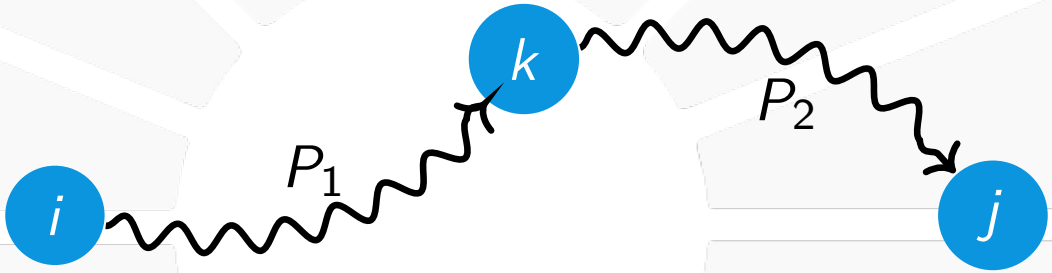


- P_1 é um caminho mínimo de **i** a **k** com vértices internos em $\{\mathbf{1}, \ldots, \mathbf{k-1}\}$.

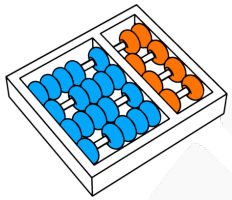


Estrutura de um caminho mínimo

Caso 2: Se **k** é um vértice interno de P, então P pode ser dividido em dois caminhos P_1 (com início em **i** e fim em **k**) e P_2 (com início em **k** e fim em **j**).

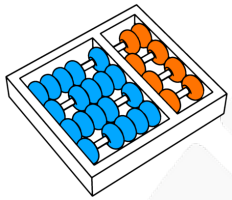


- P_1 é um caminho mínimo de **i** a **k** com vértices internos em $\{\mathbf{1}, \ldots, \mathbf{k-1}\}$.
- P_2 é um caminho mínimo de **k** a **j** com vértices internos em $\{\mathbf{1}, \ldots, \mathbf{k-1}\}$.



Recorrência para caminhos mínimos

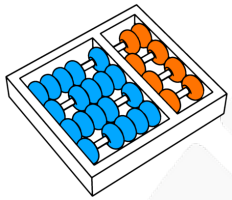
Seja $d_{ij}^{(k)}$ o peso de um caminho k-interno mínimo de **i** a **j**.



Recorrência para caminhos mínimos

Seja $d_{ij}^{(k)}$ o peso de um caminho k-interno mínimo de **i** a **j**.

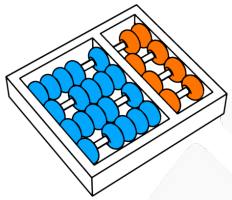
- Se $k = 0$ então $d_{ij}^{(0)} = w(i, j)$.



Recorrência para caminhos mínimos

Seja $d_{ij}^{(k)}$ o peso de um caminho k-interno mínimo de **i** a **j**.

- Se $k = 0$ então $d_{ij}^{(0)} = w(i, j)$.
- Senão, caímos em algum dos dois casos anteriores.



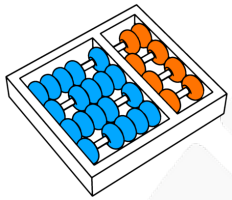
Recorrência para caminhos mínimos

Seja $d_{ij}^{(k)}$ o peso de um caminho k-interno mínimo de **i** a **j**.

- Se $k = 0$ então $d_{ij}^{(0)} = w(i,j)$.
- Senão, caímos em algum dos dois casos anteriores.

Obtemos a seguinte recorrência:

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



Recorrência para caminhos mínimos

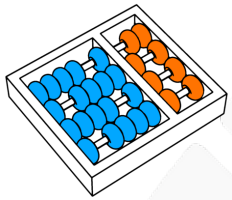
Seja $d_{ij}^{(k)}$ o peso de um caminho k-interno mínimo de **i** a **j**.

- Se $k = 0$ então $d_{ij}^{(0)} = w(i,j)$.
- Senão, caímos em algum dos dois casos anteriores.

Obtemos a seguinte recorrência:

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

Note que, $d_{ij}^{(n)} = \text{dist}(i,j)$.



Recorrência para caminhos mínimos

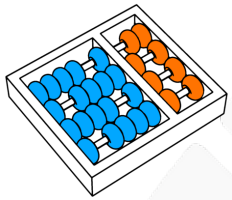
Seja $d_{ij}^{(k)}$ o peso de um caminho k-interno mínimo de **i** a **j**.

- Se $k = 0$ então $d_{ij}^{(0)} = w(i,j)$.
- Senão, caímos em algum dos dois casos anteriores.

Obtemos a seguinte recorrência:

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

Note que, $d_{ij}^{(n)} = \text{dist}(i,j)$. Por quê?



Recorrência para caminhos mínimos

Seja $d_{ij}^{(k)}$ o peso de um caminho k-interno mínimo de **i** a **j**.

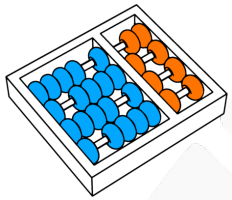
- Se $k = 0$ então $d_{ij}^{(0)} = w(i,j)$.
- Senão, caímos em algum dos dois casos anteriores.

Obtemos a seguinte recorrência:

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

Note que, $d_{ij}^{(n)} = \text{dist}(i,j)$. Por quê?

- Calculamos as matrizes $D^{(k)} = (d_{ij}^{(k)})$ para $k = 1, 2 \ldots, n$.



Recorrência para caminhos mínimos

Seja $d_{ij}^{(k)}$ o peso de um caminho k-interno mínimo de **i** a **j**.

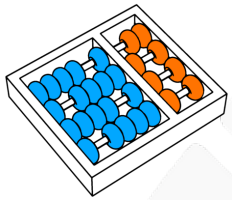
- Se $k = 0$ então $d_{ij}^{(0)} = w(i,j)$.
- Senão, caímos em algum dos dois casos anteriores.

Obtemos a seguinte recorrência:

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

Note que, $d_{ij}^{(n)} = \text{dist}(i,j)$. Por quê?

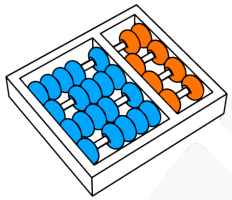
- Calculamos as matrizes $D^{(k)} = (d_{ij}^{(k)})$ para $k = 1, 2 \ldots, n$.
- A resposta do problema é $D^{(n)}$.



Algoritmo de Floyd-Warshall

Problema

Entrada: *Matriz* $W = (w(i,j))$ *com dimensões* $n \times n$.
Saída: *Matriz* $D^{(n)}$.



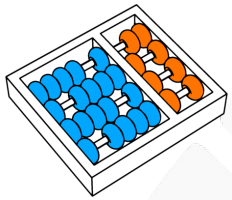
Algoritmo de Floyd-Warshall

Problema

Entrada: *Matriz* $W = (w(i,j))$ *com dimensões* $n \times n$.
Saída: *Matriz* $D^{(n)}$.

Algoritmo 8: FLOYD-WARSHALL(W, n)

```
1 D^(0) ← W
2 para k ← 1 até n
3     para i ← 1 até n
4         para j ← 1 até n
5             d_ij^(k) ← min(d_ij^(k−1), d_ik^(k−1) + d_kj^(k−1))
6 devolva D^(n)
```



Algoritmo de Floyd-Warshall

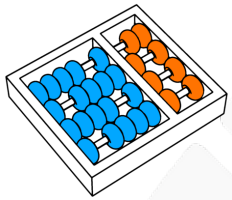
Problema

Entrada: *Matriz* $W = (w(i,j))$ *com dimensões* $n \times n$.
Saída: *Matriz* $D^{(n)}$.

Algoritmo 9: FLOYD-WARSHALL(W, n)

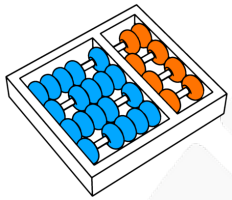
```
1 D^(0) ← W
2 para k ← 1 até n
3     para i ← 1 até n
4         para j ← 1 até n
5             d_ij^(k) ← min(d_ij^(k−1), d_ik^(k−1) + d_kj^(k−1))
6 devolva D^(n)
```

Complexidade: $O(V^3)$.



Como encontrar os caminhos?

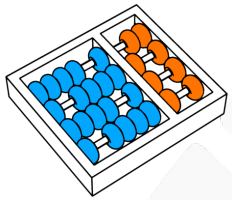
Devolvemos matriz de predecessores $\Pi = (\pi_{ij})$



Como encontrar os caminhos?

Devolvemos matriz de predecessores $\Pi = (\pi_{ij})$

(a) $\pi_{ij} = \text{NIL}$ se $\mathbf{i} = \mathbf{j}$ ou se não existe caminho de **i** a **j**,

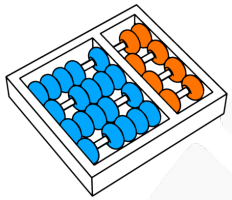


Como encontrar os caminhos?

Devolvemos matriz de predecessores $\Pi = (\pi_{ij})$

(a) $\pi_{ij} = \text{NIL}$ se **i** $=$ **j** ou se não existe caminho de **i** a **j**,

(b) π_{ij} é o **PREDECESSOR** de **j** em algum caminho mínimo a partir de **i**, caso contrário.



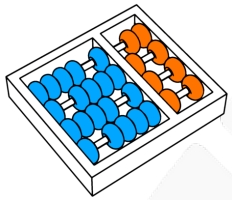
Como encontrar os caminhos?

Devolvemos matriz de predecessores $\Pi = (\pi_{ij})$

(a) $\pi_{ij} = \text{NIL}$ se **i** $=$ **j** ou se não existe caminho de **i** a **j**,

(b) π_{ij} é o **PREDECESSOR** de **j** em algum caminho mínimo a partir de **i**, caso contrário.

- Calculamos do mesmo modo que $D^{(k)}$.
- Obtemos uma sequência de matrizes $\Pi^{(0)}, \Pi^{(1)}, \ldots, \Pi^{(n)}$



Como encontrar os caminhos?

Devolvemos matriz de predecessores $\Pi = (\pi_{ij})$

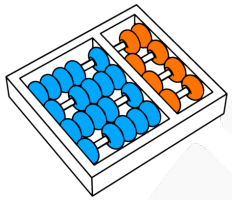
(a) $\pi_{ij} = \text{NIL}$ se **i** $=$ **j** ou se não existe caminho de **i** a **j**,

(b) π_{ij} é o **PREDECESSOR** de **j** em algum caminho mínimo a partir de **i**, caso contrário.

- Calculamos do mesmo modo que $D^{(k)}$.
- Obtemos uma sequência de matrizes $\Pi^{(0)}, \Pi^{(1)}, \ldots, \Pi^{(n)}$

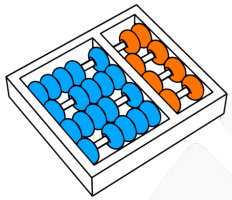
Quando $k = 0$:

$$\pi_{ij}^{(0)} = \begin{cases} \text{NIL} & \text{se } i = j \text{ ou } w(i,j) = \infty, \\ i & \text{se } i \neq j \text{ e } w(i,j) < \infty. \end{cases}$$



Como encontrar os caminhos?

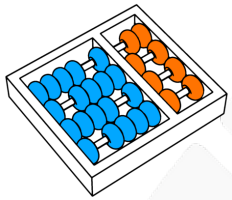
Se $k \geq 1$:



Como encontrar os caminhos?

Se $k \geq 1$:

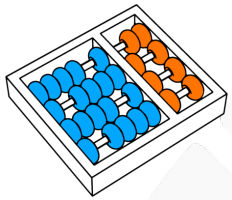
- Considere um caminho k-interno mínimo P de **i** a **j**.



Como encontrar os caminhos?

Se $k \geq 1$:

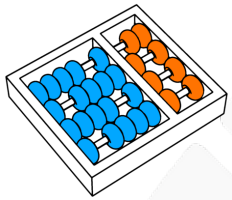
- Considere um caminho k-interno mínimo P de **i** a **j**.
- Obtemos o **PREDECESSOR DE j**:
  - Se k não aparece em P, então usamos o predecessor de um caminho $(k-1)$-interno de **i** a **j**.



Como encontrar os caminhos?

Se $k \geq 1$:

- Considere um caminho k-interno mínimo P de **i** a **j**.
- Obtemos o **PREDECESSOR DE j**:
  - Se k não aparece em P, então usamos o predecessor de um caminho $(k-1)$-interno de **i** a **j**.
  - Se k aparece em P, então usamos o predecessor de um caminho $(k-1)$-interno de **k** a **j**.



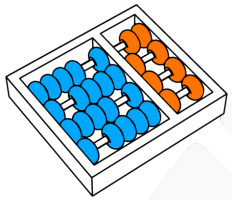
Como encontrar os caminhos?

Se $k \geq 1$:

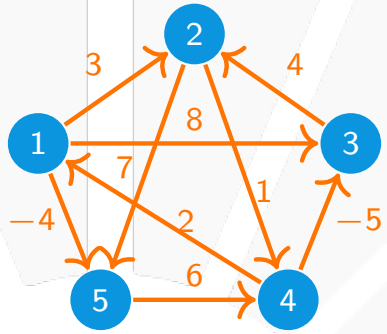
- Considere um caminho k-interno mínimo P de **i** a **j**.
- Obtemos o **PREDECESSOR DE j**:
  - Se k não aparece em P, então usamos o predecessor de um caminho $(k-1)$-interno de **i** a **j**.
  - Se k aparece em P, então usamos o predecessor de um caminho $(k-1)$-interno de **k** a **j**.

Formalmente,

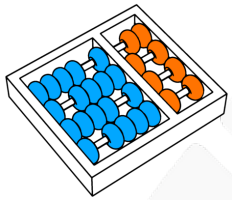
$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{se } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, \\ \pi_{kj}^{(k-1)} & \text{se } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}. \end{cases}$$



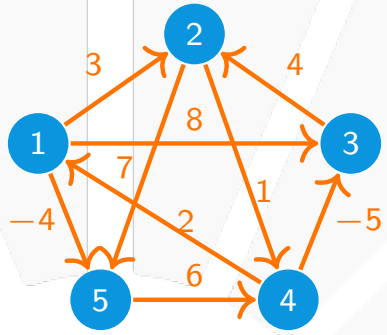
Exemplo



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

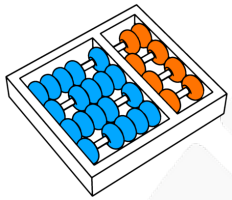


Exemplo

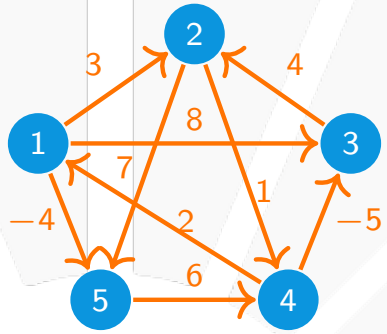


$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(0)} = \begin{pmatrix} N & 1 & 1 & N & 1 \\ N & N & N & 2 & 2 \\ N & 3 & N & N & N \\ 4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

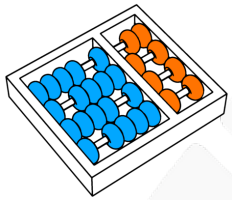
$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \mathbf{5} & -5 & 0 & \mathbf{-2} \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(1)} = \begin{pmatrix} N & 1 & 1 & N & 1 \\ N & N & N & 2 & 2 \\ N & 3 & N & N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



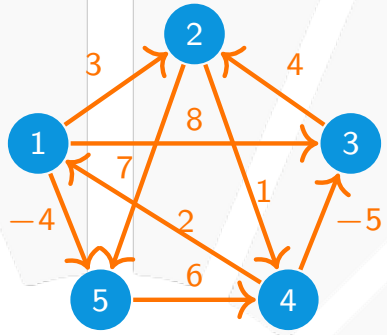
Exemplo



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

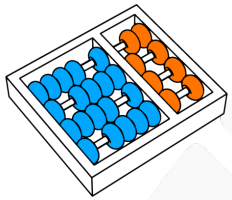


Exemplo

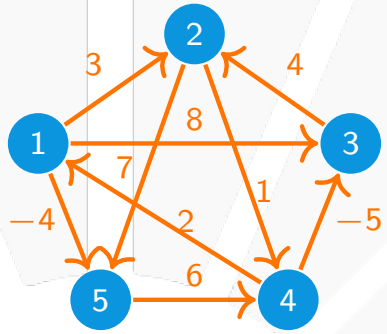


$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(1)} = \begin{pmatrix} N & 1 & 1 & N & 1 \\ N & N & N & 2 & 2 \\ N & 3 & N & N & N \\ 4 & 1 & 4 &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

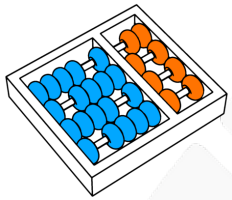
$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & \mathbf{4} & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \mathbf{5} & \mathbf{11} \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(2)} = \begin{pmatrix} N & 1 & 1 & \mathbf{2} & 1 \\ N & N & N & 2 & 2 \\ N & 3 & N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



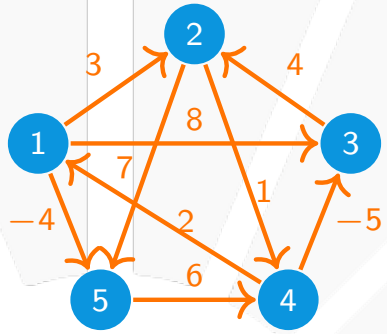
Exemplo



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

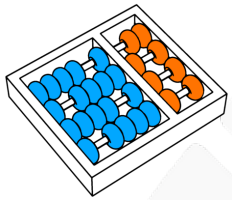


Exemplo

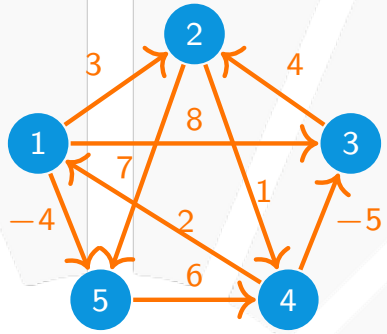


$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(2)} = \begin{pmatrix} N & 1 & 1 & 4 & 1 \\ N & N & N & 2 & 2 \\ N & 3 & N & 2 & 2 \\ 4 & 1 & 4 & N & 1 \\ N &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

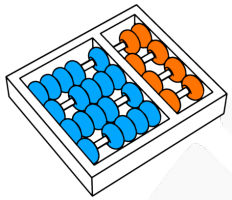
$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & \mathbf{-1} & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(3)} = \begin{pmatrix} N & 1 & 1 & 2 & 1 \\ N & N & N & 2 & 2 \\ N & 3 & N & 2 & 2 \\ 4 & \mathbf{3}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



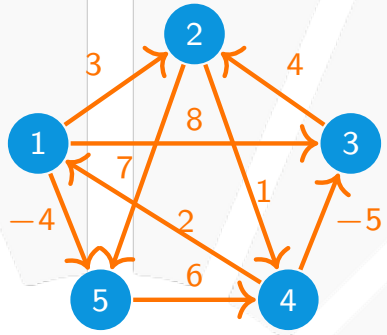
Exemplo



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

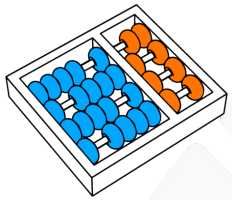


Exemplo

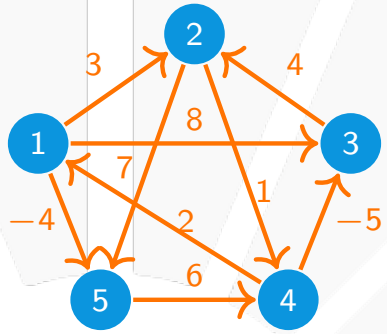


$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(3)} = \begin{pmatrix} N & 1 & 1 & 2 & 1 \\ N & N & N & 2 & 2 \\ N & 3 & N & 2 & 2 \\ 4 & 3 & 4 & N & 1 \\ N &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

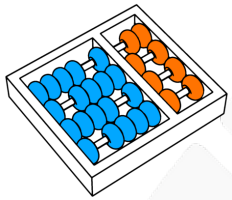
$$D^{(4)} = \begin{pmatrix} 0 & 3 & \mathbf{-1} & 4 & -4 \\ \mathbf{3} & 0 & \mathbf{-4} & 1 & \mathbf{-1} \\ \mathbf{7} & 4 & 0 & 5 & \mathbf{3} \\ 2 & -1 & -5 & 0 & -2 \\ \mathbf{8} & \mathbf{5} & \mathbf{1} & 6 & 0 \end{pmatrix} \qquad \Pi^{(4)} = \begin{pmatrix} N & 1 & \mathbf{4} & 2 & 1 \\ \mathbf{4}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



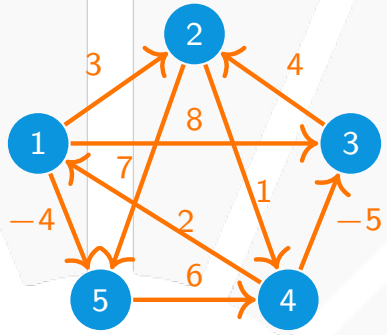
Exemplo



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



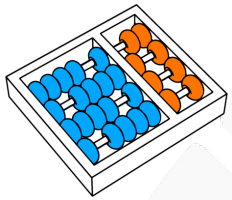
Exemplo



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

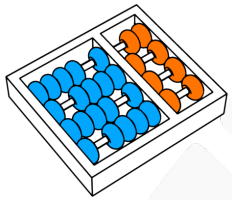
$$D^{(5)} = \begin{pmatrix} 0 & \mathbf{1} & \mathbf{-3} & \mathbf{2} & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \qquad \Pi^{(5)} = \begin{pmatrix} N & \mathbf{3} & \mathbf{4} & \mathbf{5} & 1 \\ 4 & N & 4 & 2 & 1 \\ 4 & 3 & N & 2 & 1 \\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

Fecho transitivo de grafos direcionados



Fecho transitivo de grafos direcionados

Seja $G = (\mathbf{V}, \mathbf{E})$ um grafo direcionado com $\mathbf{V} = \{\mathbf{1}, \mathbf{2}, \ldots, \mathbf{n}\}$.

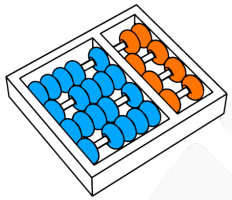


Fecho transitivo de grafos direcionados

Seja $G = (\mathbf{V}, \mathbf{E})$ um grafo direcionado com $\mathbf{V} = \{\mathbf{1}, \mathbf{2}, \ldots, \mathbf{n}\}$.

O **FECHO TRANSITIVO** de $G = (\mathbf{V}, \mathbf{E})$ é o grafo $G^* = (\mathbf{V}, \mathbf{E}^*)$ onde:

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

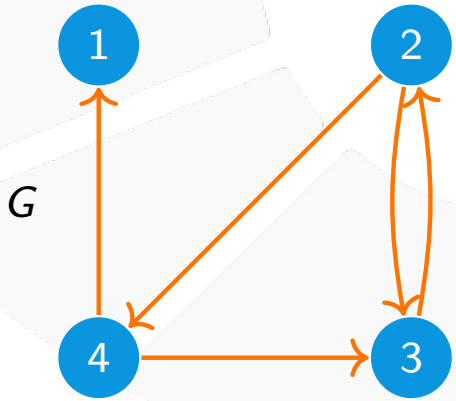


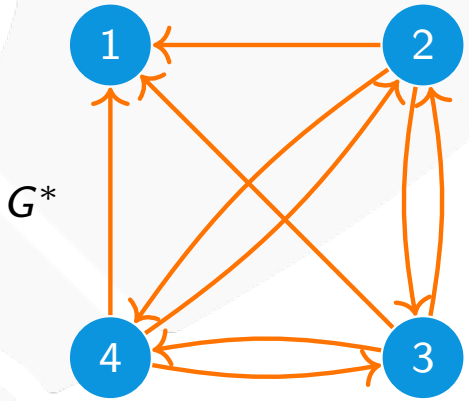
Fecho transitivo de grafos direcionados

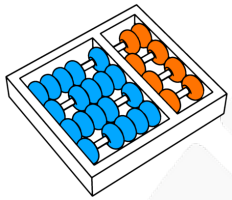
Seja $G = (\mathbf{V}, \mathbf{E})$ um grafo direcionado com $\mathbf{V} = \{\mathbf{1}, \mathbf{2}, \ldots, \mathbf{n}\}$.

O **FECHO TRANSITIVO** de $G = (\mathbf{V}, \mathbf{E})$ é o grafo $G^* = (\mathbf{V}, \mathbf{E}^*)$ onde:

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

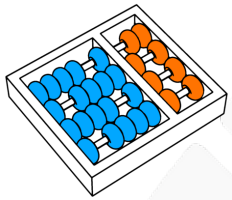






Um detalhe de implementação

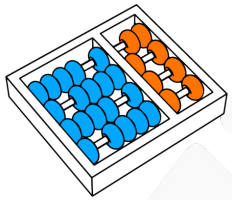
Determinando o fecho transitivo de $G = (\mathbf{V}, \mathbf{E})$:



Um detalhe de implementação

Determinando o fecho transitivo de $G = (\mathbf{V}, \mathbf{E})$:

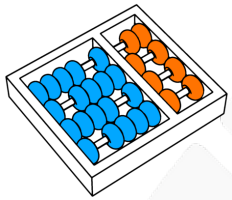
1. Atribuímos peso 1 a cada aresta.



Um detalhe de implementação

Determinando o fecho transitivo de $G = (\mathbf{V}, \mathbf{E})$:

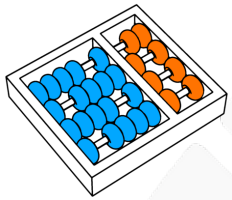
1. Atribuímos peso 1 a cada aresta.
2. Executamos FLOYD-WARSHALL em tempo $\Theta(V^3)$.



Um detalhe de implementação

Determinando o fecho transitivo de $G = (\mathbf{V}, \mathbf{E})$:

1. Atribuímos peso 1 a cada aresta.
2. Executamos FLOYD-WARSHALL em tempo $\Theta(V^3)$.
3. Existe um caminho de **i** a **j** se e somente se $d_{ij} < |V|$.

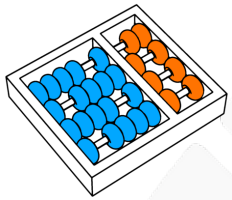


Um detalhe de implementação

Determinando o fecho transitivo de $G = (\mathbf{V}, \mathbf{E})$:

1. Atribuímos peso 1 a cada aresta.
2. Executamos FLOYD-WARSHALL em tempo $\Theta(V^3)$.
3. Existe um caminho de **i** a **j** se e somente se $d_{ij} < |V|$.

Na prática:



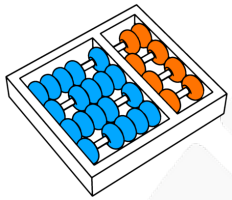
Um detalhe de implementação

Determinando o fecho transitivo de $G = (\mathbf{V}, \mathbf{E})$:

1. Atribuímos peso 1 a cada aresta.
2. Executamos FLOYD-WARSHALL em tempo $\Theta(V^3)$.
3. Existe um caminho de **i** a **j** se e somente se $d_{ij} < |V|$.

Na prática:

- Executamos FLOYD-WARSHALL substituindo:
  - min por $\vee$ (OU lógico).
  - $+$ por $\wedge$ (E lógico).



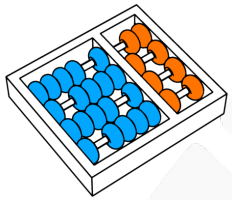
Um detalhe de implementação

Determinando o fecho transitivo de $G = (\mathbf{V}, \mathbf{E})$:

1. Atribuímos peso 1 a cada aresta.
2. Executamos FLOYD-WARSHALL em tempo $\Theta(V^3)$.
3. Existe um caminho de **i** a **j** se e somente se $d_{ij} < |V|$.

Na prática:

- Executamos FLOYD-WARSHALL substituindo:
  - min por $\vee$ (OU lógico).
  - $+$ por $\wedge$ (E lógico).
- Tem a mesma complexidade assintótica.



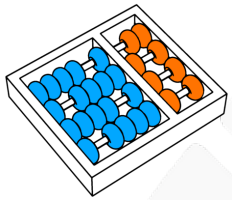
Um detalhe de implementação

Determinando o fecho transitivo de $G = (\mathbf{V}, \mathbf{E})$:

1. Atribuímos peso 1 a cada aresta.
2. Executamos FLOYD-WARSHALL em tempo $\Theta(V^3)$.
3. Existe um caminho de **i** a **j** se e somente se $d_{ij} < |V|$.

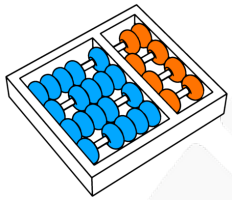
Na prática:

- Executamos FLOYD-WARSHALL substituindo:
  - min por $\vee$ (OU lógico).
  - $+$ por $\wedge$ (E lógico).
- Tem a mesma complexidade assintótica.
- Economiza tempo e espaço.



Fecho transitivo de grafos direcionados

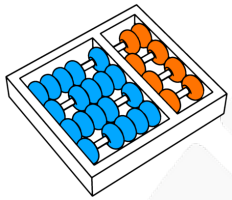
Defina $t_{i,j}^{(k)}$ o valor booleano:



Fecho transitivo de grafos direcionados

Defina $t_{i,j}^{(k)}$ o valor booleano:

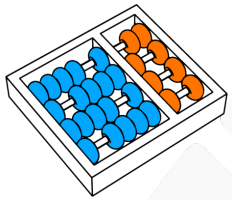
- TRUE se existe caminho k-interno de **i** a **j**.



Fecho transitivo de grafos direcionados

Defina $t_{i,j}^{(k)}$ o valor booleano:

- TRUE se existe caminho k-interno de **i** a **j**.
- FALSE se **NÃO** existe caminho k-interno de **i** a **j**.



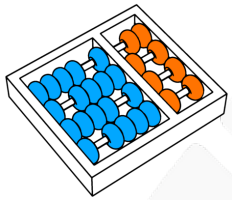
Fecho transitivo de grafos direcionados

Defina $t_{i,j}^{(k)}$ o valor booleano:

- TRUE se existe caminho k-interno de **i** a **j**.
- FALSE se **NÃO** existe caminho k-interno de **i** a **j**.

Para $k = 0$

$$t_{ij}^{(0)} = \begin{cases} 0 & \text{se } i \neq j \text{ e } (i,j) \notin E, \\ 1 & \text{se } i = j \text{ ou } (i,j) \in E, \end{cases}$$



Fecho transitivo de grafos direcionados

Defina $t_{i,j}^{(k)}$ o valor booleano:

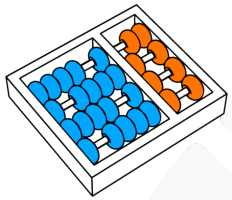
- TRUE se existe caminho k-interno de **i** a **j**.
- FALSE se **NÃO** existe caminho k-interno de **i** a **j**.

Para $k = 0$

$$t_{ij}^{(0)} = \begin{cases} 0 & \text{se } i \neq j \text{ e } (i,j) \notin E, \\ 1 & \text{se } i = j \text{ ou } (i,j) \in E, \end{cases}$$

e para $k \geq 1$,

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)} \right).$$



Fecho transitivo de grafos direcionados

Defina $t_{i,j}^{(k)}$ o valor booleano:

- TRUE se existe caminho k-interno de **i** a **j**.
- FALSE se **NÃO** existe caminho k-interno de **i** a **j**.

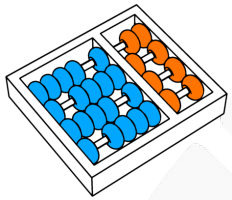
Para $k = 0$

$$t_{ij}^{(0)} = \begin{cases} 0 & \text{se } i \neq j \text{ e } (i,j) \notin E, \\ 1 & \text{se } i = j \text{ ou } (i,j) \in E, \end{cases}$$

e para $k \geq 1$,

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)} \right).$$

Como em FLOYD-WARSHALL, calculamos matrizes $T^{(k)} = (t_{ij}^{(k)})$.



Algoritmo de Transitive-Closure

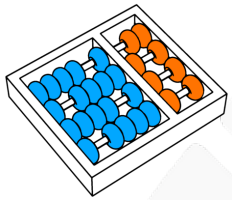
Problema

Entrada: *Matriz de adjacência A de G.*
Saída: *Matriz de adjacência $T^{(n)}$ de G^*.*

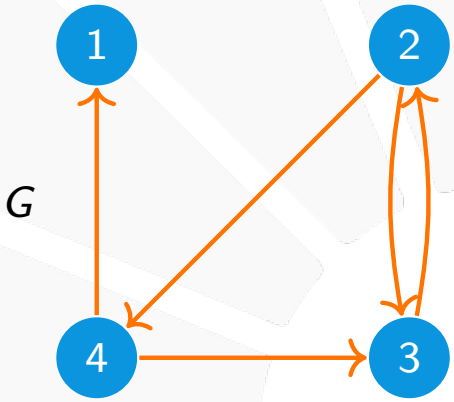
Algoritmo 10: TRANSITIVE-CLOSURE(A, n)

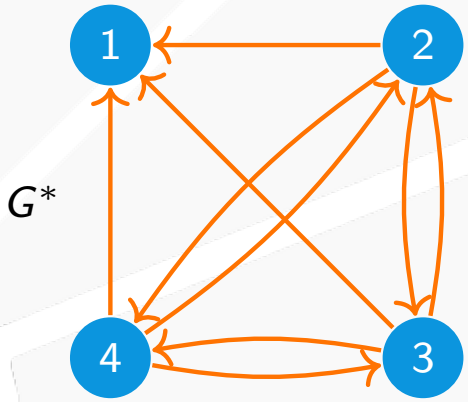
```
1 T^(0) ← A + I_n
2 para k ← 1 até n
3     para i ← 1 até n
4         para j ← 1 até n
5             t_ij^(k) ← t_ij^(k−1) ∨ (t_ik^(k−1) + t_kj^(k−1))
6 devolva T^(n)
```

Complexidade: $O(V^3)$.

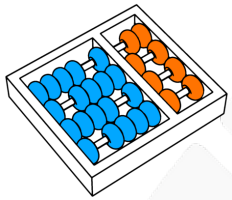


Exemplo



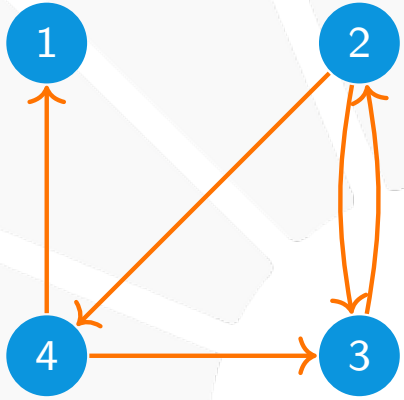


$$T^{(0)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

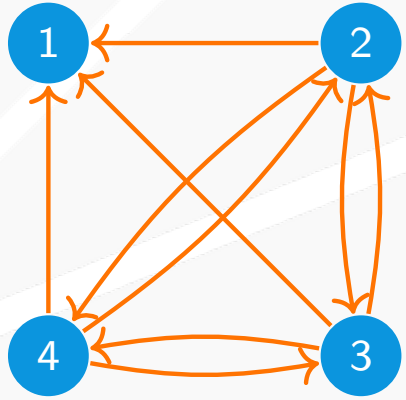


Exemplo

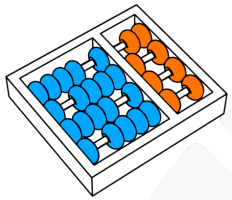
G



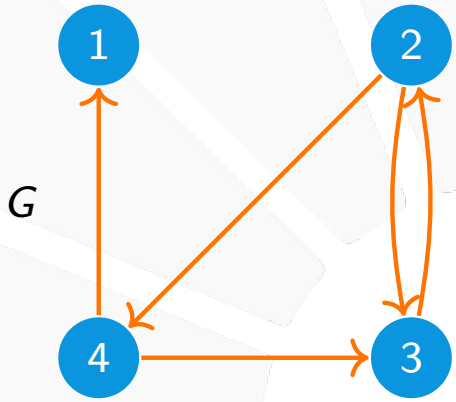
G^*

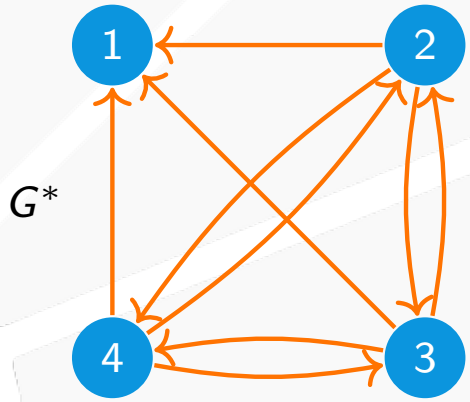


$$T^{(0)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \qquad T^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

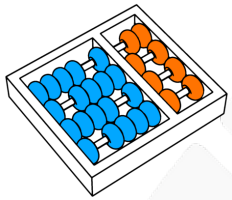


Exemplo



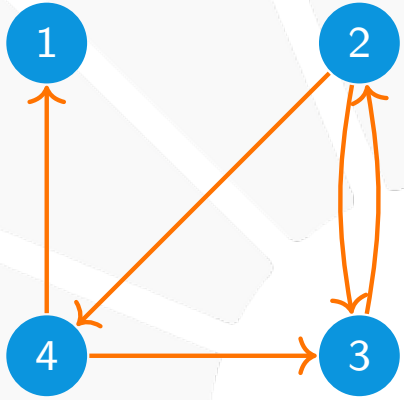


$$T^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

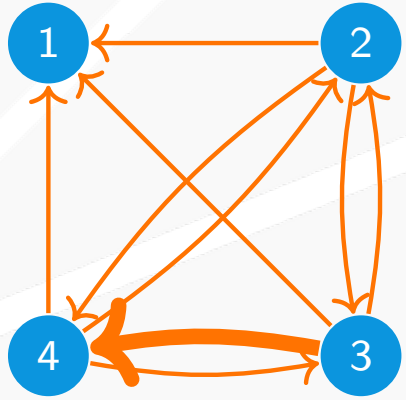


Exemplo

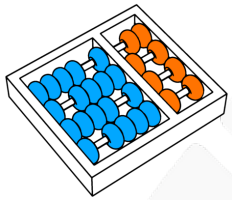
G



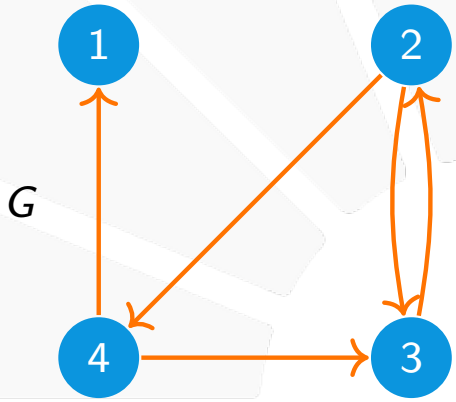
G^*

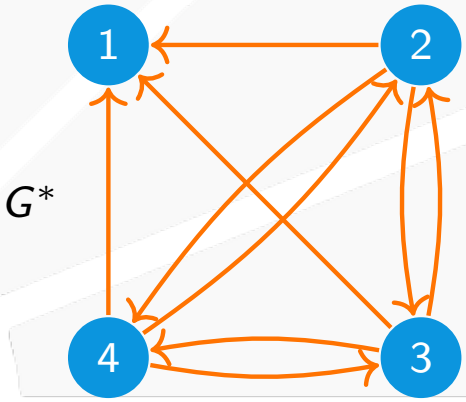


$$T^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \qquad T^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & \mathbf{1} \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

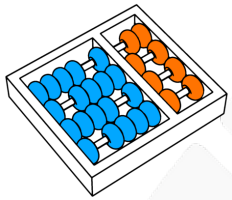


Exemplo

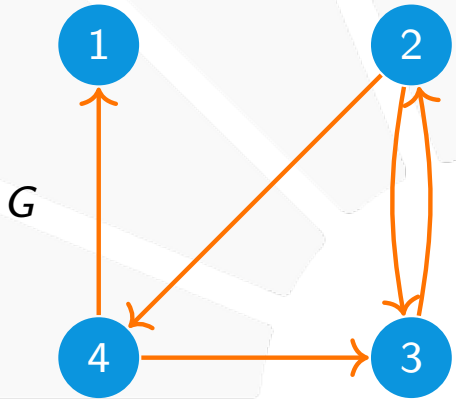


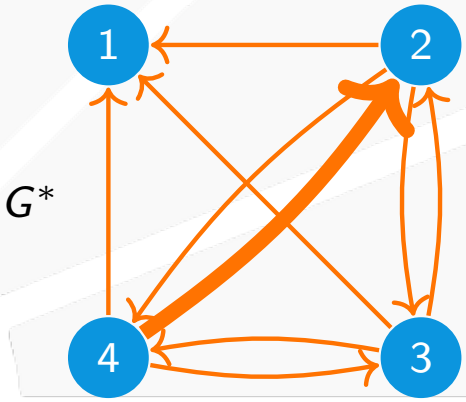


$$T^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

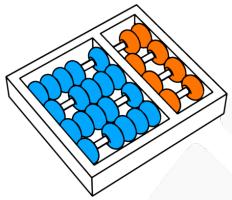


Exemplo

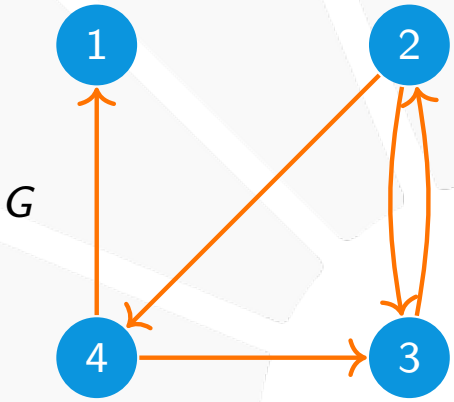


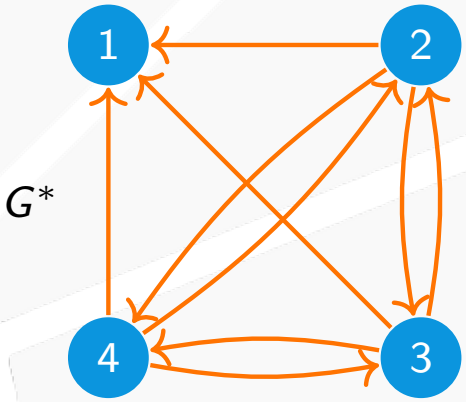


$$T^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix} \qquad T^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & \mathbf{1} & 1 & 1 \end{pmatrix}$$

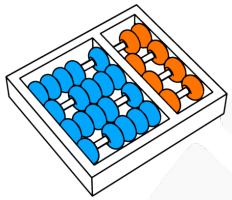


Exemplo



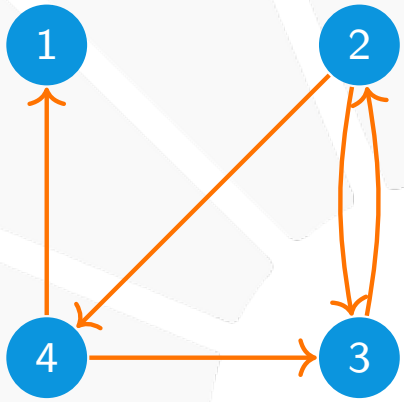


$$T^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

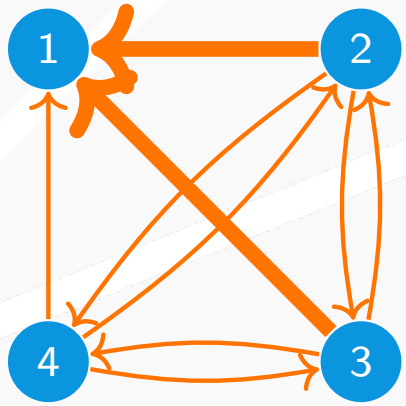


Exemplo

G



G^*



$$T^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \qquad T^{(4)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ \mathbf{1} & 1 & 1 & 1 \\ \mathbf{1} & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Caminhos mínimos

MC558 - Projeto e Análise de Algoritmos II

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

UNICAMP