Caminhos mínimos

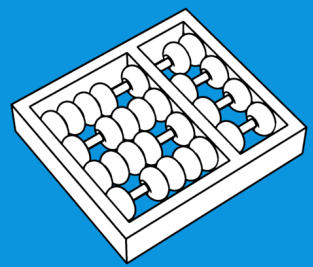
MC558 - Projeto e Análise de Algoritmos II

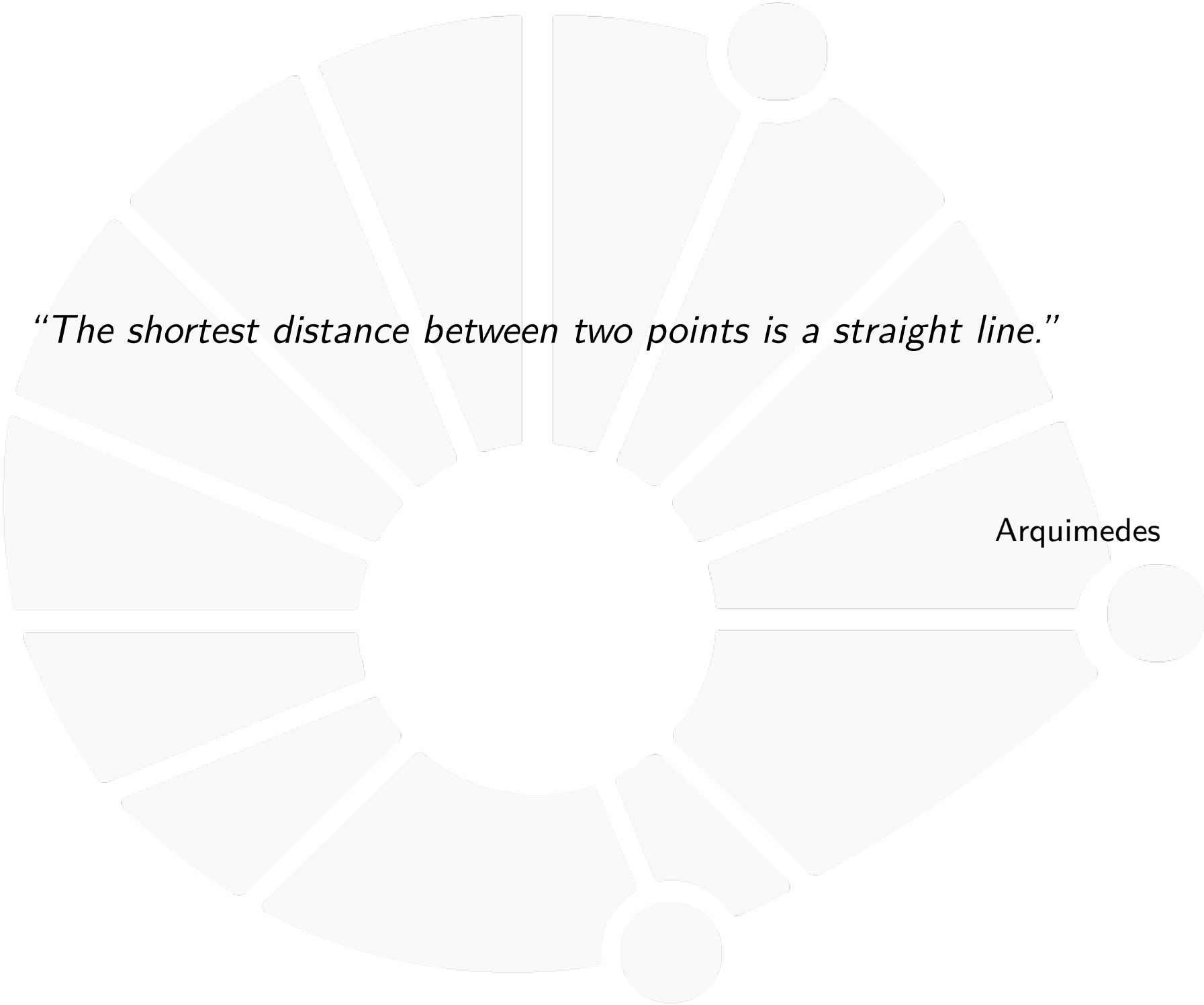
01/24

6

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

UNICAMP

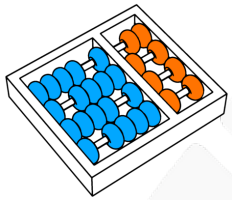




"The shortest distance between two points is a straight line."

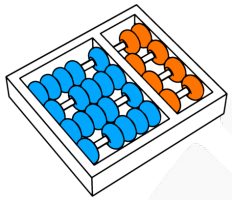
Arquimedes

Caminhos mínimos com uma origem



Problema do Caminho Mínimo

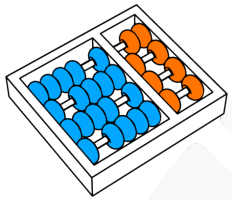
Considere um par (G, w) em que:



Problema do Caminho Mínimo

Considere um par (G, w) em que:

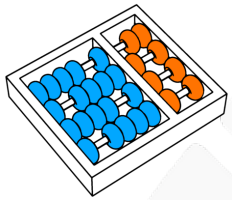
- G é um grafo direcionado.



Problema do Caminho Mínimo

Considere um par (G, w) em que:

- G é um grafo direcionado.
- w associa um peso $w(u, v)$ para cada aresta $(\mathbf{u}, \mathbf{v})$.

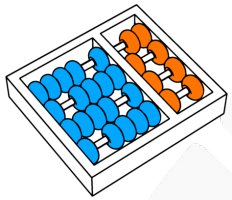


Problema do Caminho Mínimo

Considere um par (G, w) em que:

- G é um grafo direcionado.
- w associa um peso $w(u, v)$ para cada aresta $(\mathbf{u}, \mathbf{v})$.

Estamos interessados nos seguintes problemas:



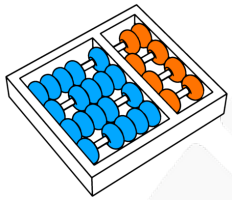
Problema do Caminho Mínimo

Considere um par (G, w) em que:

- G é um grafo direcionado.
- w associa um peso $w(u, v)$ para cada aresta $(\mathbf{u}, \mathbf{v})$.

Estamos interessados nos seguintes problemas:

1. **Problema do caminho mínimo entre dois vértices:**
   Dados **s** e **t**, encontrar um caminho de peso mínimo de **s** a **t**.



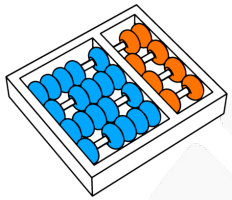
Problema do Caminho Mínimo

Considere um par (G, w) em que:

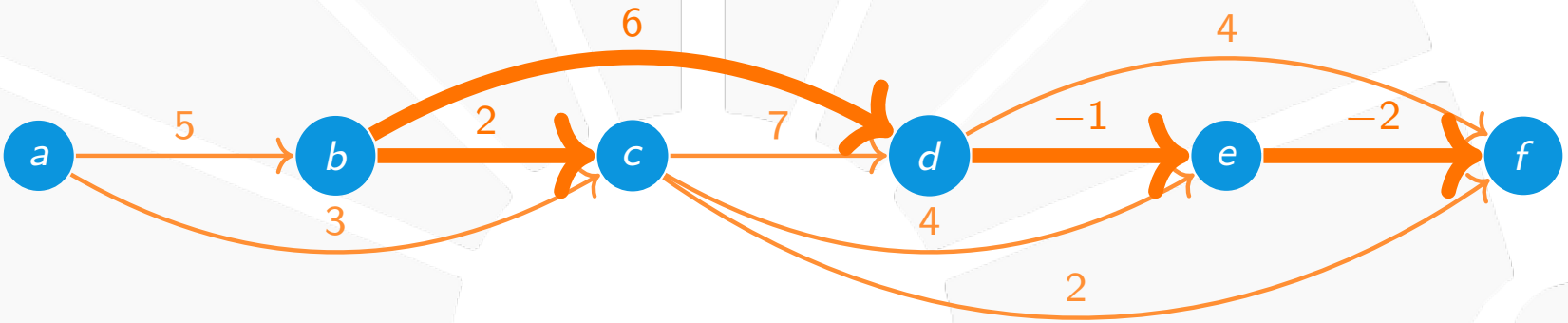
- G é um grafo direcionado.
- w associa um peso $w(u, v)$ para cada aresta $(\mathbf{u}, \mathbf{v})$.

Estamos interessados nos seguintes problemas:

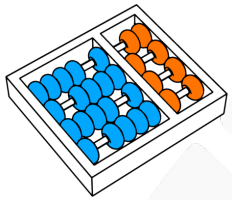
1. **Problema do caminho mínimo entre dois vértices:**
   Dados **s** e **t**, encontrar um caminho de peso mínimo de **s** a **t**.
2. **Problema dos caminhos mínimos com mesma origem:**
   Dado **s**, encontrar um caminho de peso mínimo de **s** a **v** para **TODO** vértice **v** de G.



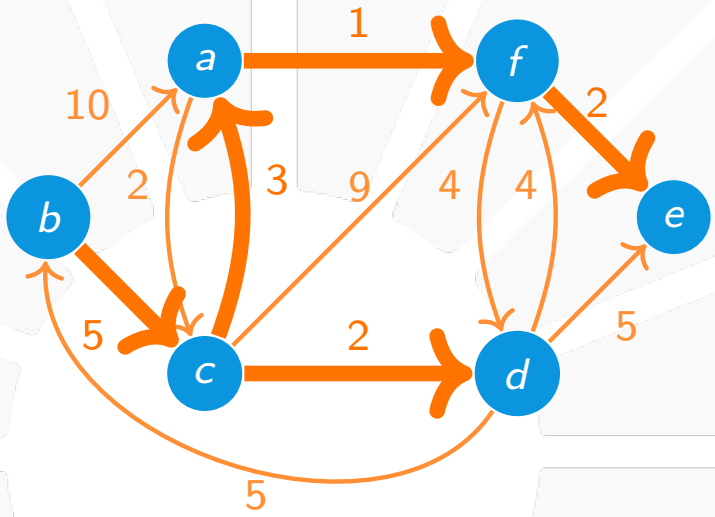
Exemplo: grafo direcionado acíclico



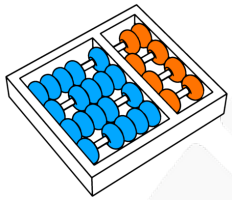
v	**a**	**b**	**c**	**d**	**e**	**f**
dist(**b**, v)	∞	0	2	6	5	3



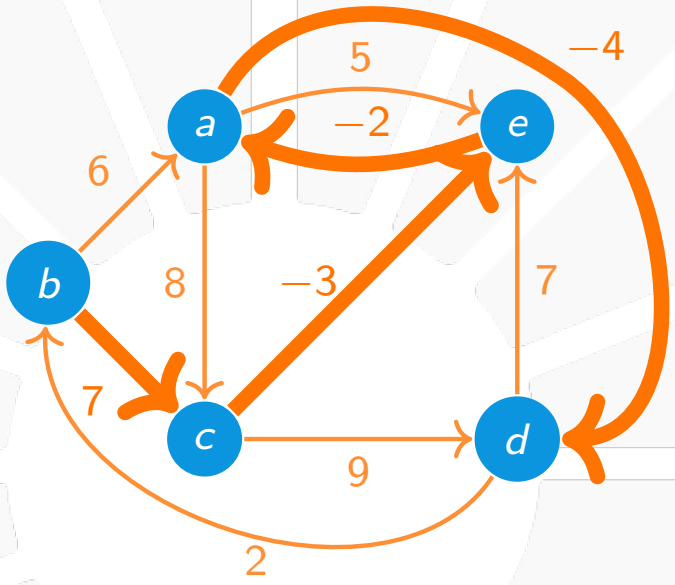
Exemplo: grafo direcionado sem arestas negativas



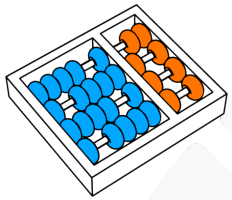
v	a	b	c	d	e	f
dist(b, v)	8	0	5	7	11	9



Exemplo: grafo direcionado com arestas negativas

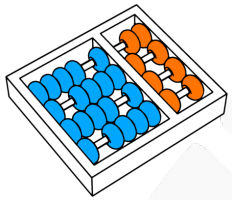


v	**a**	**b**	**c**	**d**	**e**
dist(**b**, v)	2	0	7	−2	4



Representando caminhos mínimos

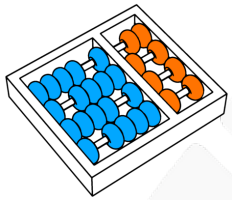
A saída é similar à da busca em largura a partir de **s**:



Representando caminhos mínimos

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

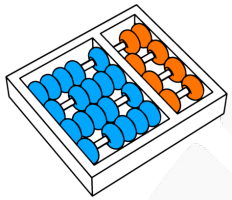
- Para cada **v** $\in$ **V[G]**, associamos um **PREDECESSOR** π[**v**].



Representando caminhos mínimos

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

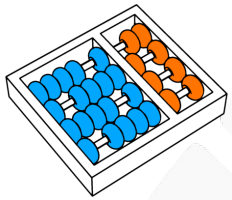
- Para cada **v** $\in$ **V[G]**, associamos um **PREDECESSOR** π[**v**].
- O vetor π induz uma **ÁRVORE DE CAMINHOS MÍNIMOS** com raiz em **s**.



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.

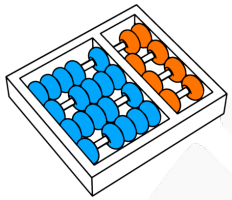


Definição

Problema (Caminhos mínimos com mesma origem)

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

Saída: *Um vetor* d *com* $d[v] = \text{dist}(s,v)$ *para* $v \in V$ *e um vetor* π *definindo uma* ***ÁRVORE DE CAMINHOS MÍNIMOS****.*



Subestrutura ótima de caminhos mínimos

Teorema

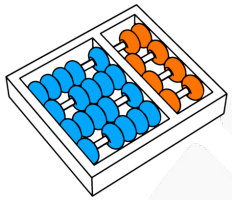
Seja (G, w) *um grafo direcionado* ***SEM CICLOS NEGATIVOS*** *e seja*

$$P = (v_1, v_2, \ldots, v_k)$$

um caminho mínimo de v_1 *a* v_k*. Então para quaisquer índices* i, j *com* $1 \leq i \leq j \leq k$*, o subcaminho*

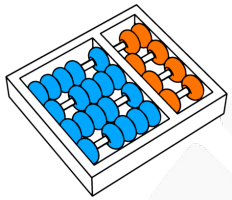
$$P_{ij} = (v_i, v_{i+1}, \ldots, v_j)$$

é um caminho mínimo de v_i *a* v_j*.*



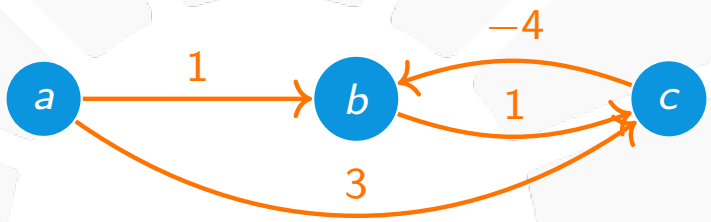
Subestrutura ótima de caminhos mínimos

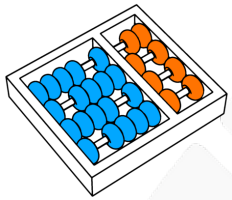
A subestrutura ótima **NÃO** vale se o grafo tiver **CICLOS NEGATIVOS**:



Subestrutura ótima de caminhos mínimos

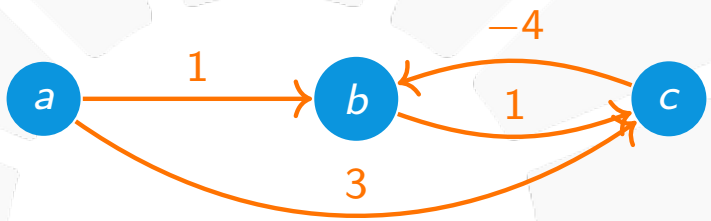
A subestrutura ótima **NÃO** vale se o grafo tiver **CICLOS NEGATIVOS**:



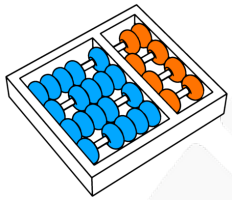


Subestrutura ótima de caminhos mínimos

A subestrutura ótima **NÃO** vale se o grafo tiver **CICLOS NEGATIVOS**:

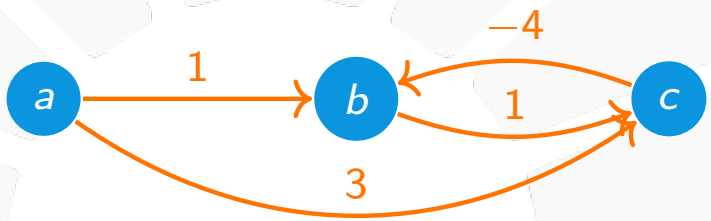


- $(\mathbf{a}, \mathbf{b}, \mathbf{c})$ é um caminho mínimo de **a** a **c** com peso $1 + 1 = 2$.

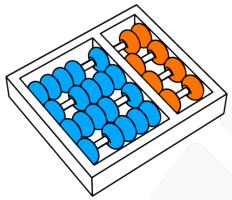


Subestrutura ótima de caminhos mínimos

A subestrutura ótima **NÃO** vale se o grafo tiver **CICLOS NEGATIVOS**:

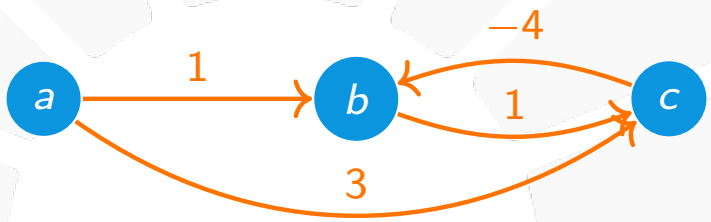


- (**a**, **b**, **c**) é um caminho mínimo de **a** a **c** com peso $1 + 1 = 2$.
- Mas, (**a**,**b**) **NÃO** é um caminho mínimo de **a** a **b**.

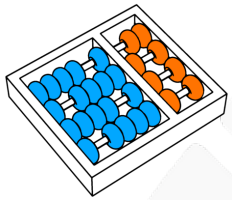


Subestrutura ótima de caminhos mínimos

A subestrutura ótima **NÃO** vale se o grafo tiver **CICLOS NEGATIVOS**:

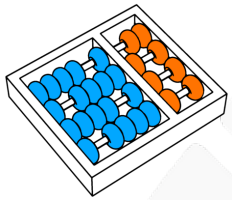


- (**a**, **b**, **c**) é um caminho mínimo de **a** a **c** com peso $1 + 1 = 2$.
- Mas, (**a**,**b**) **NÃO** é um caminho mínimo de **a** a **b**.
- (**a**,**c**,**b**) é um caminho mínimo de **a** a **b** com peso $3 - 4 = -1$.



Algoritmos baseados em relaxação

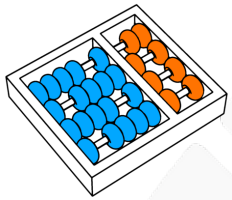
Veremos algoritmos com as seguintes características:



Algoritmos baseados em relaxação

Veremos algoritmos com as seguintes características:

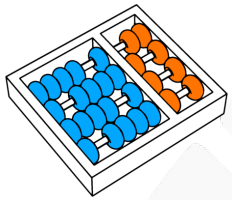
1. Inicializam d e π com uma sub-rotina INITIALIZE-SINGLE-SOURCE.



Algoritmos baseados em relaxação

Veremos algoritmos com as seguintes características:

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

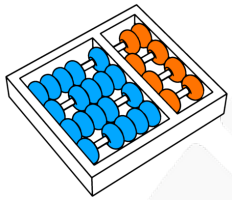


Algoritmos baseados em relaxação

Veremos algoritmos com as seguintes 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:



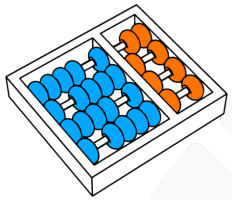
Algoritmos baseados em relaxação

Veremos algoritmos com as seguintes 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 $d[\mathbf{v}]$.



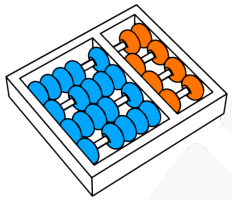
Algoritmos baseados em relaxação

Veremos algoritmos com as seguintes 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.



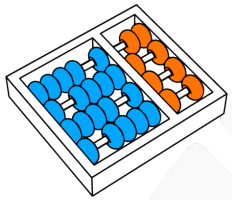
Algoritmos baseados em relaxação

Veremos algoritmos com as seguintes 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**).



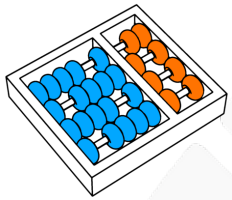
Algoritmos baseados em relaxação

Veremos algoritmos com as seguintes 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:

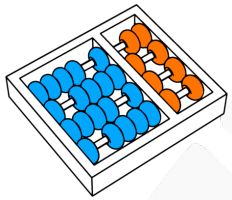
- Existe um caminho de **s** a **v** com peso $d[\mathbf{v}]$.
- Esse caminho pode ser recuperado por meio π.
- Assim, $d[\mathbf{v}]$ é sempre **MAIOR OU IGUAL** a dist(**s**,**v**).
- Queremos que no final valha $d[\mathbf{v}] = \text{dist}(\mathbf{s},\mathbf{v})$.



Inicialização

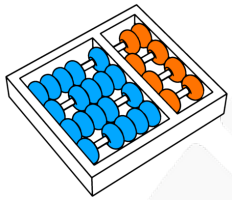
Algoritmo 1: INITIALIZE-SINGLE-SOURCE(G, s)

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



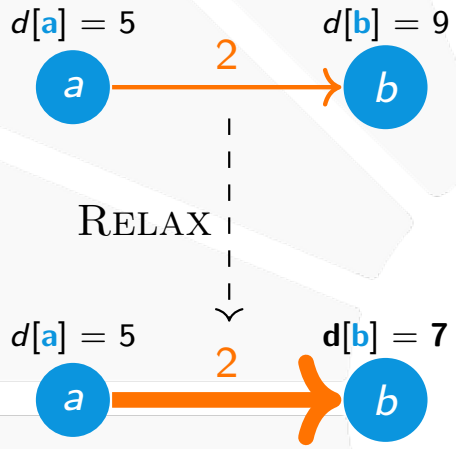
Relaxação

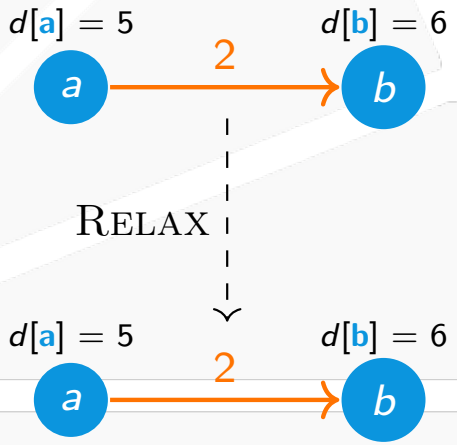
Tenta melhorar a estimativa $d[\mathbf{v}]$ examinando **(u,v)**.

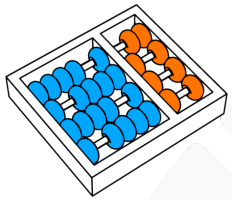


Relaxação

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

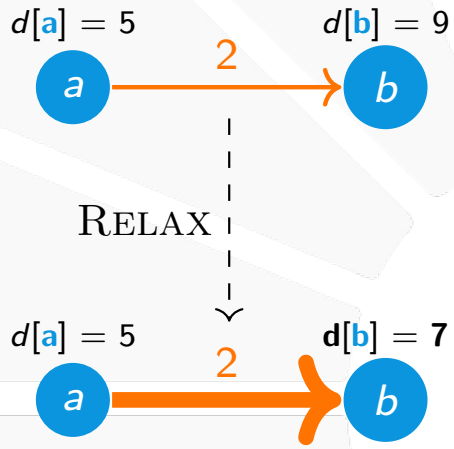


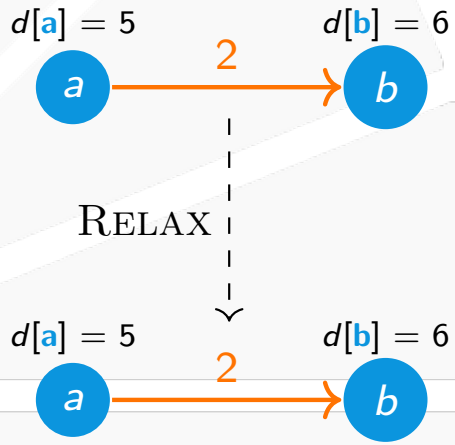




Relaxação

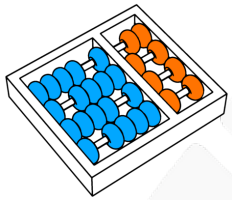
Tenta melhorar a estimativa $d[\mathbf{v}]$ examinando **(u,v)**.





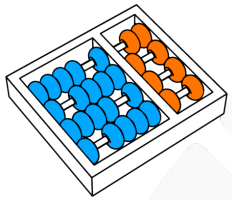
Algoritmo 4: RELAX(u, v, w)

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



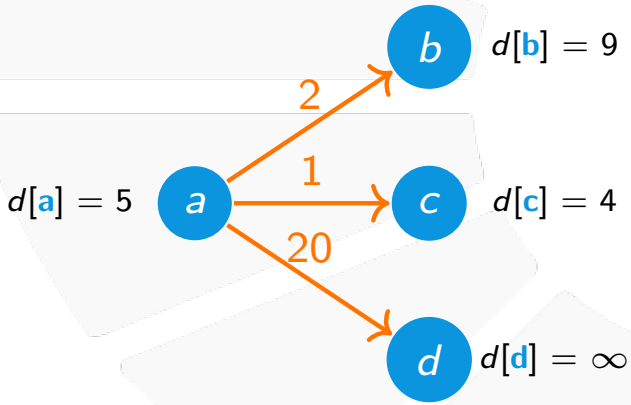
Relaxação dos vizinhos

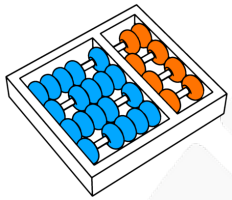
Em cada iteração o algoritmo seleciona um vértice **u** e para cada vizinho **v** de **u** aplica $\textsc{Relax}(\mathbf{u},\mathbf{v}, w)$.



Relaxação dos vizinhos

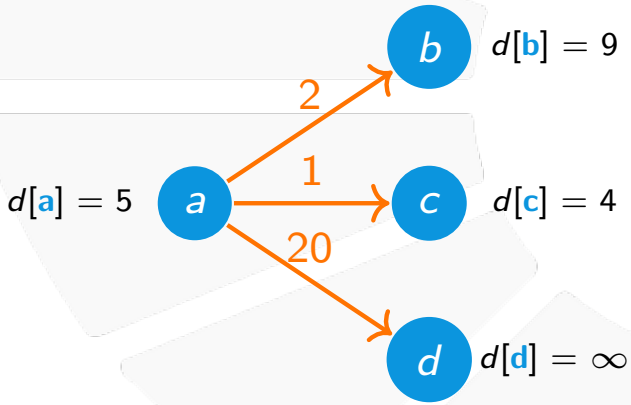
Em cada iteração o algoritmo seleciona um vértice **u** e para cada vizinho **v** de **u** aplica $\textsc{Relax}(\mathbf{u},\mathbf{v}, w)$.

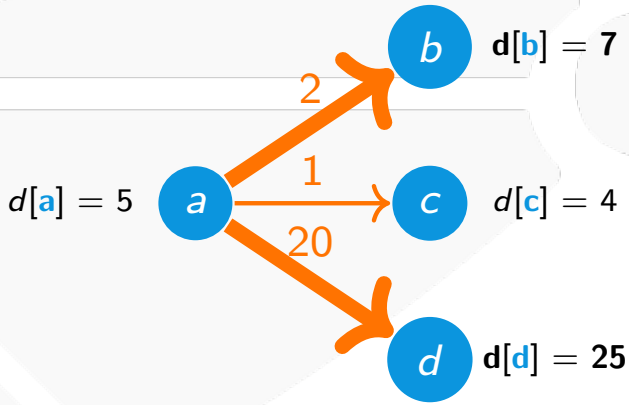


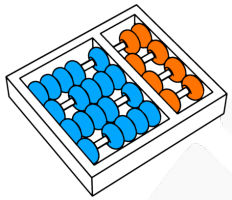


Relaxação dos vizinhos

Em cada iteração o algoritmo seleciona um vértice **u** e para cada vizinho **v** de **u** aplica $\textsc{Relax}(\mathbf{u},\mathbf{v}, w)$.

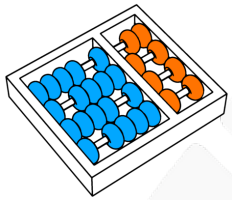






Casos do problema de caminhos mínimos

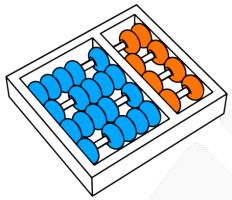
Veremos três algoritmos baseados em relaxação para tipos de subcasos diferentes do Problema de caminhos mínimos:



Casos do problema de caminhos mínimos

Veremos três algoritmos baseados em relaxação para tipos de subcasos diferentes do Problema de caminhos mínimos:

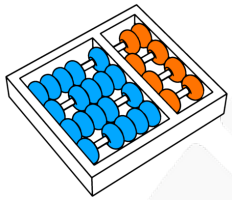
1. **APLICAÇÃO DE ORDENAÇÃO TOPOLÓGICA**: G é acíclico:



Casos do problema de caminhos mínimos

Veremos três algoritmos baseados em relaxação para tipos de subcasos diferentes do Problema de caminhos mínimos:

1. **APLICAÇÃO DE ORDENAÇÃO TOPOLÓGICA**: G é acíclico:
2. **ALGORITMO DE DIJKSTRA**: (G, w) não tem arestas de peso negativo.

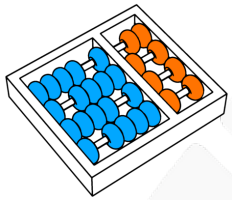


Casos do problema de caminhos mínimos

Veremos três algoritmos baseados em relaxação para tipos de subcasos diferentes do Problema de caminhos mínimos:

1. **APLICAÇÃO DE ORDENAÇÃO TOPOLÓGICA**: G é acíclico:
2. **ALGORITMO DE DIJKSTRA**: (G, w) não tem arestas de peso negativo.
3. **ALGORITMO DE BELLMAN-FORD**: (G, w) pode ter arestas de peso negativo, mas não contém ciclos negativos.

Caminhos mínimos em grafos acíclicos

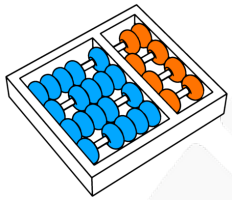


Caminhos mínimos em grafos acíclicos

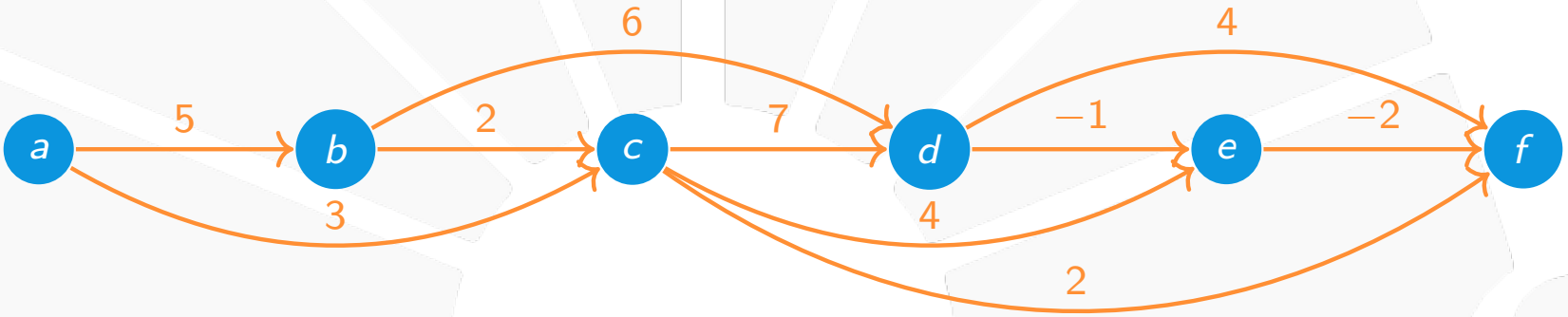
Problema

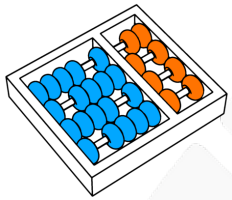
Entrada: *Um grafo direcionado acíclico* $G = (V, E)$, *uma função de peso* w *nas arestas e um vértice origem* s.

Saída: *Um vetor* d *com* $d[v] = \text{dist}(s,v)$ *para* $v \in V$ *e um vetor* π *definindo uma* ***ÁRVORE DE CAMINHOS MÍNIMOS***.

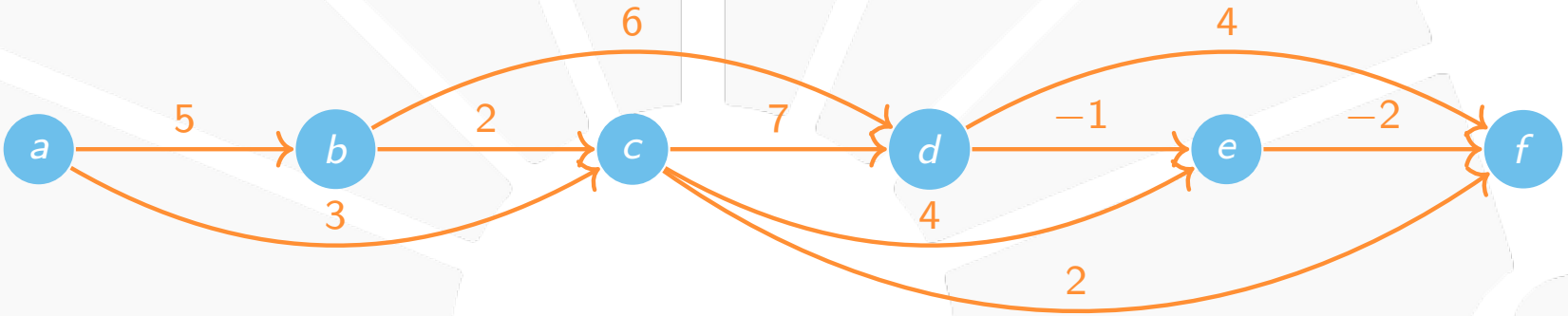


Exemplo

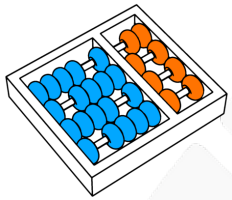




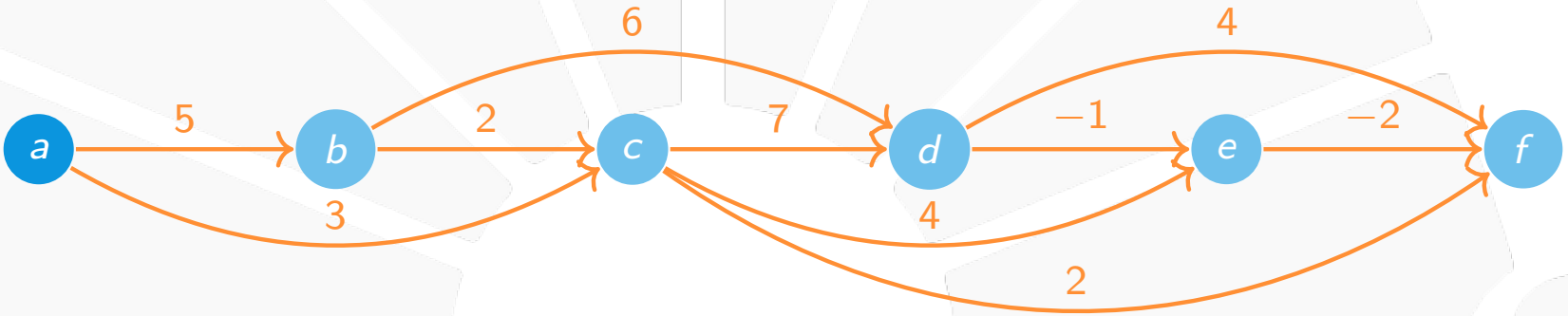
Exemplo



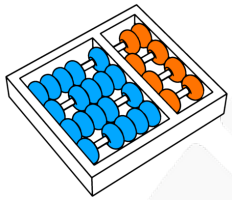
	a	b	c	d	e	f
d	∞	0	∞	∞	∞	∞



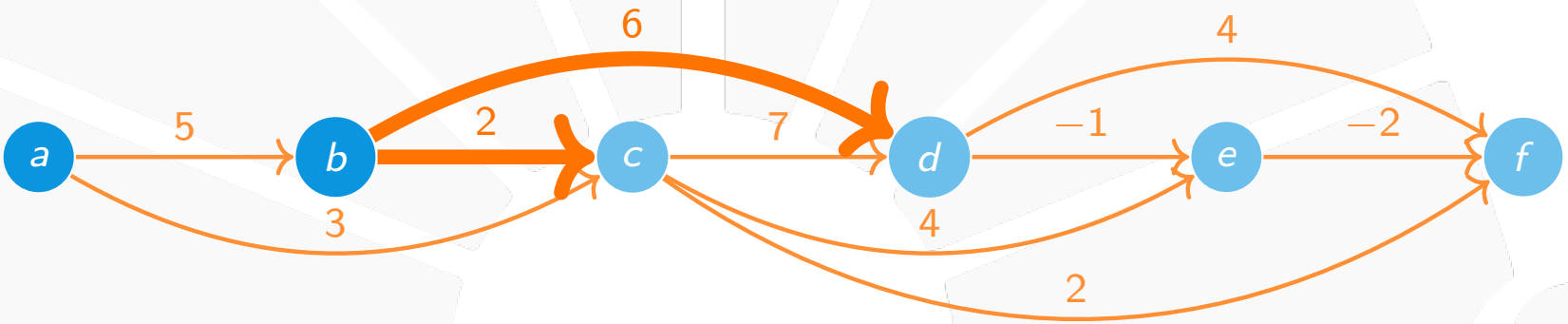
Exemplo



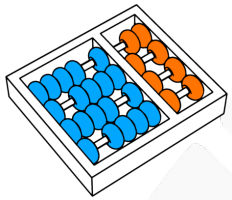
	a	b	c	d	e	f
d	∞	0	∞	∞	∞	∞



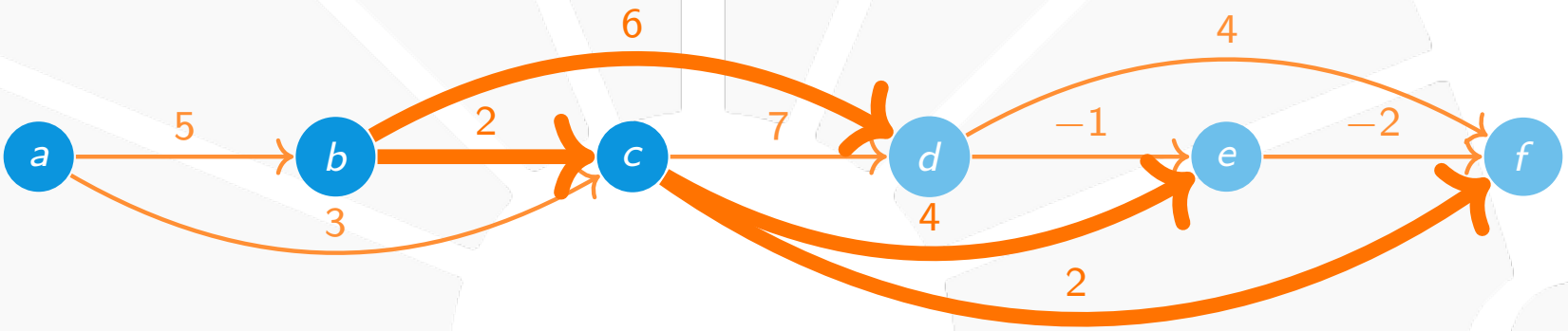
Exemplo



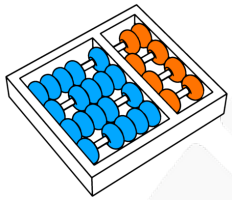
	a	b	c	d	e	f
d	∞	0	2	6	∞	∞



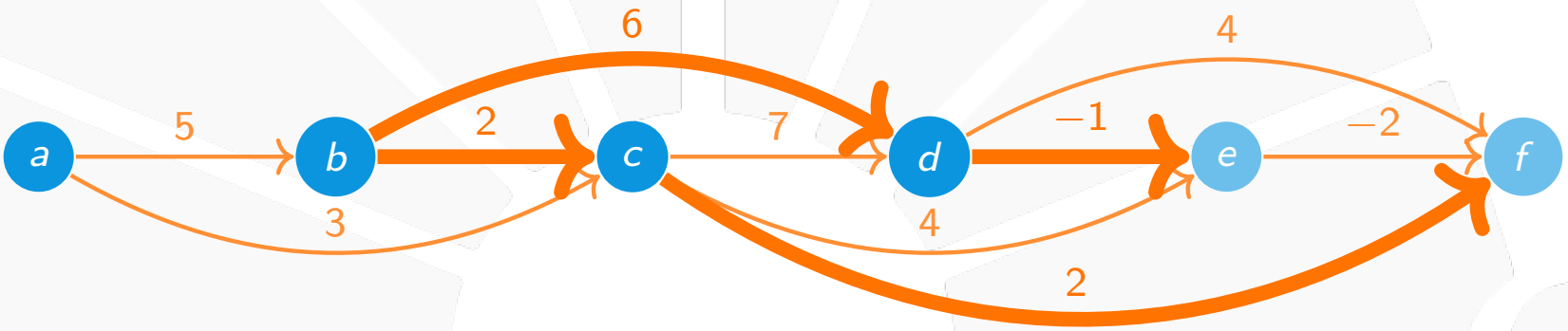
Exemplo



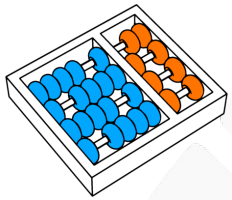
	a	b	c	d	e	f
d	∞	0	2	6	6	4



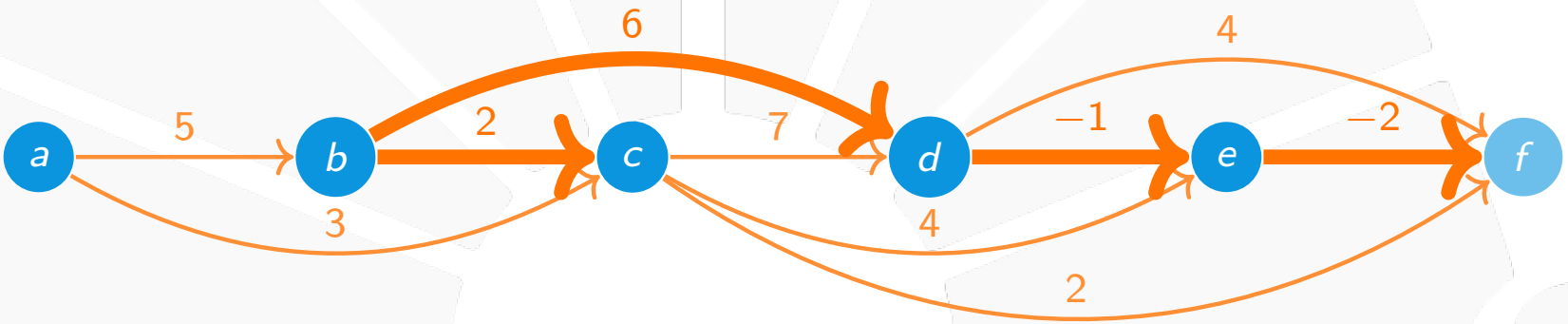
Exemplo



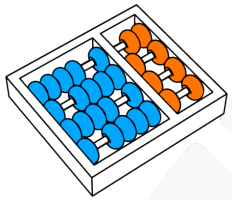
	a	b	c	d	e	f
d	∞	0	2	6	5	4



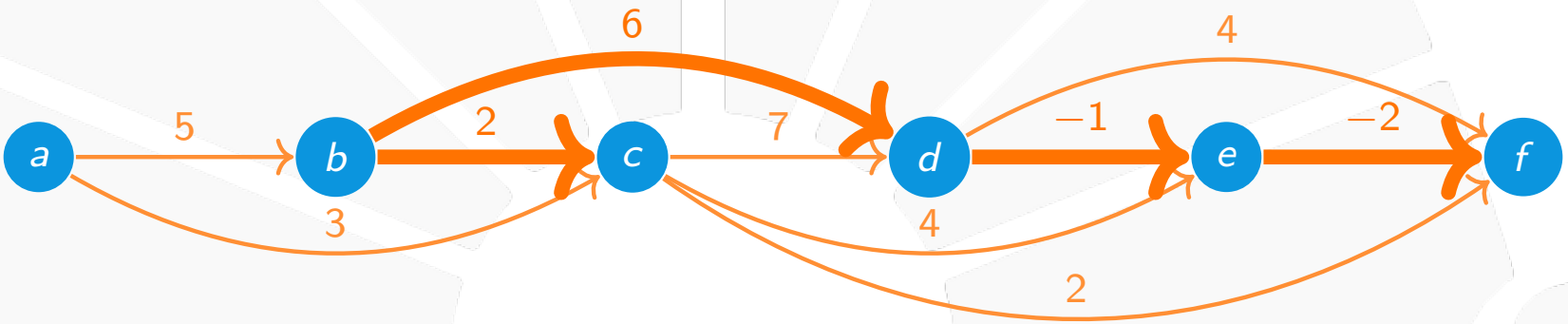
Exemplo



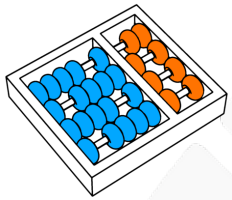
	a	b	c	d	e	f
d	∞	0	2	6	5	3



Exemplo



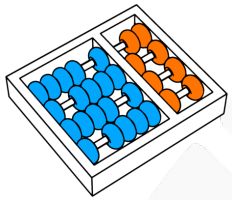
	a	b	c	d	e	f
d	∞	0	2	6	5	3



Algoritmo

Algoritmo 5: DAG-SHORTEST-PATHS(G, w, s)

```
1 ordene topologicamente os vértices de G
2 INITIALIZE-SINGLE-SOURCE(G, s)
3 para cada u na ordem topológica
4     para cada v ∈ Adj[u]
5         RELAX(u, v, w)
6 devolva d, π
```

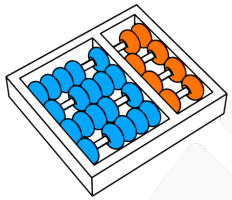


Algoritmo

Algoritmo 6: DAG-SHORTEST-PATHS(G, w, s)

```
1 ordene topologicamente os vértices de G
2 INITIALIZE-SINGLE-SOURCE(G, s)
3 para cada u na ordem topológica
4 |   para cada v ∈ Adj[u]
5 |   |   RELAX(u, v, w)
6 devolva d, π
```

Linha(s)	Tempo total
1	$O(V + E)$
2	$O(V)$
3-5	$O(V + E)$



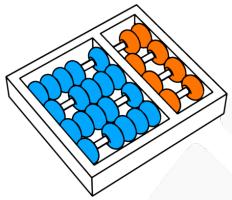
Algoritmo

Algoritmo 7: DAG-SHORTEST-PATHS(G, w, s)

```
1 ordene topologicamente os vértices de G
2 INITIALIZE-SINGLE-SOURCE(G, s)
3 para cada u na ordem topológica
4 |   para cada v ∈ Adj[u]
5 |   |   RELAX(u, v, w)
6 devolva d, π
```

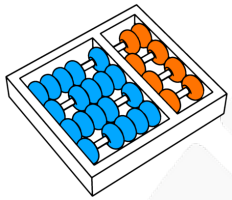
Linha(s)	Tempo total
1	$O(V + E)$
2	$O(V)$
3-5	$O(V + E)$

Complexidade de DAG-SHORTEST-PATHS: $O(V + E)$.



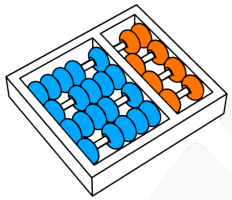
Correção

- Há diversas estratégias para demonstrar que DAG-SHORTEST-PATHS esteja correto.



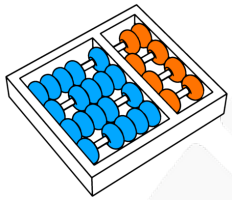
Correção

- Há diversas estratégias para demonstrar que DAG-SHORTEST-PATHS esteja correto.
- Vamos demonstrar algumas propriedades que valem porque o algoritmo é baseado em relaxação.



Correção

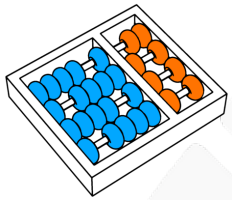
- Há diversas estratégias para demonstrar que DAG-SHORTEST-PATHS esteja correto.
- Vamos demonstrar algumas propriedades que valem porque o algoritmo é baseado em relaxação.
- Essas propriedades também serão úteis para analisar os outros algoritmos depois.



Propriedade de algoritmos baseados em relaxação

Ao longo de um algoritmo baseado em relaxação sempre vale:

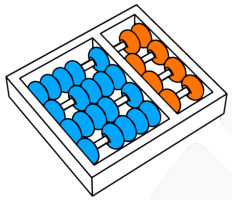
- **Limite superior:** Vale $d[\mathbf{v}] \geq \text{dist}(\mathbf{s},\mathbf{v})$ e, tão logo $d[\mathbf{v}]$ alcança $\text{dist}(\mathbf{s},\mathbf{v})$, nunca mais muda.



Propriedade de algoritmos baseados em relaxação

Ao longo de um algoritmo baseado em relaxação sempre vale:

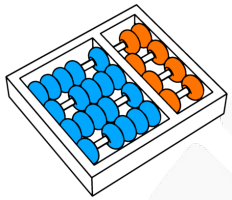
- **Limite superior:** Vale $d[\mathbf{v}] \geq \text{dist}(\mathbf{s},\mathbf{v})$ e, tão logo $d[\mathbf{v}]$ alcança dist(**s**,**v**), nunca mais muda.
- **Inexistência de caminho:** Se não existe caminho de **s** a **v**, então $d[\mathbf{v}] = \infty$.



Propriedade de algoritmos baseados em relaxação

Ao longo de um algoritmo baseado em relaxação sempre vale:

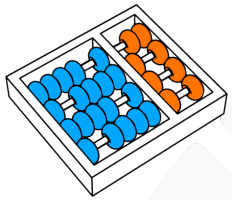
- **Limite superior:** Vale $d[\mathbf{v}] \geq \text{dist}(\mathbf{s},\mathbf{v})$ e, tão logo $d[\mathbf{v}]$ alcança $\text{dist}(\mathbf{s},\mathbf{v})$, nunca mais muda.
- **Inexistência de caminho:** Se não existe caminho de **s** a **v**, então $d[\mathbf{v}] = \infty$.
- **Subgrafo de predecessores:**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

Ao longo de um algoritmo baseado em relaxação sempre vale:

- **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.

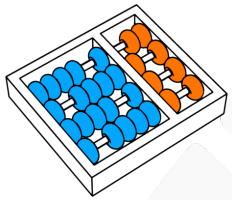


Propriedade de algoritmos baseados em relaxação

Ao longo de um algoritmo baseado em relaxação sempre vale:

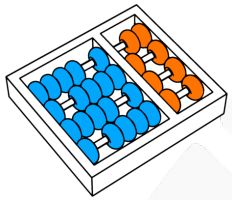
- **Limite superior:** Vale $d[v] \geq \text{dist}(s,v)$ e, tão logo $d[v]$ alcança $\text{dist}(s,v)$, nunca mais muda.
- **Inexistência de caminho:** Se não existe caminho de s a v, então $d[v] = \infty$.
- **Subgrafo de predecessores:** Se $d[v] < \infty$, então o subgrafo dos predecessores induzido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



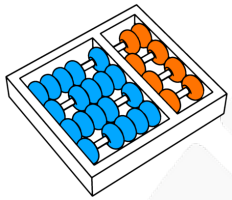
Correção de DAG-SHORTEST-PATHS

- Seja v um vértice e suponha que $p = (v_0, v_1, \ldots, v_k)$ é um caminho mínimo de $s = v_0$ a $v = v_k$.



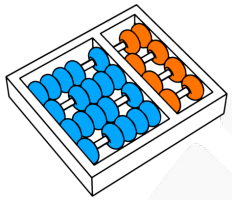
Correção de Dag-Shortest-Paths

- Seja v um vértice e suponha que $p = (v_0, v_1, \ldots, v_k)$ é um caminho mínimo de $s = v_0$ a $v = v_k$.
- Como $v_0, v_1, \ldots, v_k$ aparecem em ordem na ordenação topológica, as arestas $(v_0, v_1), \ldots, (v_{k-1}, v_k)$ são relaxadas em ordem.



Correção de DAG-SHORTEST-PATHS

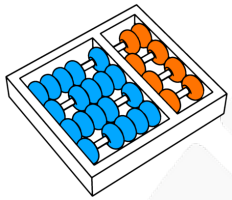
- Seja v um vértice e suponha que $p = (v_0, v_1, \ldots, v_k)$ é um caminho mínimo de $s = v_0$ a $v = v_k$.
- Como $v_0, v_1, \ldots, v_k$ aparecem em ordem na ordenação topológica, as arestas $(v_0, v_1), \ldots, (v_{k-1}, v_k)$ são relaxadas em ordem.
- Logo, pela propriedade do **Relaxamento de caminho**, o algoritmo computa corretamente $d[v] = \text{dist}(s,v)$ para cada $v \in V$.



Correção de Dag-Shortest-Paths

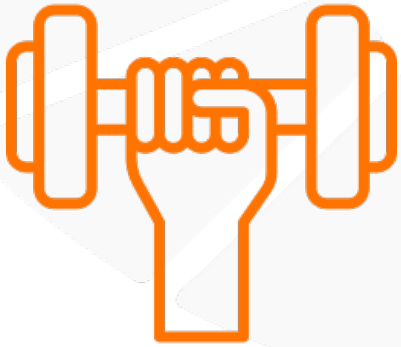
- Seja v um vértice e suponha que $p = (v_0, v_1, \ldots, v_k)$ é um caminho mínimo de $s = v_0$ a $v = v_k$.
- Como $v_0, v_1, \ldots, v_k$ aparecem em ordem na ordenação topológica, as arestas $(v_0, v_1), \ldots, (v_{k-1}, v_k)$ são relaxadas em ordem.
- Logo, pela propriedade do **Relaxamento de caminho**, o algoritmo computa corretamente $d[v] = \text{dist}(s,v)$ para cada $v \in V$.

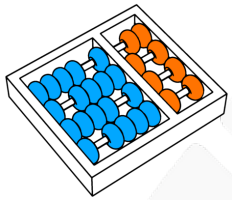
Também é fácil ver que π define uma árvore de caminhos mínimos. Por quê?



Perguntas

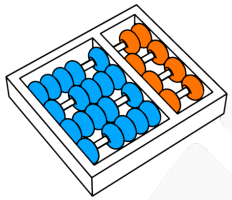
Vamos fazer alguns exercícios?





Exercício 1

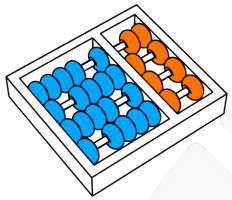
Como se resolve o problema de encontrar um caminho de peso **MÁXIMO** de **s** a **t** em um grafo direcionado acíclico (G, w)?



Exercício 2

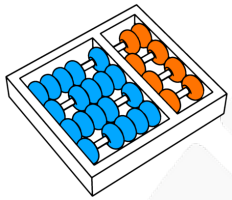
Como se resolve o problema do caminho mínimo de **s** a **t** em **TEMPO LINEAR** para um grafo direcionado em que todas as arestas têm o mesmo peso $C > 0$?

Algoritmo de Dijkstra



Sobre o algoritmo

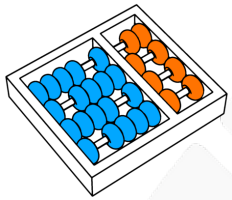
Veremos agora um algoritmo para caminhos mínimos em grafos que podem conter ciclos, mas **SEM ARESTAS DE PESOS NEGATIVO**.



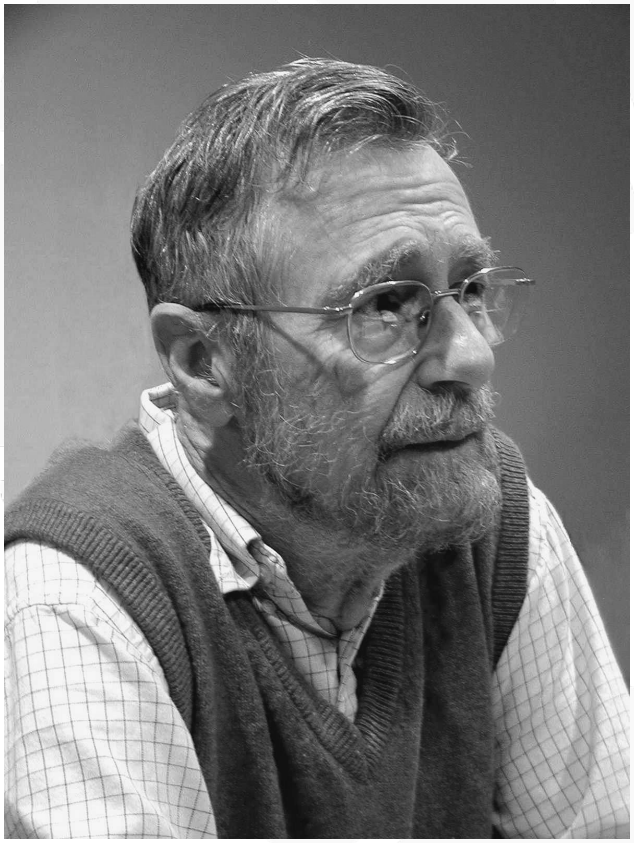
Sobre o algoritmo

Veremos agora um algoritmo para caminhos mínimos em grafos que podem conter ciclos, mas **SEM ARESTAS DE PESOS NEGATIVO**.

O algoritmo também é **BASEADO EM RELAXAÇÃO**.

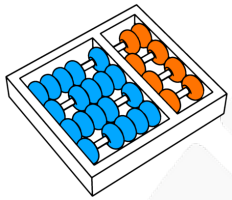


Edsger Wybe Dijkstra (11/05/1930 – 06/08/2002)



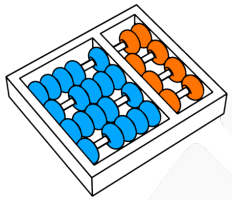
Principais áreas de atuação:

- Algoritmos em grafos.
- Programação concorrente e distribuída.
- Sistemas operacionais.
- Compiladores e linguagens de programação.



Publicação

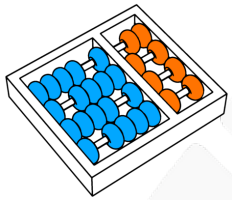
1957 *M. Leyzorek, R.S. Gray, A.A. Johnson, W.C. Ladew, S.R. Meaker, R.M. Petry* e *R.N. Seitz. Investigation of model techniques - First annual report - 6 June 1956 - 1 July 1957 - A study of model techniques for communication systems, Case Institute of Technology, Cleveland, Ohio*.



Publicação

1957 *M. Leyzorek, R.S. Gray, A.A. Johnson, W.C. Ladew, S.R. Meaker, R.M. Petry* e *R.N. Seitz. Investigation of model techniques - First annual report - 6 June 1956 - 1 July 1957 - A study of model techniques for communication systems, Case Institute of Technology, Cleveland, Ohio*.

1958 *G.B. Dantzig. On the shortest route through a network, The RAND Corporation, Santa Monica, California*.

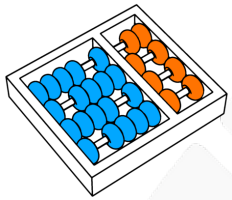


Publicação

1957 *M. Leyzorek, R.S. Gray, A.A. Johnson, W.C. Ladew, S.R. Meaker, R.M. Petry* e *R.N. Seitz. Investigation of model techniques - First annual report - 6 June 1956 - 1 July 1957 - A study of model techniques for communication systems, Case Institute of Technology, Cleveland, Ohio*.

1958 *G.B. Dantzig. On the shortest route through a network, The RAND Corporation, Santa Monica, California*.

1959 *E.W. Dijkstra. A note on two problems in connexion with graphs, Numerische Mathematik*.

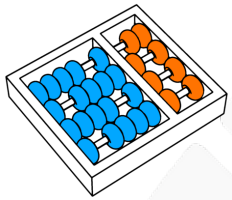


Problema

Problema

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

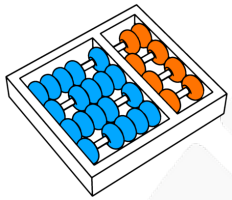
Saída: *Um vetor* d *com* $d[v] = \text{dist}(s,v)$ *para* $v \in V$ *e um vetor* π *definindo uma* ***ÁRVORE DE CAMINHOS MÍNIMOS***.



O algoritmo

Algoritmo 8: DIJKSTRA(G, w, s)

```
1 INITIALIZE-SINGLE-SOURCE(G, s)
2 S ← ∅
3 Q ← V[G] enquanto Q ≠ ∅
4     u ← EXTRACT-MIN(Q)
5     S ← S ∪ {u}
6     para cada v ∈ Adj[u]
7         RELAX(u, v, w)
8 devolva d, π
```

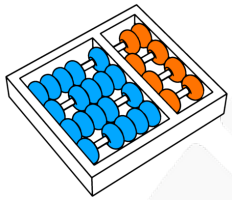


O algoritmo

Algoritmo 9: DIJKSTRA(G, w, s)

```
1 INITIALIZE-SINGLE-SOURCE(G, s)
2 S ← ∅
3 Q ← V[G] enquanto Q ≠ ∅
4     u ← EXTRACT-MIN(Q)
5     S ← S ∪ {u}
6     para cada v ∈ Adj[u]
7         RELAX(u, v, w)
8 devolva d, π
```

- O conjunto Q é implementado como uma fila de prioridade com chave d.

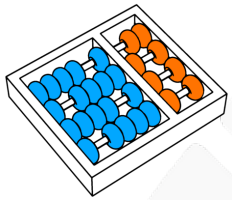


O algoritmo

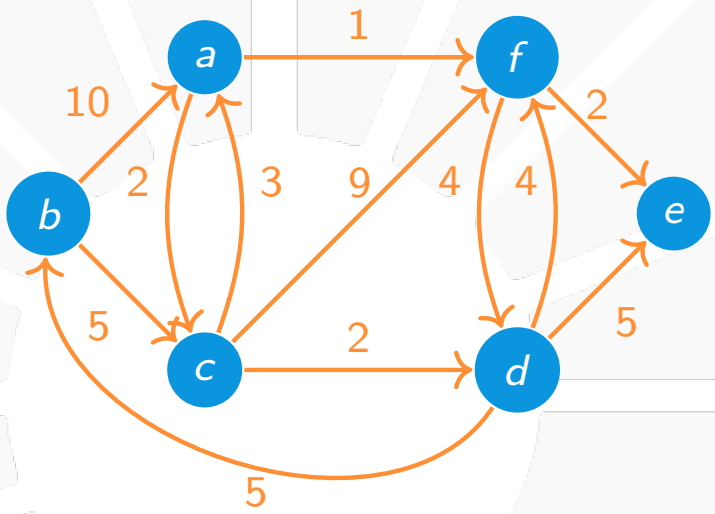
Algoritmo 10: DIJKSTRA(G, w, s)

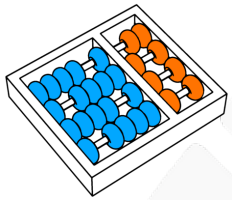
```
1 INITIALIZE-SINGLE-SOURCE(G, s)
2 S ← ∅
3 Q ← V[G] enquanto Q ≠ ∅
4     u ← EXTRACT-MIN(Q)
5     S ← S ∪ {u}
6     para cada v ∈ Adj[u]
7         RELAX(u, v, w)
8 devolva d, π
```

- O conjunto Q é implementado como uma fila de prioridade com chave d.
- O conjunto S não é realmente necessário, mas simplifica a análise do algoritmo.

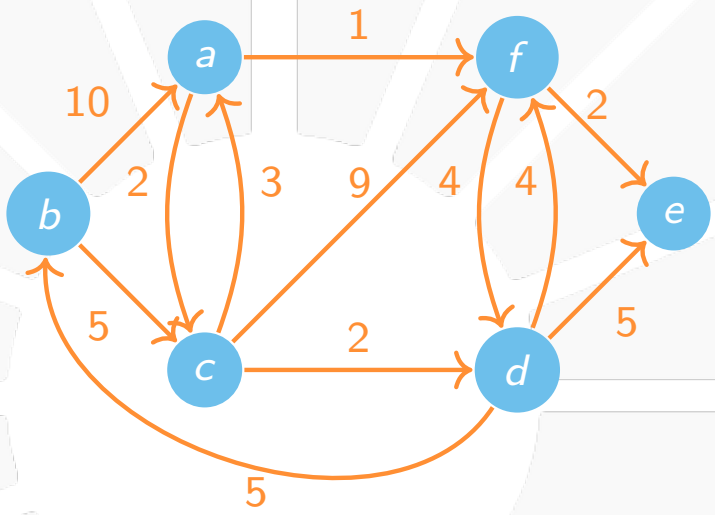


Exemplo

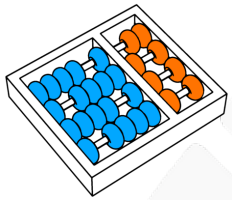




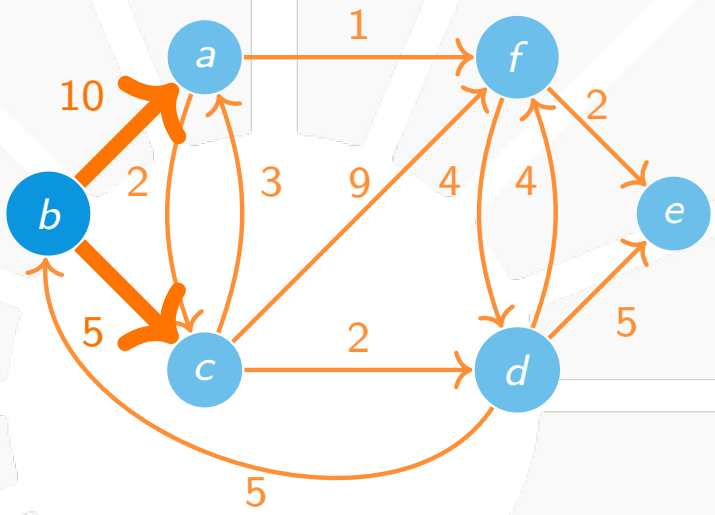
Exemplo



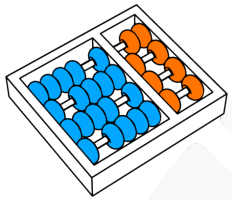
	a	b	c	d	e	f
d	∞	0	∞	∞	∞	∞



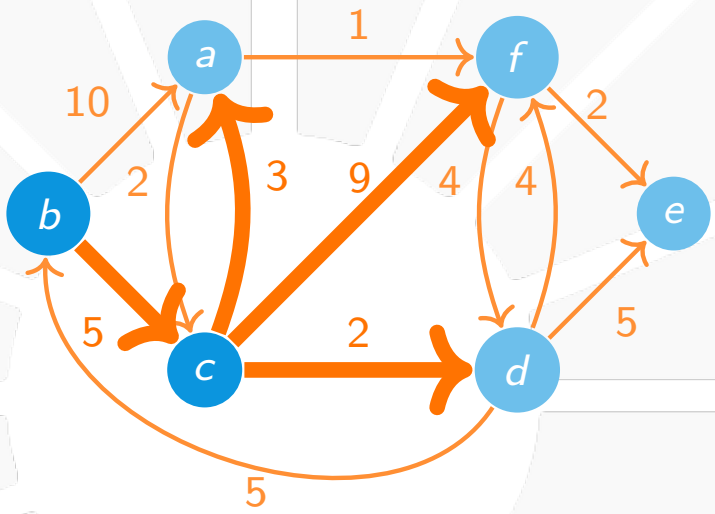
Exemplo



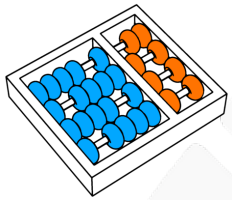
	a	b	c	d	e	f
d	10	0	5	∞	∞	∞



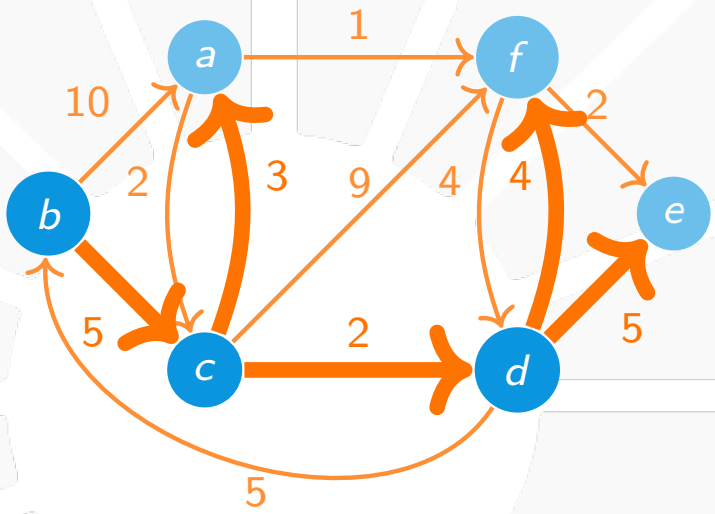
Exemplo



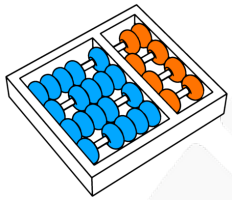
	a	b	c	d	e	f
d	8	0	5	7	∞	14



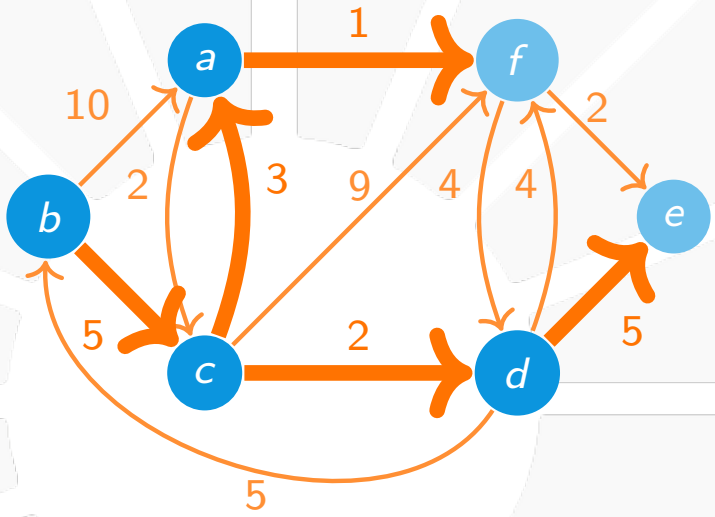
Exemplo



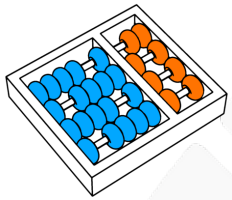
	a	b	c	d	e	f
d	8	0	5	7	12	11



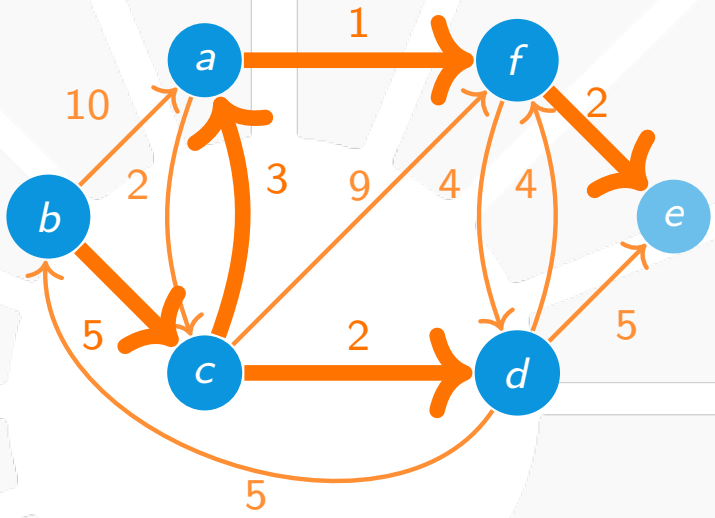
Exemplo



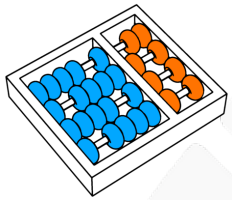
	a	b	c	d	e	f
d	8	0	5	7	12	9



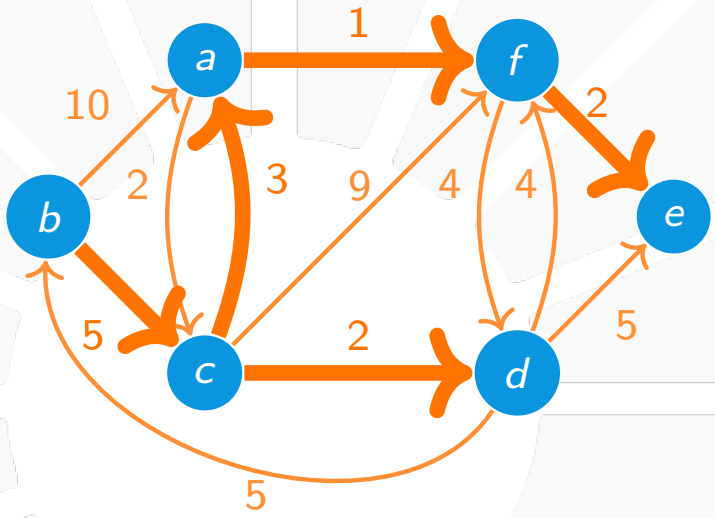
Exemplo



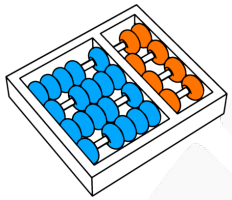
	a	b	c	d	e	f
d	8	0	5	7	11	9



Exemplo

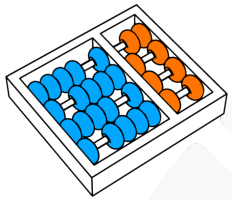


	a	b	c	d	e	f
d	8	0	5	7	11	9



Correção do algoritmo

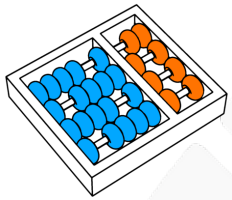
Precisamos provar que algoritmo está correto, temos que mostrar que, quando o algoritmo termina:



Correção do algoritmo

Precisamos provar que algoritmo está correto, temos que mostrar que, quando o algoritmo termina:

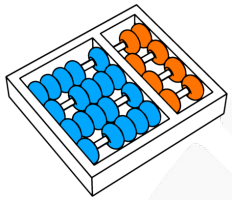
1. $d[\mathbf{v}] = \text{dist}(\mathbf{s},\mathbf{v})$ para todo $\mathbf{v} \in \mathbf{V[G]}$ e



Correção do algoritmo

Precisamos provar que algoritmo está correto, temos que mostrar que, quando o algoritmo termina:

1. $d[\mathbf{v}] = \text{dist}(\mathbf{s},\mathbf{v})$ para todo $\mathbf{v} \in \mathbf{V[G]}$ e
2. π induz uma **ÁRVORE DE CAMINHOS MÍNIMOS**.

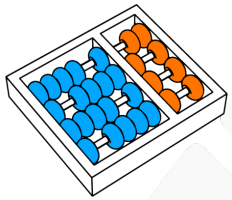


Correção do algoritmo

Precisamos provar que algoritmo está correto, temos que mostrar que, quando o algoritmo termina:

1. $d[\mathbf{v}] = \text{dist}(\mathbf{s},\mathbf{v})$ para todo $\mathbf{v} \in \mathbf{V[G]}$ e
2. π induz uma **ÁRVORE DE CAMINHOS MÍNIMOS**.

Observe que:



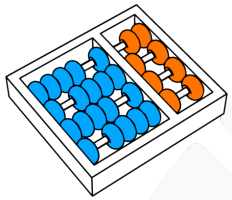
Correção do algoritmo

Precisamos provar que algoritmo está correto, temos que mostrar que, quando o algoritmo termina:

1. $d[\mathbf{v}] = \text{dist}(\mathbf{s},\mathbf{v})$ para todo $\mathbf{v} \in \mathbf{V[G]}$ e
2. π induz uma **ÁRVORE DE CAMINHOS MÍNIMOS**.

Observe que:

- DIJKSTRA é baseado em relaxação.



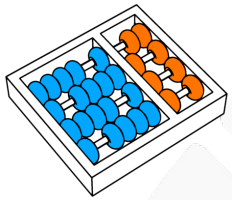
Correção do algoritmo

Precisamos provar que algoritmo está correto, temos que mostrar que, quando o algoritmo termina:

1. $d[\mathbf{v}] = \text{dist}(\mathbf{s},\mathbf{v})$ para todo $\mathbf{v} \in \mathbf{V[G]}$ e
2. π induz uma **ÁRVORE DE CAMINHOS MÍNIMOS**.

Observe que:

- DIJKSTRA é baseado em relaxação.
- Pela propriedade do **Subgrafo de predecessores**, sabemos que o vetor π induz uma árvore que testemunha d.



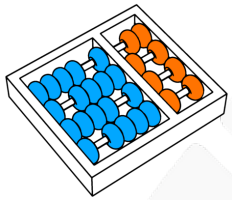
Correção do algoritmo

Precisamos provar que algoritmo está correto, temos que mostrar que, quando o algoritmo termina:

1. $d[\mathbf{v}] = \text{dist}(\mathbf{s},\mathbf{v})$ para todo $\mathbf{v} \in \mathbf{V[G]}$ e
2. π induz uma **ÁRVORE DE CAMINHOS MÍNIMOS**.

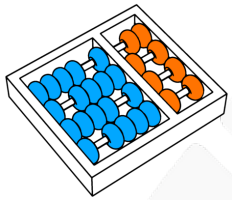
Observe que:

- DIJKSTRA é baseado em relaxação.
- Pela propriedade do **Subgrafo de predecessores**, sabemos que o vetor π induz uma árvore que testemunha d.
- Assim, basta mostrar que de fato $d[\mathbf{v}] = \text{dist}(\mathbf{s},\mathbf{v})$.



Invariante

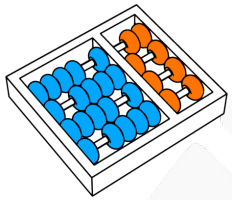
Iremos mostrar que no no início de cada iteração da linha 4 no algoritmo DIJKSTRA, vale $d[x] = \text{dist}(s, x)$ **para cada** $x \in S$.



Invariante

Iremos mostrar que no no início de cada iteração da linha 4 no algoritmo DIJKSTRA, vale $d[x] = \text{dist}(s, x)$ **para cada** $x \in S$.

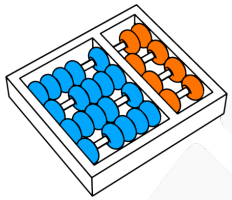
- **Inicialização.** No início, **S=**$\emptyset$, então a invariante vale trivialmente.



Invariante

Iremos mostrar que no no início de cada iteração da linha 4 no algoritmo DIJKSTRA, vale $d[x] = \text{dist}(s, x)$ **para cada** $x \in S$.

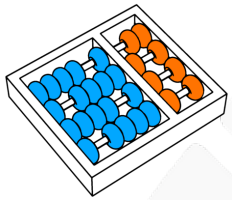
- **Inicialização.** No início, $S = \emptyset$, então a invariante vale trivialmente.
- **Manutenção.**



Invariante

Iremos mostrar que no no início de cada iteração da linha 4 no algoritmo DIJKSTRA, vale $d[x] = \text{dist}(s, x)$ **para cada** $x \in S$.

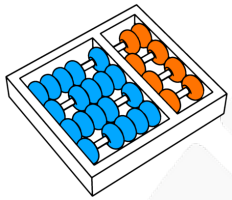
- **Inicialização.** No início, **S=**$\emptyset$, então a invariante vale trivialmente.
- **Manutenção.**
  - Suponha que a invariante vale para S no início da iteração.



Invariante

Iremos mostrar que no no início de cada iteração da linha 4 no algoritmo DIJKSTRA, vale $d[x] = \text{dist}(s, x)$ **para cada** $x \in S$.

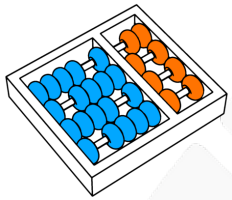
- **Inicialização.** No início, **S**$=\emptyset$, então a invariante vale trivialmente.
- **Manutenção.**
  - Suponha que a invariante vale para S no início da iteração.
  - Nessa iteração, DIJKSTRA escolhe um vértice **u** com menor d[**u**] em **Q** e o adiciona a **S**.



Invariante

Iremos mostrar que no no início de cada iteração da linha 4 no algoritmo DIJKSTRA, vale $d[x] = \text{dist}(s, x)$ **para cada** $x \in S$.

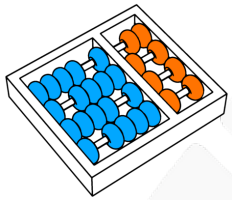
- **Inicialização.** No início, **S**$=\emptyset$, então a invariante vale trivialmente.
- **Manutenção.**
  - Suponha que a invariante vale para S no início da iteração.
  - Nessa iteração, DIJKSTRA escolhe um vértice **u** com menor d[**u**] em **Q** e o adiciona a **S**.
  - Queremos mostrar que a invariante vale para **S** $\cup \{$**u**$\}$.



Invariante

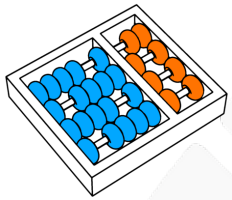
Iremos mostrar que no no início de cada iteração da linha 4 no algoritmo DIJKSTRA, vale $d[x] = \text{dist}(s, x)$ **para cada** $x \in S$.

- **Inicialização.** No início, **S**=$\emptyset$, então a invariante vale trivialmente.
- **Manutenção.**
  - Suponha que a invariante vale para S no início da iteração.
  - Nessa iteração, DIJKSTRA escolhe um vértice **u** com menor d[**u**] em **Q** e o adiciona a **S**.
  - Queremos mostrar que a invariante vale para **S** $\cup \{$**u**$\}$.
  - Basta verificar que neste instante d[**u**] = dist(**s**,**u**).



Demonstração

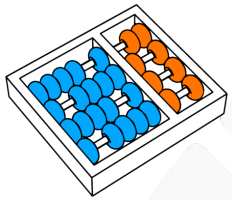
Sejam:



Demonstração

Sejam:

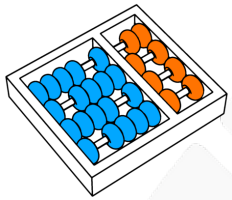
- P um caminho mínimo de **s** a **u** (i.e., com peso dist(**s**,**u**)).



Demonstração

Sejam:

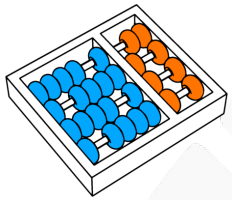
- P um caminho mínimo de **s** a **u** (i.e., com peso dist(**s**,**u**)).
- **y** o primeiro vértice de P que não pertence a **S**



Demonstração

Sejam:

- P um caminho mínimo de **s** a **u** (i.e., com peso dist(**s**,**u**)).
- **y** o primeiro vértice de P que não pertence a **S**
- **x** o vértice em P que precede **y**.

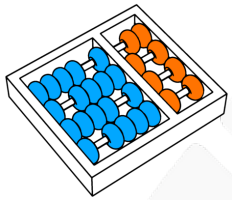


Demonstração

Sejam:

- P um caminho mínimo de **s** a **u** (i.e., com peso dist(**s**,**u**)).
- **y** o primeiro vértice de P que não pertence a **S**
- **x** o vértice em P que precede **y**.

Suponha que $d[\mathbf{u}] >$ dist(**s**,**u**):



Demonstração

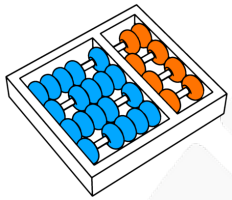
Sejam:

- P um caminho mínimo de **s** a **u** (i.e., com peso dist(**s**,**u**)).
- **y** o primeiro vértice de P que não pertence a **S**
- **x** o vértice em P que precede **y**.

Suponha que $d[\mathbf{u}] > \text{dist}(\mathbf{s},\mathbf{u})$:

- Pela hipótese de indução, $d[\mathbf{x}] = \text{dist}(\mathbf{s},\mathbf{x})$ pois $\mathbf{x} \in \mathbf{S}$. Logo,

$$d[\mathbf{y}] \quad \leq$$



Demonstração

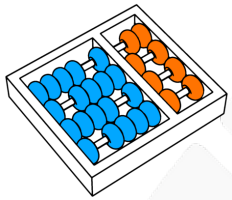
Sejam:

- P um caminho mínimo de **s** a **u** (i.e., com peso dist(**s**,**u**)).
- **y** o primeiro vértice de P que não pertence a **S**
- **x** o vértice em P que precede **y**.

Suponha que $d[\mathbf{u}] > \text{dist}(\mathbf{s},\mathbf{u})$:

- Pela hipótese de indução, $d[\mathbf{x}] = \text{dist}(\mathbf{s},\mathbf{x})$ pois $\mathbf{x} \in \mathbf{S}$. Logo,

$$d[\mathbf{y}] \quad \leq$$



Demonstração

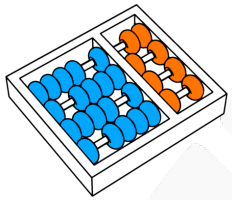
Sejam:

- P um caminho mínimo de **s** a **u** (i.e., com peso dist(**s**,**u**)).
- **y** o primeiro vértice de P que não pertence a **S**
- **x** o vértice em P que precede **y**.

Suponha que $d[\mathbf{u}] > \text{dist}(\mathbf{s},\mathbf{u})$:

- Pela hipótese de indução, $d[\mathbf{x}] = \text{dist}(\mathbf{s},\mathbf{x})$ pois $\mathbf{x} \in \mathbf{S}$. Logo,

$$d[\mathbf{y}] \leq d[\mathbf{x}] + w(\mathbf{x},\mathbf{y}) \quad \text{(pois relaxamos } (\mathbf{x},\mathbf{y})\text{)}$$



Demonstração

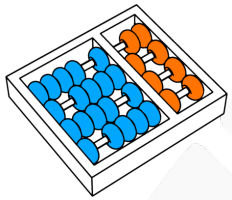
Sejam:

- P um caminho mínimo de **s** a **u** (i.e., com peso dist(**s**,**u**)).
- **y** o primeiro vértice de P que não pertence a **S**
- **x** o vértice em P que precede **y**.

Suponha que $d[\mathbf{u}] > \text{dist}(\mathbf{s},\mathbf{u})$:

- Pela hipótese de indução, $d[\mathbf{x}] = \text{dist}(\mathbf{s},\mathbf{x})$ pois $\mathbf{x} \in \mathbf{S}$. Logo,

$$\begin{aligned} d[\mathbf{y}] &\leq d[\mathbf{x}] + w(\mathbf{x},\mathbf{y}) \quad \text{(pois relaxamos } (\mathbf{x},\mathbf{y})) \\ &= \text{dist}(\mathbf{s},\mathbf{x}) + w(\mathbf{x},\mathbf{y}) \end{aligned}$$



Demonstração

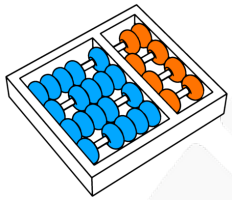
Sejam:

- P um caminho mínimo de **s** a **u** (i.e., com peso dist(**s**,**u**)).
- **y** o primeiro vértice de P que não pertence a **S**
- **x** o vértice em P que precede **y**.

Suponha que $d[\mathbf{u}] > \text{dist}(\mathbf{s},\mathbf{u})$:

- Pela hipótese de indução, $d[\mathbf{x}] = \text{dist}(\mathbf{s},\mathbf{x})$ pois $\mathbf{x} \in \mathbf{S}$. Logo,

$$\begin{aligned} d[\mathbf{y}] &\leq d[\mathbf{x}] + w(\mathbf{x},\mathbf{y}) \quad \text{(pois relaxamos } (\mathbf{x},\mathbf{y})) \\ &= \text{dist}(\mathbf{s},\mathbf{x}) + w(\mathbf{x},\mathbf{y}) \\ &\leq w(P) \end{aligned}$$



Demonstração

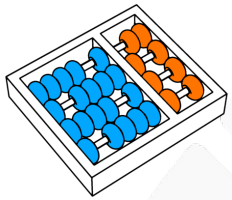
Sejam:

- P um caminho mínimo de **s** a **u** (i.e., com peso dist(**s**,**u**)).
- **y** o primeiro vértice de P que não pertence a **S**
- **x** o vértice em P que precede **y**.

Suponha que $d[\mathbf{u}] > \text{dist}(\mathbf{s},\mathbf{u})$:

- Pela hipótese de indução, $d[\mathbf{x}] = \text{dist}(\mathbf{s},\mathbf{x})$ pois $\mathbf{x} \in \mathbf{S}$. Logo,

$$\begin{aligned} d[\mathbf{y}] &\leq d[\mathbf{x}] + w(\mathbf{x},\mathbf{y}) \quad \text{(pois relaxamos } (\mathbf{x},\mathbf{y})) \\ &= \text{dist}(\mathbf{s},\mathbf{x}) + w(\mathbf{x},\mathbf{y}) \\ &\leq w(P) = \text{dist}(\mathbf{s},\mathbf{u}) \end{aligned}$$



Demonstração

Sejam:

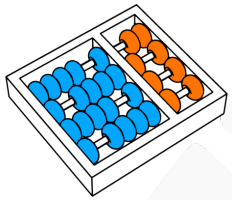
- P um caminho mínimo de **s** a **u** (i.e., com peso dist(**s**,**u**)).
- **y** o primeiro vértice de P que não pertence a **S**
- **x** o vértice em P que precede **y**.

Suponha que $d[\mathbf{u}] > \text{dist}(\mathbf{s},\mathbf{u})$:

- Pela hipótese de indução, $d[\mathbf{x}] = \text{dist}(\mathbf{s},\mathbf{x})$ pois $\mathbf{x} \in \mathbf{S}$. Logo,

$$\begin{aligned} d[\mathbf{y}] &\leq d[\mathbf{x}] + w(\mathbf{x},\mathbf{y}) \quad \text{(pois relaxamos } (\mathbf{x},\mathbf{y})) \\ &= \text{dist}(\mathbf{s},\mathbf{x}) + w(\mathbf{x},\mathbf{y}) \\ &\leq w(P) = \text{dist}(\mathbf{s},\mathbf{u}) < d[\mathbf{u}]. \end{aligned}$$

- Mas, $d[\mathbf{y}] < d[\mathbf{u}]$ contraria a escolha de **u**.



Demonstração

Sejam:

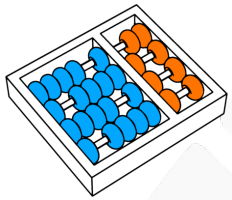
- P um caminho mínimo de **s** a **u** (i.e., com peso dist(**s**,**u**)).
- **y** o primeiro vértice de P que não pertence a **S**
- **x** o vértice em P que precede **y**.

Suponha que $d[\mathbf{u}] > \text{dist}(\mathbf{s},\mathbf{u})$:

- Pela hipótese de indução, $d[\mathbf{x}] = \text{dist}(\mathbf{s},\mathbf{x})$ pois $\mathbf{x} \in \mathbf{S}$. Logo,

$$\begin{aligned} d[\mathbf{y}] &\leq d[\mathbf{x}] + w(\mathbf{x},\mathbf{y}) \quad \text{(pois relaxamos } (\mathbf{x},\mathbf{y})) \\ &= \text{dist}(\mathbf{s},\mathbf{x}) + w(\mathbf{x},\mathbf{y}) \\ &\leq w(P) = \text{dist}(\mathbf{s},\mathbf{u}) < d[\mathbf{u}]. \end{aligned}$$

- Mas, $d[\mathbf{y}] < d[\mathbf{u}]$ contraria a escolha de **u**.
- Portanto, $d[\mathbf{u}] \leq \text{dist}(\mathbf{s},\mathbf{u})$.



Demonstração

Sejam:

- P um caminho mínimo de **s** a **u** (i.e., com peso dist(**s**,**u**)).
- **y** o primeiro vértice de P que não pertence a **S**
- **x** o vértice em P que precede **y**.

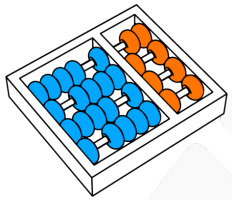
Suponha que $d[\mathbf{u}] > \text{dist}(\mathbf{s},\mathbf{u})$:

- Pela hipótese de indução, $d[\mathbf{x}] = \text{dist}(\mathbf{s},\mathbf{x})$ pois $\mathbf{x} \in \mathbf{S}$. Logo,

$$\begin{aligned} d[\mathbf{y}] &\leq d[\mathbf{x}] + w(\mathbf{x},\mathbf{y}) \quad \text{(pois relaxamos } (\mathbf{x},\mathbf{y})) \\ &= \text{dist}(\mathbf{s},\mathbf{x}) + w(\mathbf{x},\mathbf{y}) \\ &\leq w(P) = \text{dist}(\mathbf{s},\mathbf{u}) < d[\mathbf{u}]. \end{aligned}$$

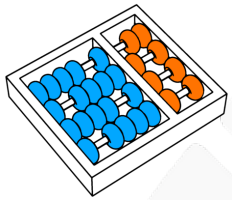
- Mas, $d[\mathbf{y}] < d[\mathbf{u}]$ contraria a escolha de **u**.
- Portanto, $d[\mathbf{u}] \leq \text{dist}(\mathbf{s},\mathbf{u})$.

Concluindo que $d[\mathbf{u}] = \text{dist}(\mathbf{s},\mathbf{u})$ (propriedade de **Limite superior**).



Demonstração

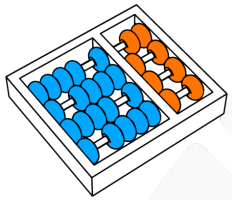
Para terminar a demonstração, observe que:



Demonstração

Para terminar a demonstração, observe que:

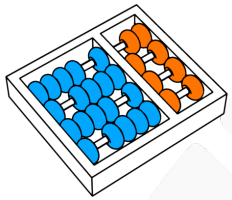
- Após a última iteração, **S** é o conjunto dos vértices atingíveis por **s**.



Demonstração

Para terminar a demonstração, observe que:

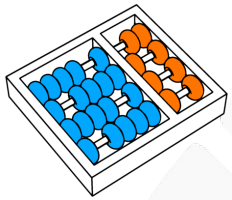
- Após a última iteração, **S** é o conjunto dos vértices atingíveis por **s**.
- Por **Inexistência de caminho**, se um vértice **v** não é atingível, então $d[\mathbf{v}] = \infty$.



Demonstração

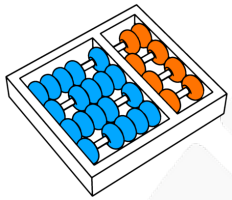
Para terminar a demonstração, observe que:

- Após a última iteração, **S** é o conjunto dos vértices atingíveis por **s**.
- Por **Inexistência de caminho**, se um vértice **v** não é atingível, então $d[\mathbf{v}] = \infty$.
- Portanto, para todo $\mathbf{v} \in \mathbf{V}$, vale $d[\mathbf{v}] = \text{dist}(\mathbf{s},\mathbf{v})$.



Dijkstra precisa de arestas com peso não negativo

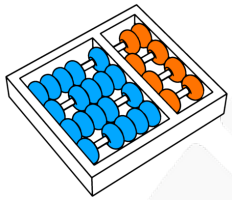
Na demonstração, supomos que **NÃO HÁ ARESTAS NEGATIVAS**



Dijkstra precisa de arestas com peso não negativo

Na demonstração, supomos que **NÃO HÁ ARESTAS NEGATIVAS**

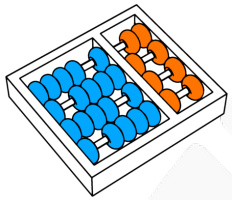
- Se a hipótese não valer, o algoritmo de Dijkstra pode falhar.



Dijkstra precisa de arestas com peso não negativo

Na demonstração, supomos que **NÃO HÁ ARESTAS NEGATIVAS**

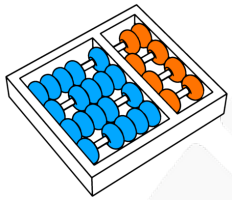
- Se a hipótese não valer, o algoritmo de Dijkstra pode falhar.
- Encontre um grafo com arestas negativas para o qual o algoritmo de Dijkstra **NÃO** funciona (exercício).



Dijkstra precisa de arestas com peso não negativo

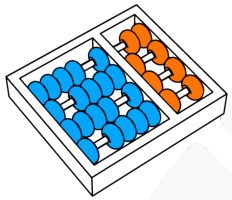
Na demonstração, supomos que **NÃO HÁ ARESTAS NEGATIVAS**

- Se a hipótese não valer, o algoritmo de Dijkstra pode falhar.
- Encontre um grafo com arestas negativas para o qual o algoritmo de Dijkstra **NÃO** funciona (exercício).
- Existe um exemplo com 4 vértices, com apenas uma aresta negativa e sem ciclos de peso negativo.



Complexidade de tempo

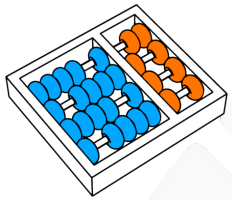
Depende de como a fila de prioridade Q é implementada:



Complexidade de tempo

Depende de como a fila de prioridade Q é implementada:

- Operações Insert, Extract-Min, Decrease-Key.

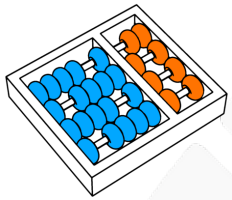


Complexidade de tempo

Depende de como a fila de prioridade Q é implementada:

- Operações INSERT, EXTRACT-MIN, DECREASE-KEY.

Observe que:



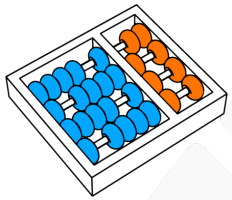
Complexidade de tempo

Depende de como a fila de prioridade Q é implementada:

- Operações INSERT, EXTRACT-MIN, DECREASE-KEY.

Observe que:

- Os passos INITIALIZE-SINGLE-SOURCE e $Q \leftarrow V[G]$ escondem chamadas a INSERT.



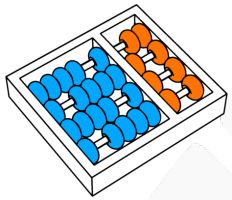
Complexidade de tempo

Depende de como a fila de prioridade Q é implementada:

- Operações INSERT, EXTRACT-MIN, DECREASE-KEY.

Observe que:

- Os passos INITIALIZE-SINGLE-SOURCE e $Q \leftarrow V[G]$ escondem chamadas a INSERT.
- RELAX esconde chamada a DECREASE-KEY.



Complexidade de tempo

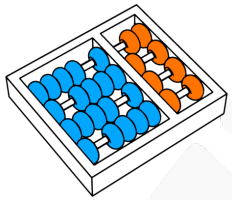
Depende de como a fila de prioridade Q é implementada:

- Operações INSERT, EXTRACT-MIN, DECREASE-KEY.

Observe que:

- Os passos INITIALIZE-SINGLE-SOURCE e $Q \leftarrow V[G]$ escondem chamadas a INSERT.
- RELAX esconde chamada a DECREASE-KEY.

Linha(s)	Tempo total
1–3	$\|V\|$ chamadas a INSERT
5	$\|V\|$ chamadas a EXTRACT-MIN
8	$\|E\|$ chamadas a DECREASE-KEY



Complexidade de tempo

Depende de como a fila de prioridade Q é implementada:

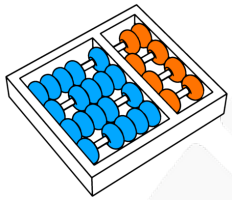
- Operações INSERT, EXTRACT-MIN, DECREASE-KEY.

Observe que:

- Os passos INITIALIZE-SINGLE-SOURCE e $Q \leftarrow V[G]$ escondem chamadas a INSERT.
- RELAX esconde chamada a DECREASE-KEY.

Linha(s)	Tempo total
1–3	$\|V\|$ chamadas a INSERT
5	$\|V\|$ chamadas a EXTRACT-MIN
8	$\|E\|$ chamadas a DECREASE-KEY

Complexidade de Dijkstra:
$O(|V| \times \text{INSERT} + |V| \times \text{EXTRACT-MIN} + |E| \times \text{DECREASE-KEY})$.



Complexidade de tempo

Total: $O(|V| \times \textsc{Insert} + |V| \times \textsc{Extract-Min} + |E| \times \textsc{Decrease-Key})$

Tipo de fila	INSERT	EXTRACT-MIN	DECREASE-KEY	TOTAL
Vetor	$O(1)$	$O(V)$	$O(1)$	$O(V^2)$
Min-Heap	$O(\log V)$	$O(\log V)$	$O(\log V)$	$O((V+E)\log V)$
Fibonacci	$O(1)$	$O(\log V)$	$O(1)$	$O(V \log V + E)$

Caminhos mínimos

MC558 - Projeto e Análise de Algoritmos II

01/24

6

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

UNICAMP