Grafos

Algoritmo de Bellman-Ford

Prof. Edson Alves

Faculdade UnB Gama

Proponentes

Proponentes

Lester Randolph Ford Jr.

(1956)

Proponentes

Lester Randolph Ford Jr.
(1956)

Richard Ernest Bellman
(1958)

Características do algoritmo de Bellman-Ford

Características do algoritmo de Bellman-Ford

- $\star$ **Computa o caminho mínimo de todos os vértices de $G(V, E)$ a um dado nó s**

Características do algoritmo de Bellman-Ford

- ⋆ Computa o caminho mínimo de todos os vértices de $G(V, E)$ a um dado nó s
- ⋆ É capaz de processar arestas negativas

Características do algoritmo de Bellman-Ford

- ⋆ Computa o caminho mínimo de todos os vértices de $G(V, E)$ a um dado nó s
- ⋆ É capaz de processar arestas negativas
- ⋆ Não processa, mas identifica ciclos negativos

Características do algoritmo de Bellman-Ford

- $\star$ **Computa o caminho mínimo de todos os vértices de $G(V, E)$ a um dado nó s**
- $\star$ **É capaz de processar arestas negativas**
- $\star$ **Não processa, mas identifica ciclos negativos**
- $\star$ **Complexidade:** $O(VE)$

Pseudocódigo

Pseudocódigo

Entrada: um grafo $G(V, E)$ e um vértice $s \in V$

Saída: um vetor d tal que $d[u]$ é a distância mínima em G entre s e u

Pseudocódigo

Entrada: **um grafo** $G(V, E)$ **e um vértice** $s \in V$
Saída: **um vetor** d **tal que** $d[u]$ **é a distância mínima em** G **entre** s **e** u

1. **Faça** $d[s] = 0$ **e** $d[u] = \infty$ **para todos vértices** $u \in V$ **tais que** $u \neq s$

Pseudocódigo

Entrada: um grafo $G(V, E)$ e um vértice $s \in V$
Saída: um vetor d tal que $d[u]$ é a distância mínima em G entre s e u

1. Faça $d[s] = 0$ e $d[u] = \infty$ para todos vértices $u \in V$ tais que $u \neq s$

2. Para cada aresta $(u, v, w) \in E$, se existe um caminho de s a u (isto é, $d[u] < \infty$) e $d[u] + w < d[v]$, faça $d[v] = d[u] + w$

Pseudocódigo

Entrada: um grafo $G(V, E)$ e um vértice $s \in V$
Saída: um vetor d tal que $d[u]$ é a distância mínima em G entre s e u

1. Faça $d[s] = 0$ e $d[u] = \infty$ para todos vértices $u \in V$ tais que $u \neq s$

2. Para cada aresta $(u, v, w) \in E$, se existe um caminho de s a u (isto é, $d[u] < \infty$) e $d[u] + w < d[v]$, faça $d[v] = d[u] + w$

3. Se o vetor d foi atualizado ao menos uma vez, volte ao passo 2.

Pseudocódigo

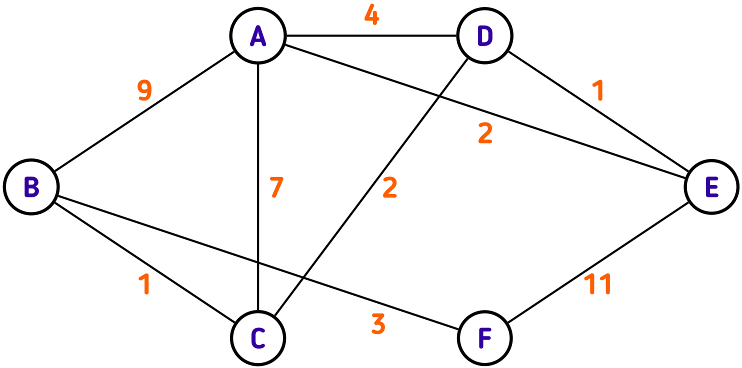
Entrada: um grafo $G(V, E)$ e um vértice $s \in V$
Saída: um vetor d tal que $d[u]$ é a distância mínima em G entre s e u

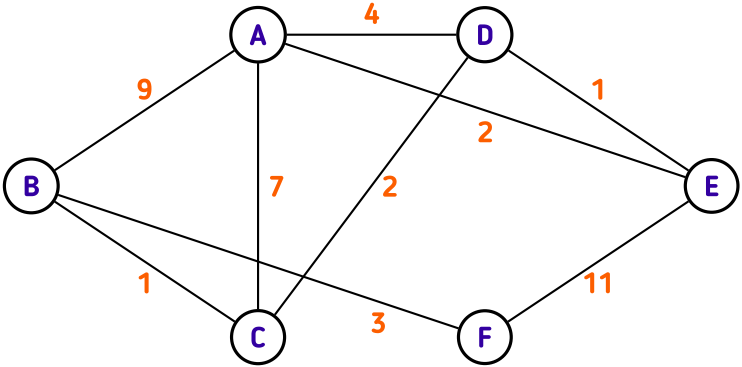
1. Faça $d[s] = 0$ e $d[u] = \infty$ para todos vértices $u \in V$ tais que $u \neq s$

2. Para cada aresta $(u, v, w) \in E$, se existe um caminho de s a u (isto é, $d[u] < \infty$) e $d[u] + w < d[v]$, faça $d[v] = d[u] + w$

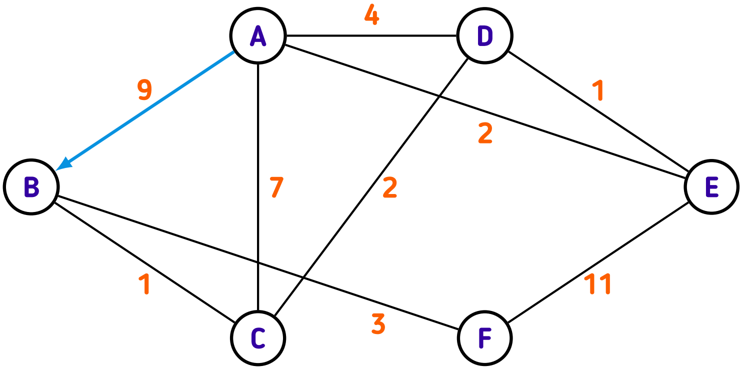
3. Se o vetor d foi atualizado ao menos uma vez, volte ao passo 2.

4. Retorne d

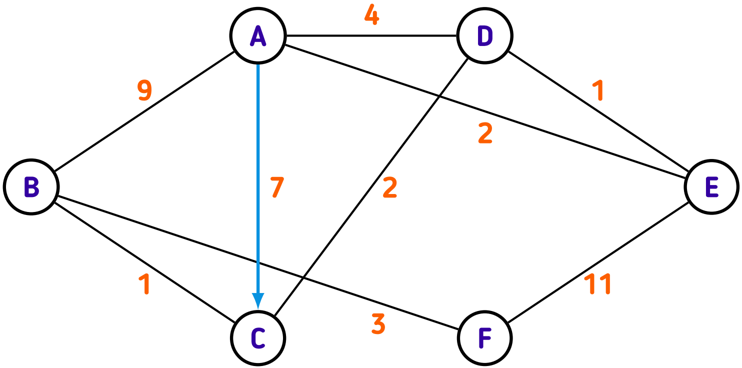

A
B
C
D
E
F
4
9
7
2
1
2
1
3
11



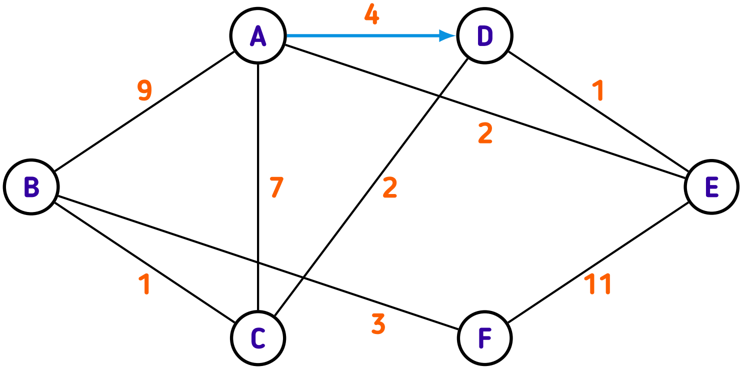
	A	B	C	D	E	F
dist$(u, \mathbf{A})$	0	∞	∞	∞	∞	∞



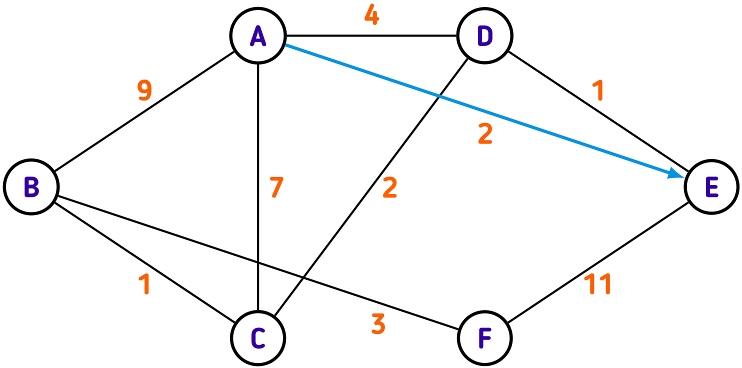
	A	B	C	D	E	F
dist$(u,$ **A**$)$	0	9	∞	∞	∞	∞



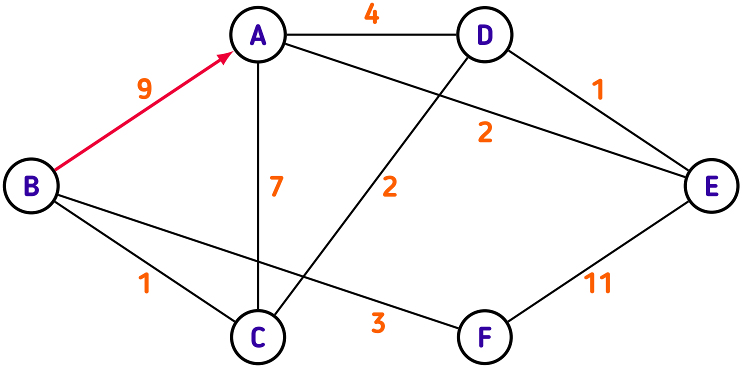
	A	B	C	D	E	F
dist$(u,$ **A**$)$	0	9	**7**	∞	∞	∞



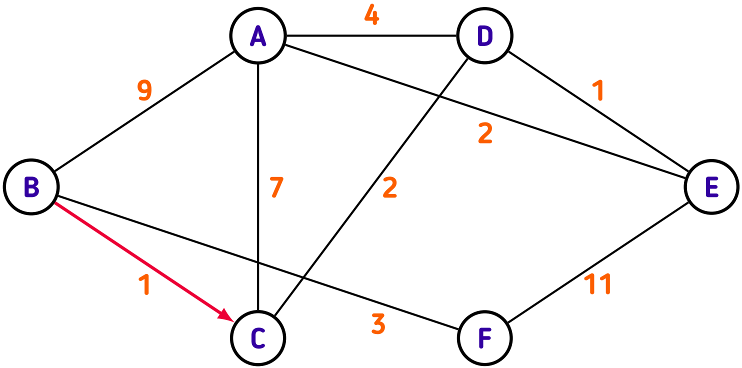
	A	B	C	D	E	F
dist(u, **A**)	0	9	7	**4**	∞	∞



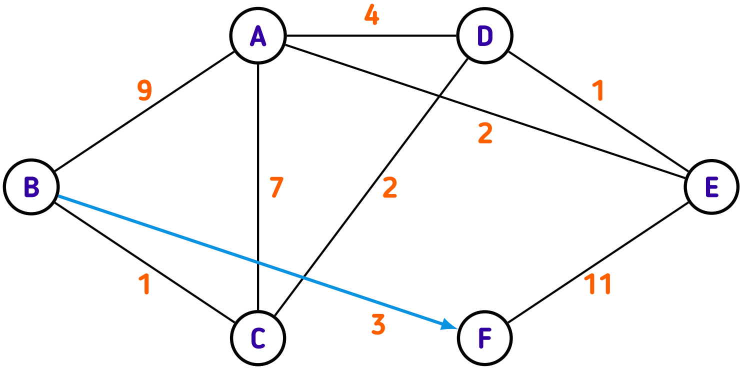
	A	B	C	D	E	F
dist(u, **A**)	0	9	7	4	**2**	∞



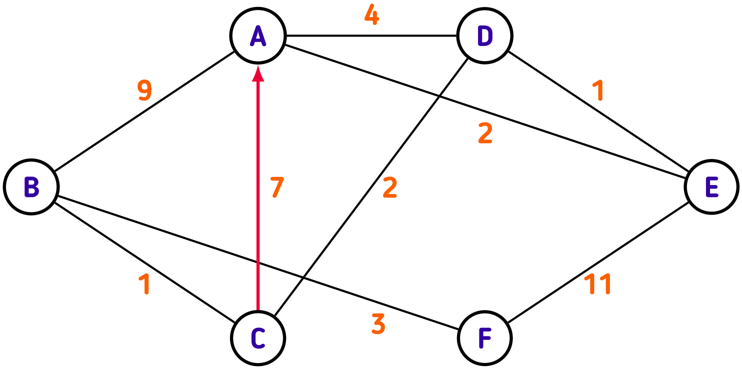
	A	B	C	D	E	F
dist$(u,$ **A**$)$	0	9	7	4	2	∞



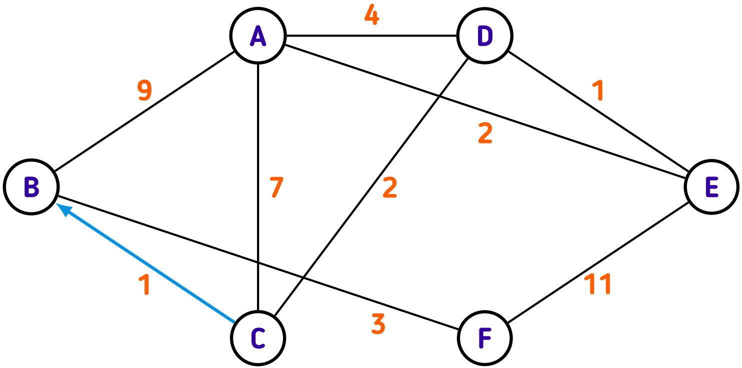
	A	B	C	D	E	F
dist$(u,$ **A**$)$	0	9	7	4	2	∞



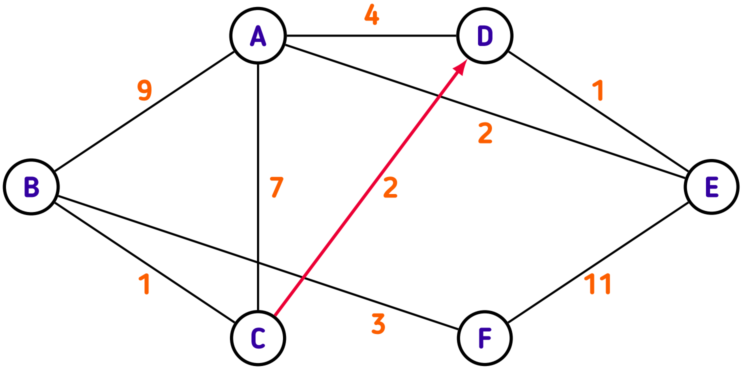
	A	B	C	D	E	F
dist(u, **A**)	0	9	7	4	2	**12**



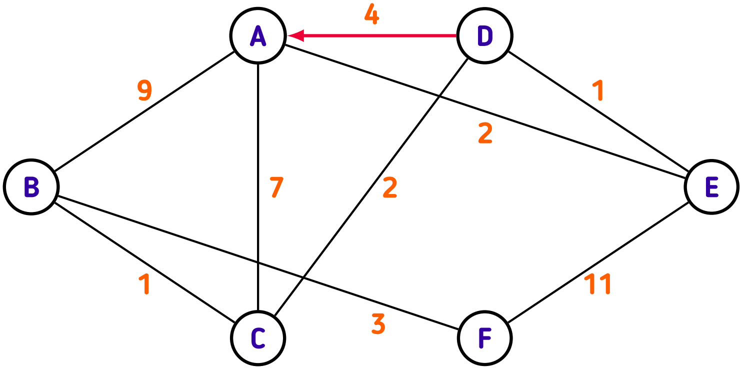
	A	B	C	D	E	F
dist$(u,$ **A**$)$	0	9	7	4	2	12



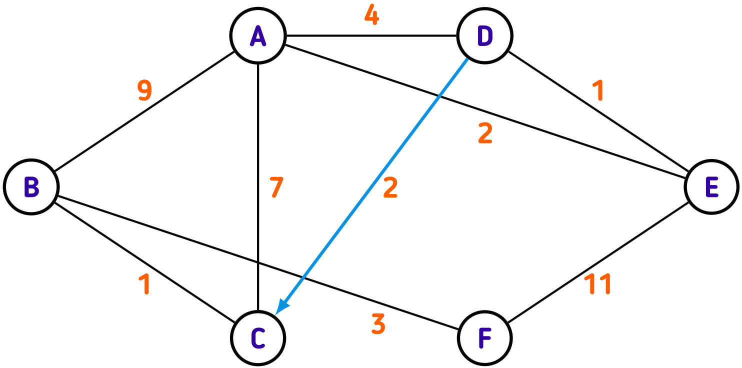
	A	B	C	D	E	F
dist(u, **A**)	0	**8**	7	4	2	12



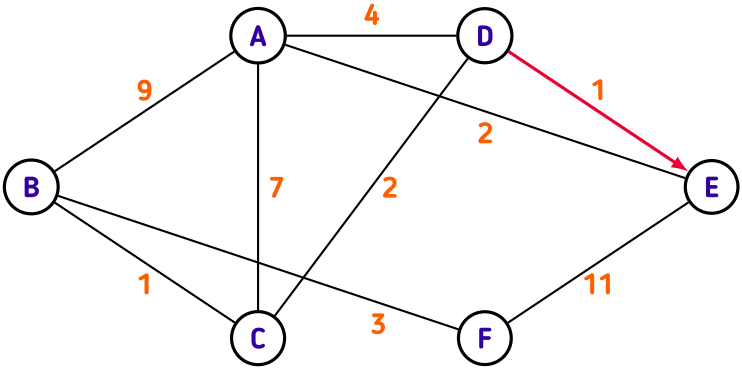
	A	B	C	D	E	F
$\mathsf{dist}(u, \mathbf{A})$	0	8	7	4	2	12



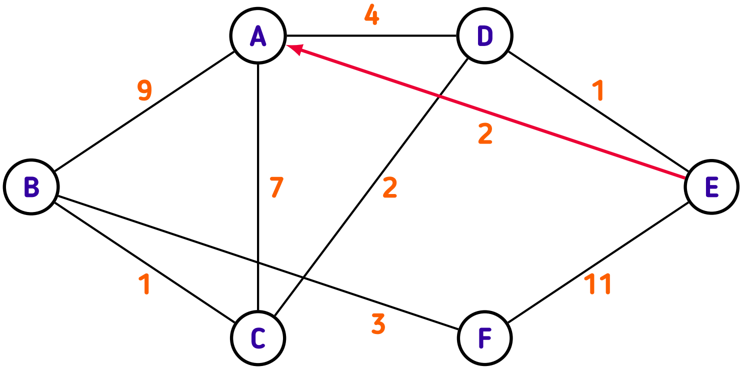
	A	B	C	D	E	F
dist$(u,$ **A**$)$	0	8	7	4	2	12



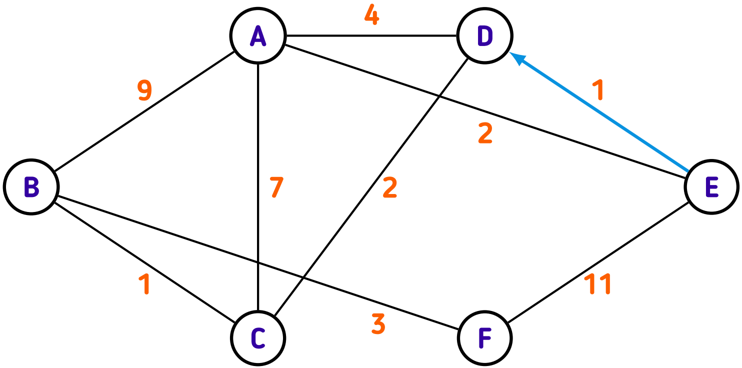
	A	B	C	D	E	F
dist$(u,$ **A**$)$	0	8	**6**	4	2	12



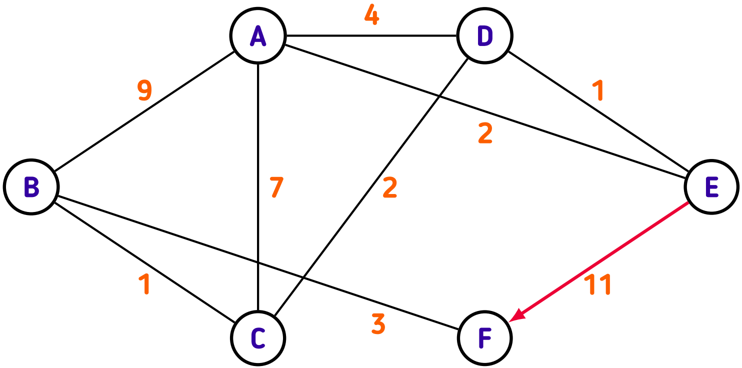
	A	B	C	D	E	F
dist$(u, \textbf{A})$	0	8	6	4	2	12



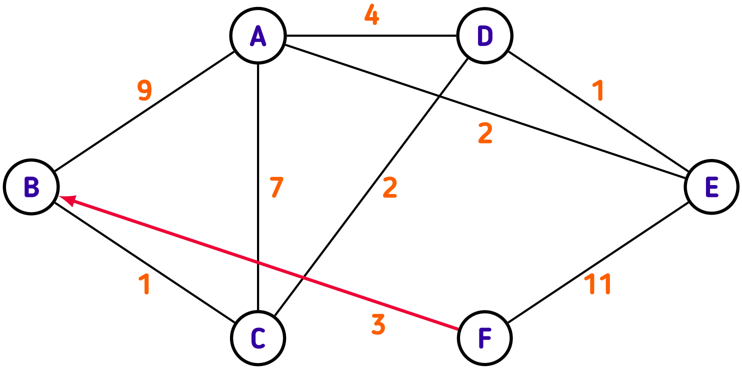
	A	B	C	D	E	F
dist(u, **A**)	0	8	6	4	2	12



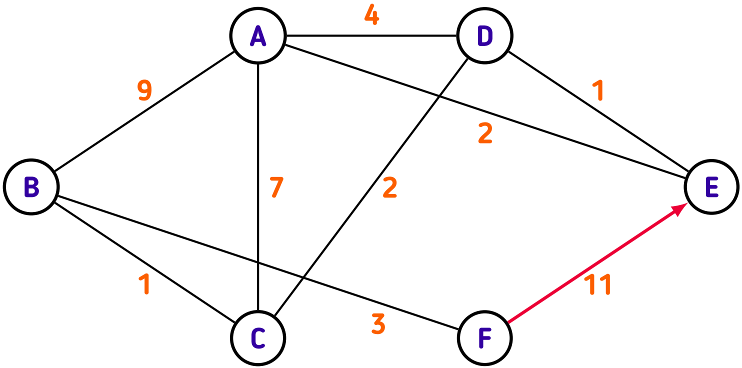
	A	B	C	D	E	F
dist(u, **A**)	0	8	6	**3**	2	12



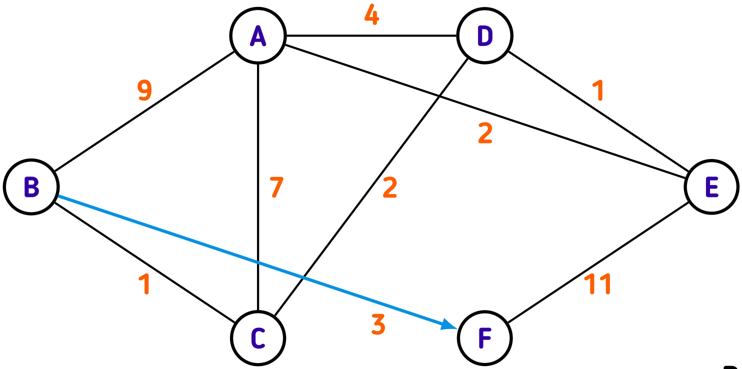
	A	B	C	D	E	F
dist$(u, \mathbf{A})$	0	8	6	3	2	12



	A	B	C	D	E	F
dist$(u,$ **A**$)$	0	8	6	3	2	12

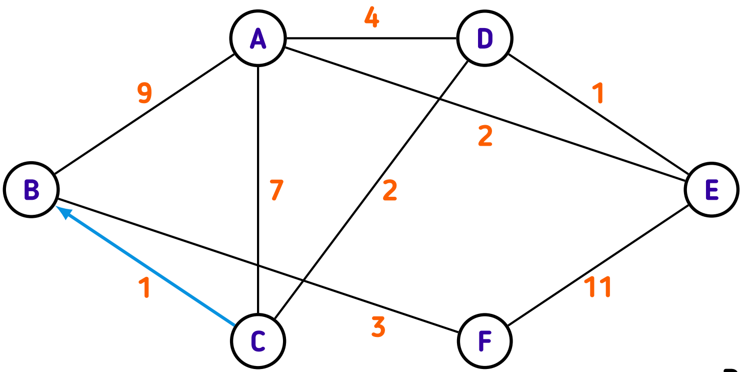


	A	B	C	D	E	F
dist$(u,$ **A**$)$	0	8	6	3	2	12



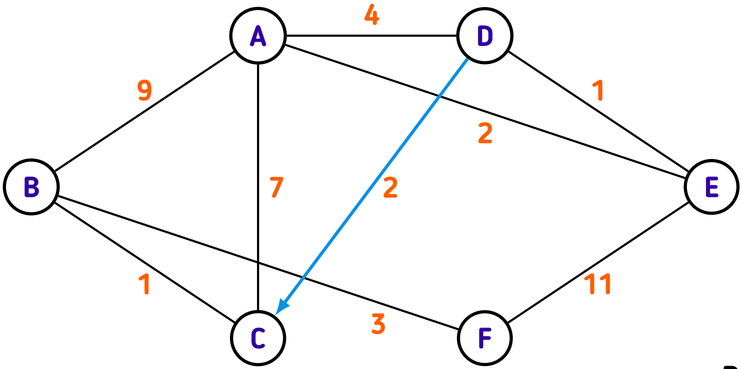
Round #2

	A	B	C	D	E	F
dist(u, **A**)	0	8	6	3	2	**11**



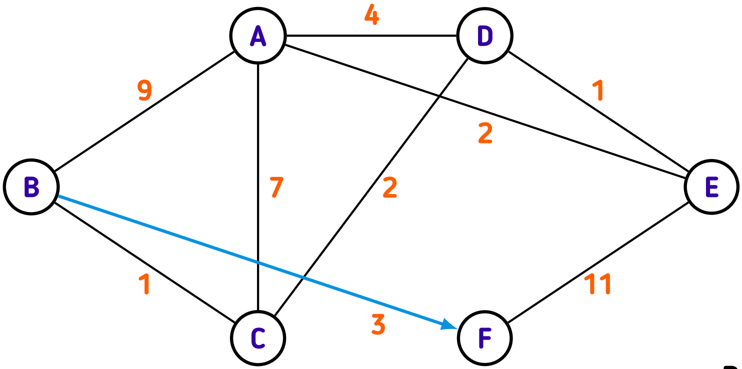
Round #2

	A	B	C	D	E	F
dist(u, **A**)	0	**7**	6	3	2	11



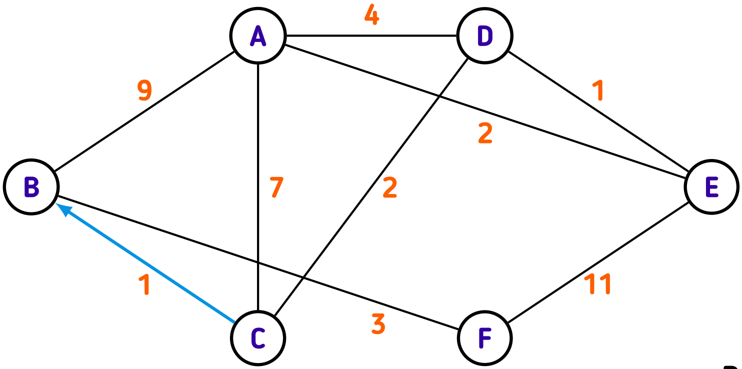
Round #2

dist(u, **A**)	A	B	C	D	E	F
	0	7	**5**	3	2	11



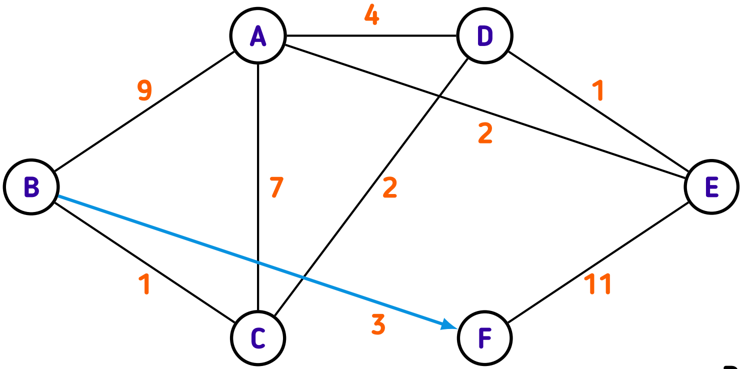
Round #3

	A	B	C	D	E	F
dist$(u, \textbf{A})$	0	7	5	3	2	**10**



Round #3

	A	B	C	D	E	F
dist(u, **A**)	0	**6**	5	3	2	10



Round #4

	A	B	C	D	E	F
dist$(u,$ **A**$)$	0	6	5	3	2	**9**

```cpp
vector<int> bellman_ford(int s, int N, const vector<edge>& edges)
{
    const int oo { 1000000010 };

    vector<int> dist(N + 1, oo);
    dist[s] = 0;

    for (int i = 1; i <= N - 1; i++)
        for (auto [u, v, w] : edges)
            if (dist[u] < oo and dist[v] > dist[u] + w)
                dist[v] = dist[u] + w;

    return dist;
```

Identificação do caminho mínimo

Identificação do caminho mínimo

- O algoritmo de Bellman-Ford computa as distâncias mínimas, mas não os caminhos mínimos

Identificação do caminho mínimo

$\star$ **O algoritmo de Bellman-Ford computa as distâncias mínimas, mas não os caminhos mínimos**

$\star$ **Para determinar um caminho mínimo, é preciso definir o vetor auxiliar** pred**, onde** $\mathsf{pred}[u] =$ **antecessor de** u **no caminho mínimo de** s **a** u

Identificação do caminho mínimo

$\star$ **O algoritmo de Bellman-Ford computa as distâncias mínimas, mas não os caminhos mínimos**

$\star$ **Para determinar um caminho mínimo, é preciso definir o vetor auxiliar** pred**, onde** $\mathsf{pred}[u] =$ **antecessor de** u **no caminho mínimo de** s **a** u

$\star$ **No início do algoritmo,** $\mathsf{pred}[s] = s$ **e** $\mathsf{pred}[u] = \mathsf{undef}$**, se** $u \neq s$

Identificação do caminho mínimo

- **Se** (u, v) **atualizar** $d[v]$**, faça** $\mathsf{pred}[v] = u$

Identificação do caminho mínimo

- **Se** (u, v) **atualizar** $d[v]$**, faça** $\mathsf{pred}[v] = u$
- **A sequência**

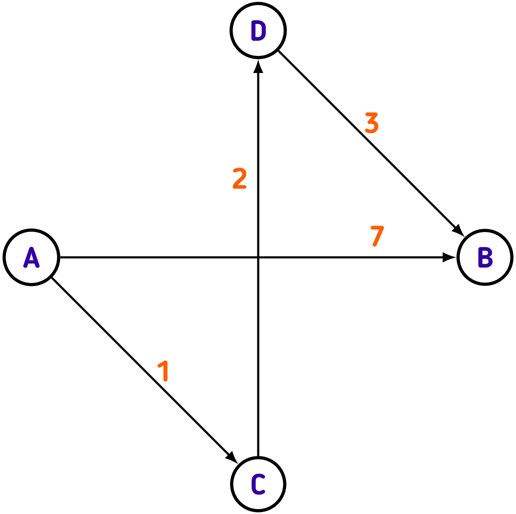
$$p = \{(s, \mathsf{pred}^{k-1}[u]), \ldots, (\mathsf{pred}[\mathsf{pred}[u]], \mathsf{pred}[u]), (\mathsf{pred}[u], u)\}$$

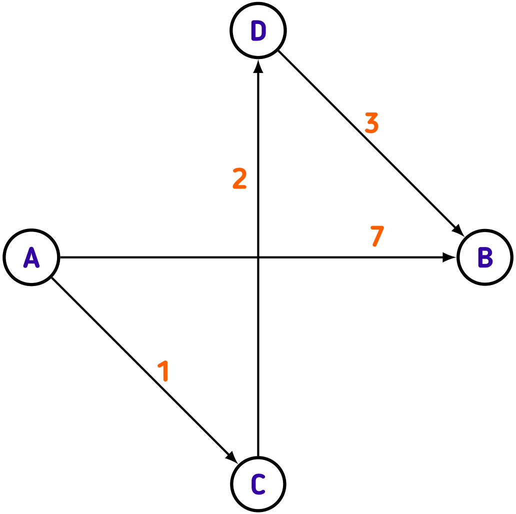
Identificação do caminho mínimo

- $\star$ **Se** (u, v) **atualizar** $d[v]$**, faça** $\mathsf{pred}[v] = u$
- $\star$ **A sequência**

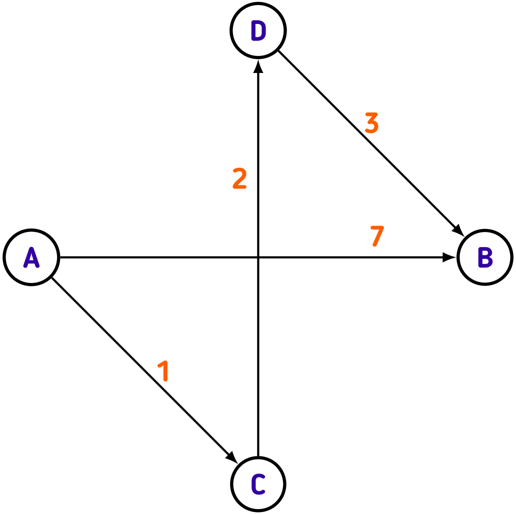
$$p = \{(s, \mathsf{pred}^{k-1}[u]), \ldots, (\mathsf{pred}[\mathsf{pred}[u]], \mathsf{pred}[u]), (\mathsf{pred}[u], u)\}$$

é um caminho mínimo de s **a** u **composto de** k **arestas e tamanho** $d[u]$


D
2
3
A
7
B
1
C

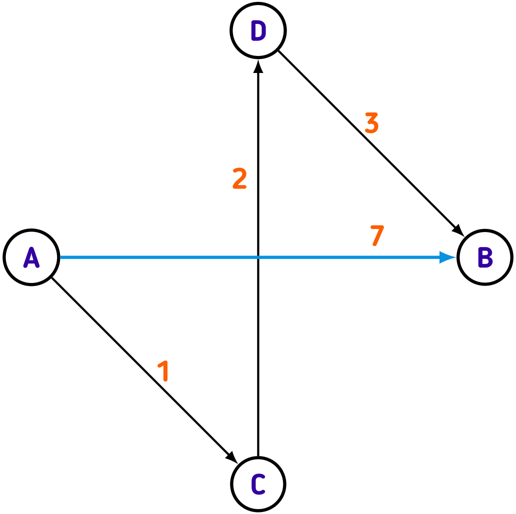


	A	B	C	D
$\text{dist}(u, \textbf{A})$	0	∞	∞	∞



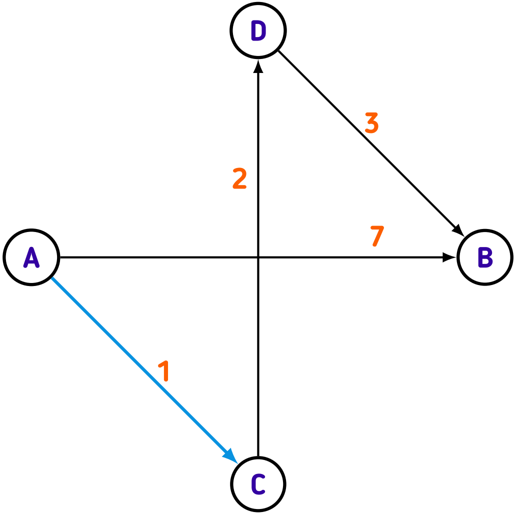
	A	B	C	D
$\text{dist}(u, \textbf{A})$	0	∞	∞	∞

	A	B	C	D
$\text{pred}(u)$	A	-	-	-



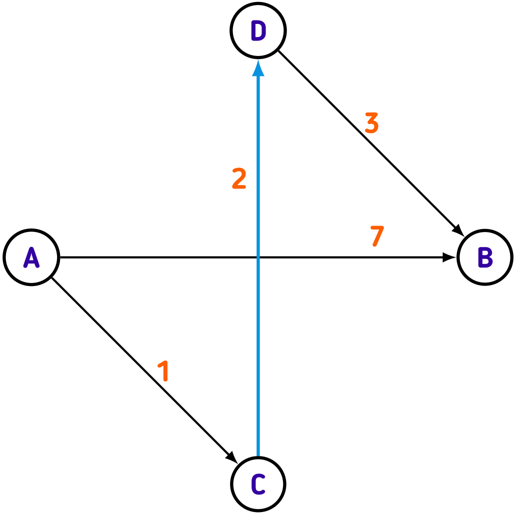
	A	B	C	D
$\text{dist}(u, \textbf{A})$	0	**7**	∞	∞

	A	B	C	D
$\text{pred}(u)$	**A**	**A**	-	-



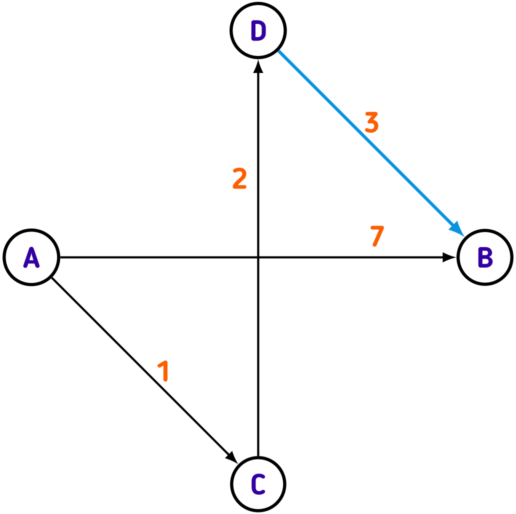
	A	B	C	D
dist$(u,$ **A**$)$	0	7	1	∞

	A	B	C	D
pred(u)	A	A	A	-



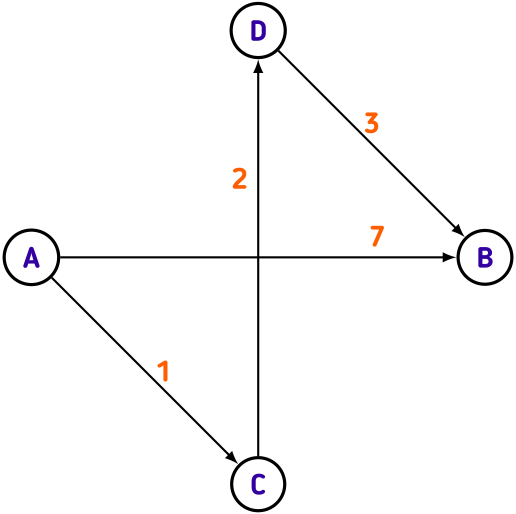
	A	B	C	D
dist$(u, \textbf{A})$	0	7	1	**3**

	A	B	C	D
pred(u)	**A**	**A**	**A**	**C**



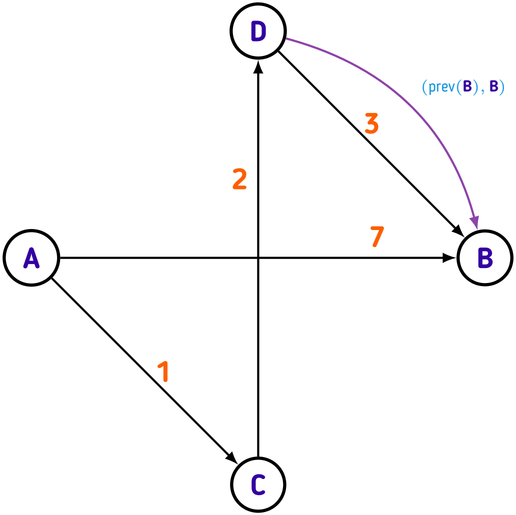
	A	B	C	D
$\mathsf{dist}(u, \mathbf{A})$	0	**6**	1	3

	A	B	C	D
$\mathsf{pred}(u)$	A	D	A	C



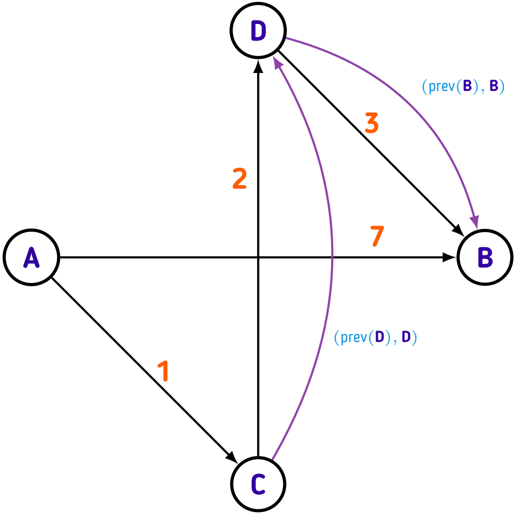
	A	B	C	D
dist$(u, \textbf{A})$	0	6	1	3

	A	B	C	D
pred(u)	A	D	A	C



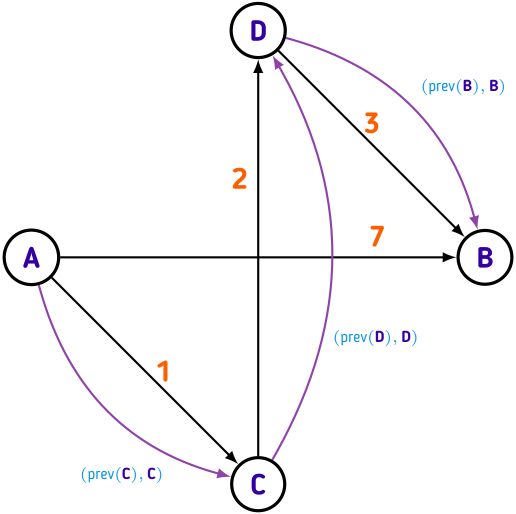
	A	B	C	D
$\text{dist}(u, \textbf{A})$	0	6	1	3

	A	B	C	D
$\text{pred}(u)$	A	D	A	C



	A	B	C	D
dist$(u,$ A$)$	0	6	1	3

	A	B	C	D
pred(u)	A	D	A	C



	A	B	C	D
dist$(u,$ A$)$	0	6	1	3

	A	B	C	D
pred(u)	A	D	A	C

```cpp
pair<vector<int>, vector<int>>
bellman_ford(int s, int N, const vector<edge>& edges)
{
    vector<int> dist(N + 1, oo), pred(N + 1, oo);

    dist[s] = 0;
    pred[s] = s;

    for (int i = 1; i <= N - 1; i++)
        for (auto [u, v, w] : edges)
            if (dist[u] < oo and dist[v] > dist[u] + w) {
                dist[v] = dist[u] + w;
                pred[v] = u;
            }

    return { dist, pred };
}
```

```cpp
vector<ii> path(int s, int u, const vector<int>& pred)
{
    vector<ii> p;
    int v = u;

    do {
        p.push_back(ii(pred[v], v));
        v = pred[v];
    } while (v != s);

    reverse(p.begin(), p.end());

    return p;
}
```

Caminhos mínimos e ciclos

Caminhos mínimos e ciclos

Seja

$$p = \{(a, u_1), (u_1, u_2), \ldots, (v, u_r), \ldots, (u_s, v), \ldots, (u_t, b)\}$$

um caminho de a a b e $\omega(c)$ o custo do ciclo $c = \{(v, u_r), \ldots, (u_s, v)\}$, isto é

$$\omega(c) = \sum_{e \in c} w(e)$$

Caminhos mínimos e ciclos

Seja

$$p = \{(a, u_1), (u_1, u_2), \ldots, (v, u_r), \ldots, (u_s, v), \ldots, (u_t, b)\}$$

um caminho de a a b e $\omega(c)$ o custo do ciclo $c = \{(v, u_r), \ldots, (u_s, v)\}$, isto é

$$\omega(c) = \sum_{e \in c} w(e)$$

custo da aresta e

Caminhos mínimos e ciclos

Seja

$$p = \{(a, u_1), (u_1, u_2), \dots, (v, u_r), \dots, (u_s, v), \dots, (u_t, b)\}$$

um caminho de a **a** b **e** $\omega(c)$ **o custo do ciclo** $c = \{(v, u_r), \dots, (u_s, v)\}$**, isto é**

$$\omega(c) = \sum_{e \in c} w(e)$$

custo da aresta e

Se p **é caminho mínimo de** a **a** b **então** $\omega(c) = 0$**.**

Caminhos mínimos e ciclos positivos

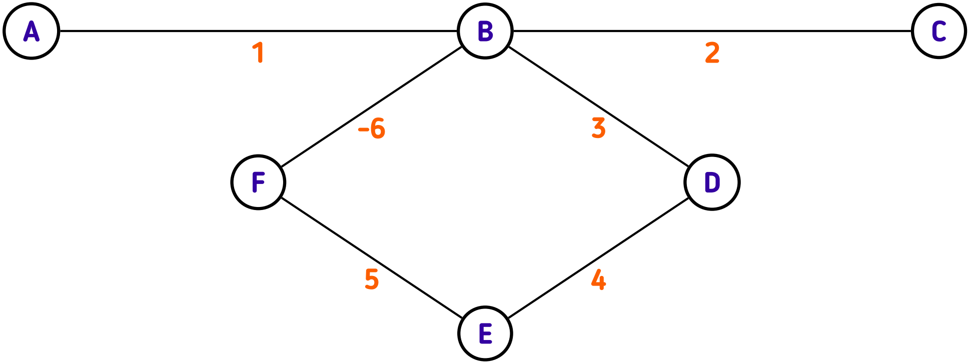
Caminhos mínimos e ciclos positivos

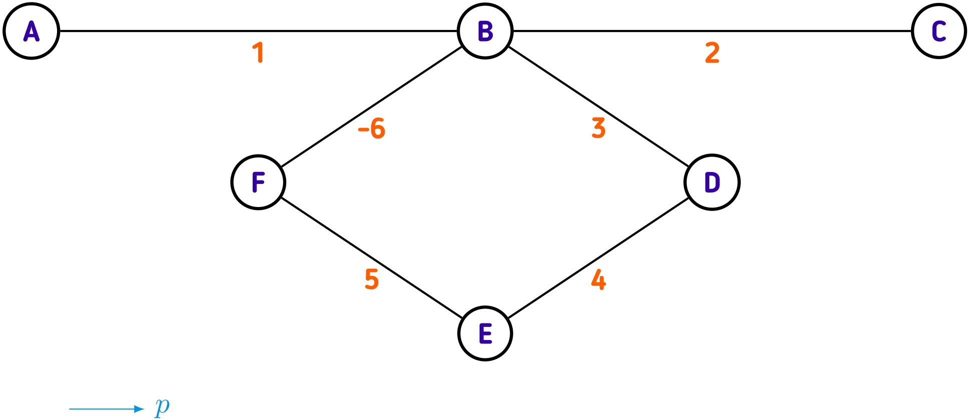
Seja $\omega(c) > 0$ **e**

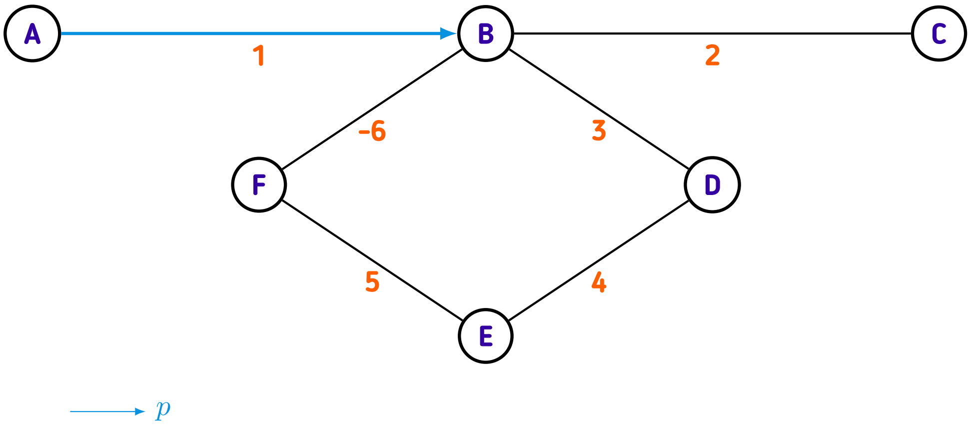
$$q = \{(a, u_1), (u_1, u_2), \ldots, (u_{r-1}, v), (v, u_{s+1}), \ldots, (u_t, b)\},$$

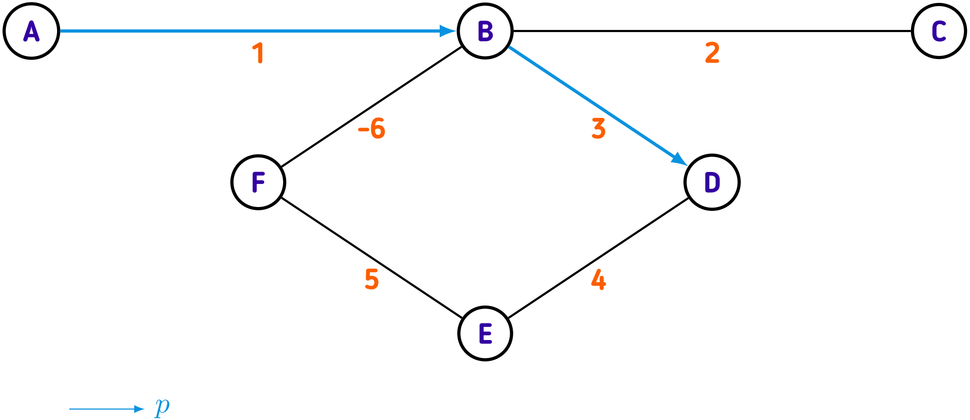
o caminho resultante da exclusão do ciclo c **de** p. **Então** $\omega(q) < \omega(p)$, **pois**

$$\omega(p) = \sum_{e_i \in p} w(e_i) = \sum_{e_j \in q} w(e_j) + \sum_{e_k \in c} w(e_k) = \omega(q) + \omega(c) > \omega(q)$$

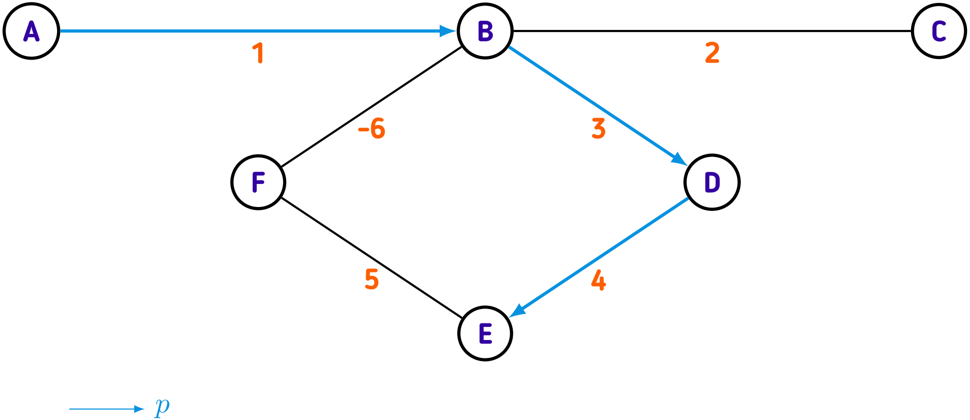

A
B
C
D
E
F
1
2
-6
3
5
4

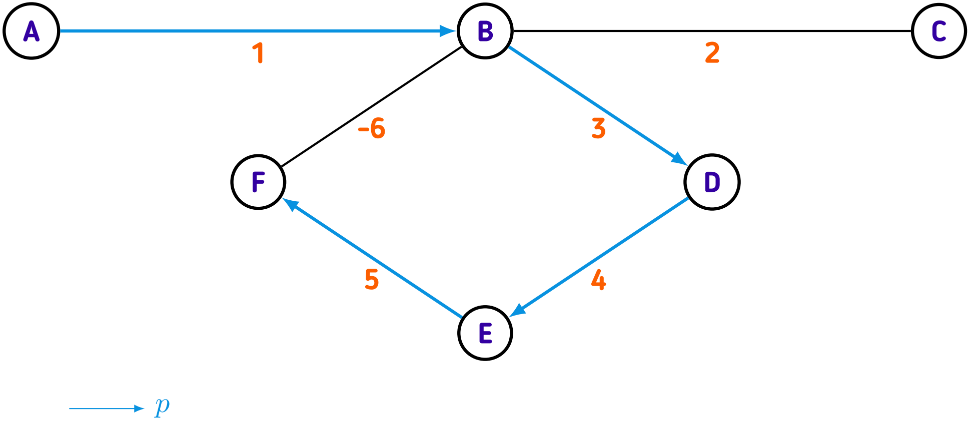

A
B
C
F
D
E
1
2
-6
3
5
4
p

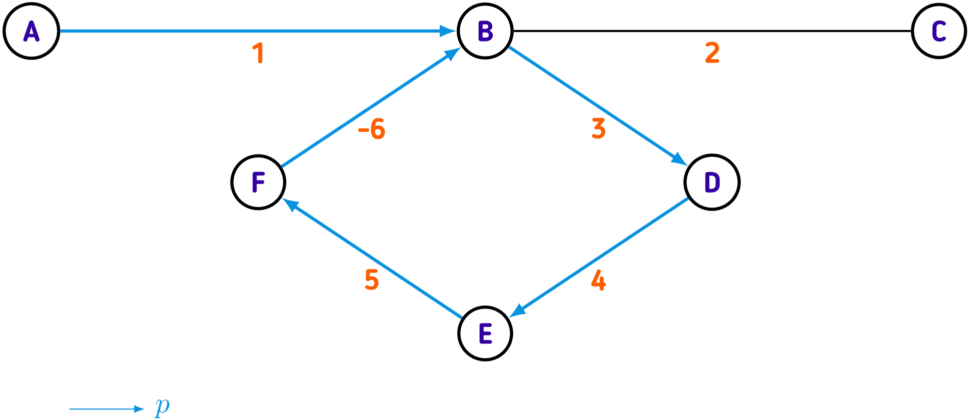

A
B
C
D
E
F
1
2
3
4
5
-6
p

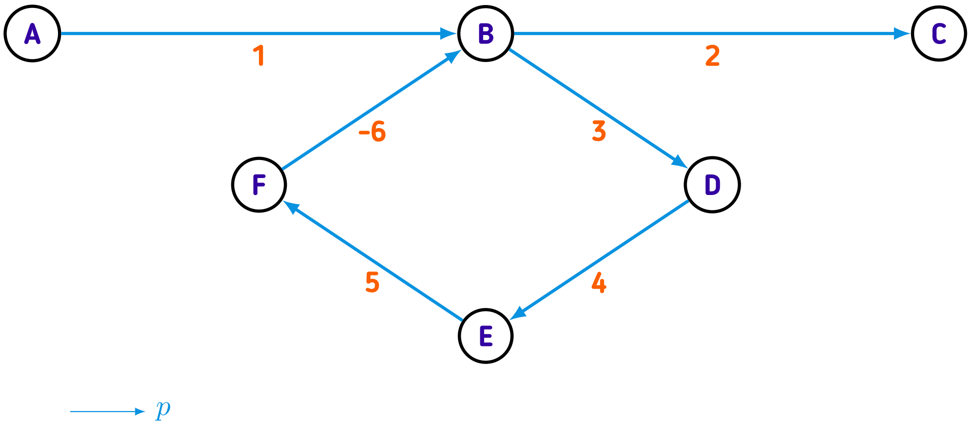


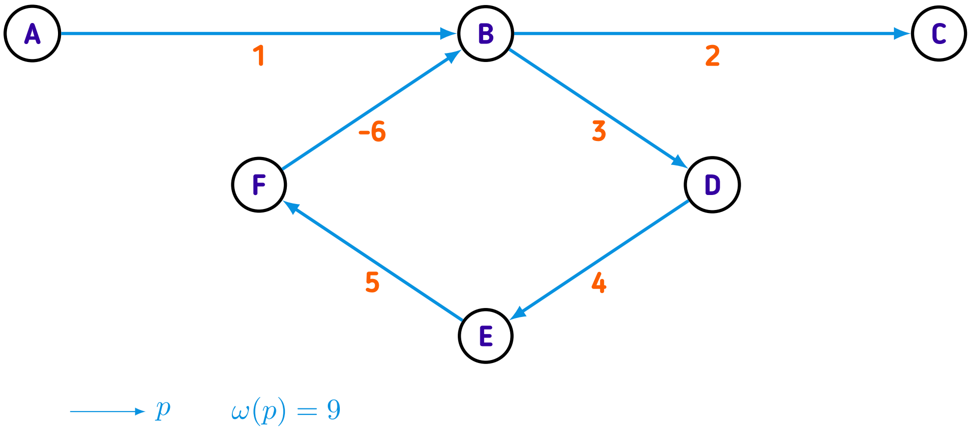
A
B
C
D
E
F
1
2
-6
3
5
4
p

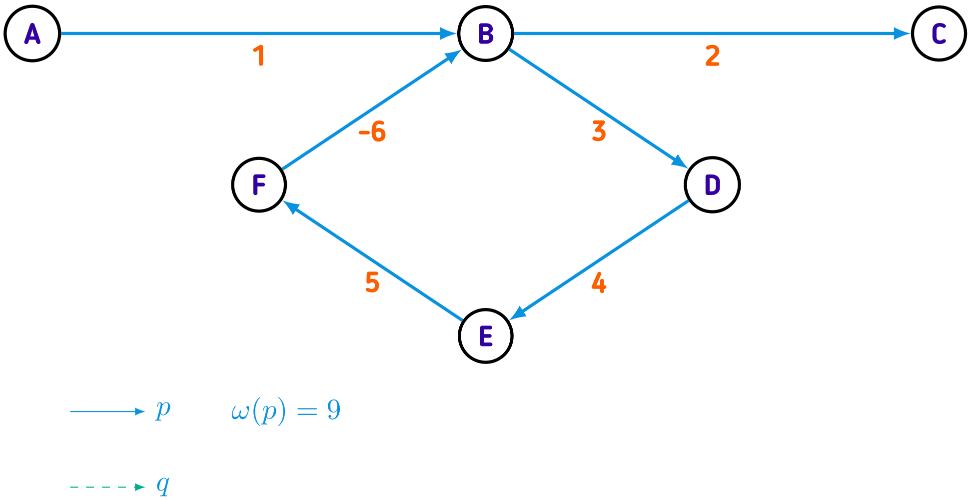

A
B
C
1
2
-6
3
F
D
5
4
E
p

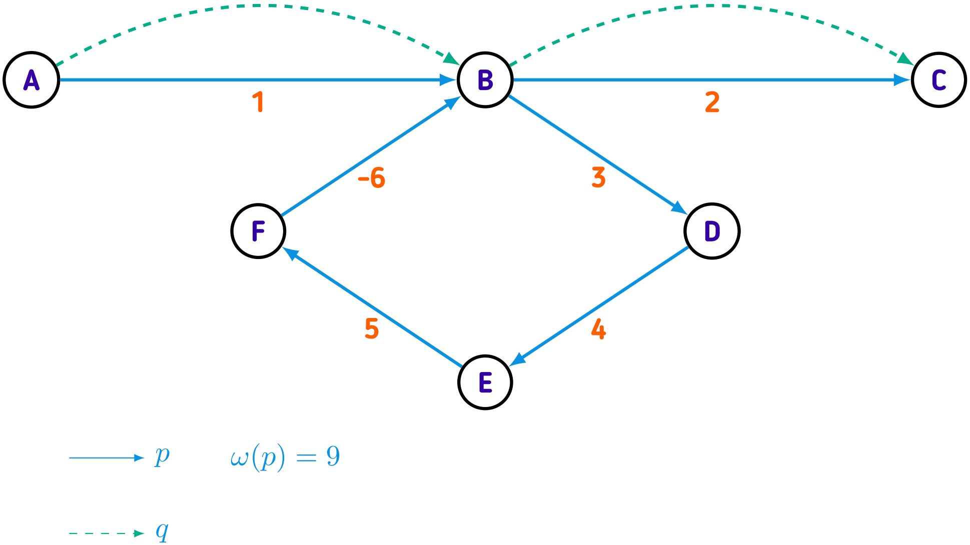

A
B
C
D
E
F
1
2
3
4
5
-6
p

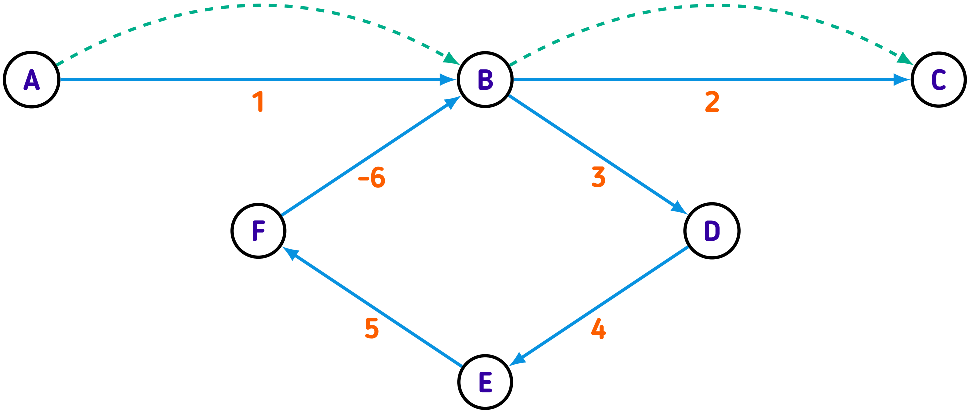

A
B
C
D
E
F
1
2
3
4
5
-6
p


A
B
C
D
E
F
1
2
3
4
5
-6
p


A
B
C
D
E
F
1
2
3
4
5
-6
p
$\omega(p) = 9$


A
B
C
D
E
F
1
2
3
4
5
-6
p
$\omega(p) = 9$
q


A
B
C
D
E
F
1
2
3
4
5
-6
p
$\omega(p) = 9$
q



$p \qquad \omega(p) = 9$

$q \qquad \omega(q) = 3$

Caminhos mínimos e ciclos negativos

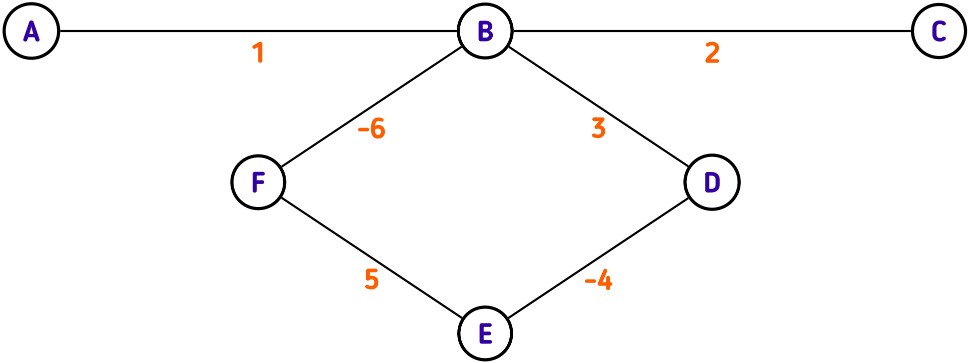
Caminhos mínimos e ciclos negativos

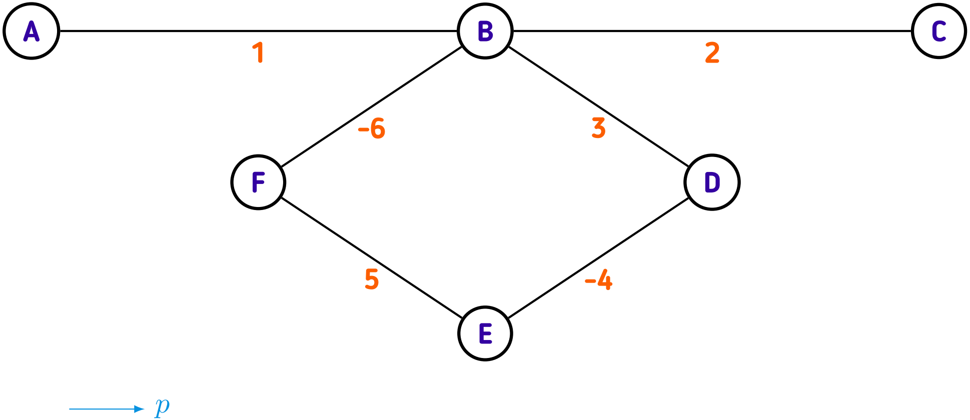
Seja $\omega(c) < 0$ **e**

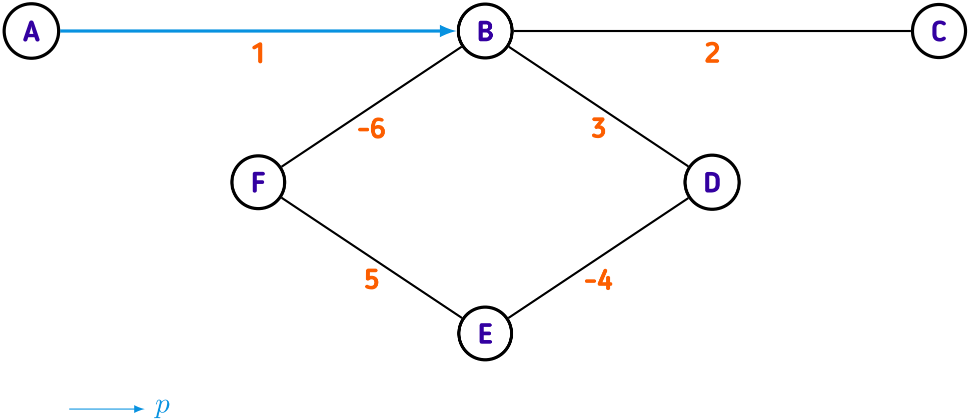
$$q = \{(a, u_1), (u_1, u_2), \ldots, (v, u_r), \ldots, (u_s, v), (v, u_r), \ldots, (u_s, v), \ldots, (u_t, b)\}$$

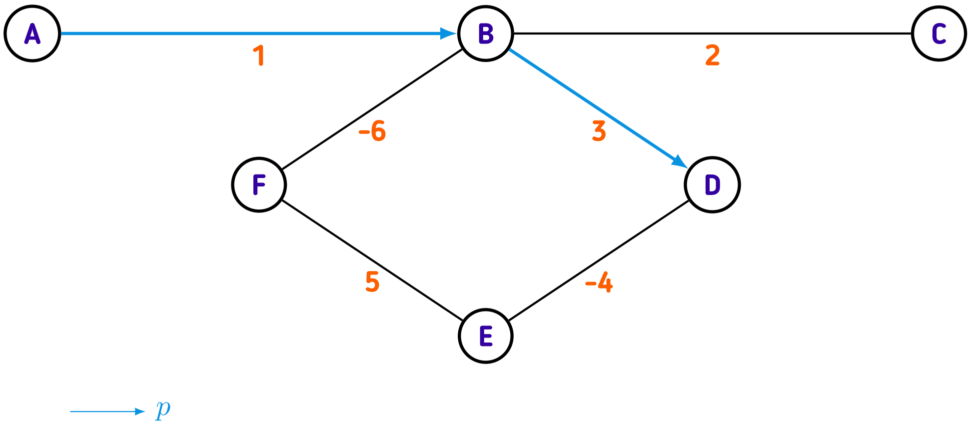
o caminho resultante da duplicação do ciclo c **de** p**. Então** $\omega(q) < \omega(p)$**, pois**

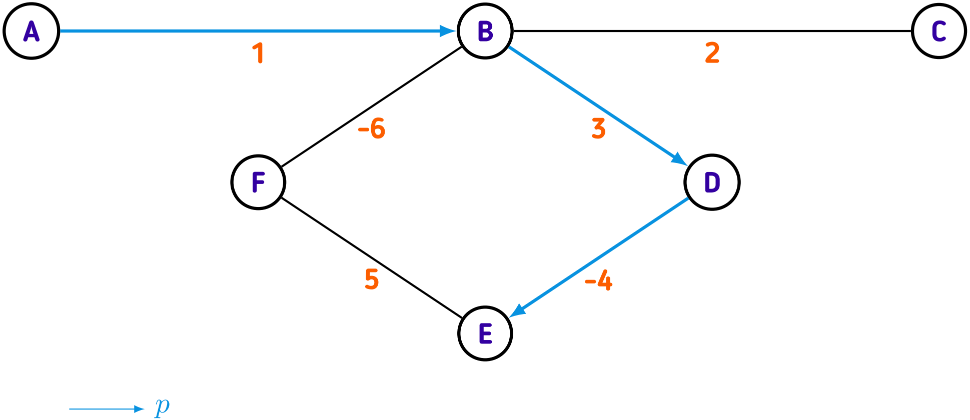
$$\omega(q) = \sum_{e_i \in q} w(e_i) = \sum_{e_j \in p} w(e_j) + \sum_{e_k \in c} w(e_k) = \omega(p) + \omega(c) < \omega(p)$$

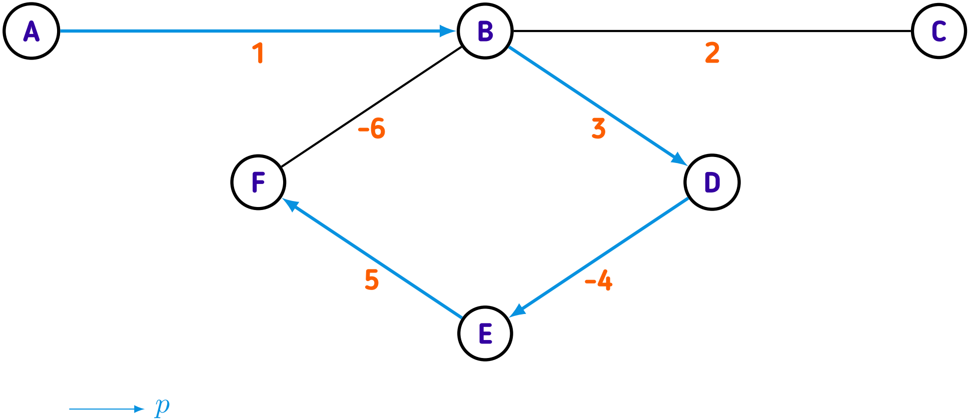

A
B
C
1
2
-6
3
F
D
5
-4
E

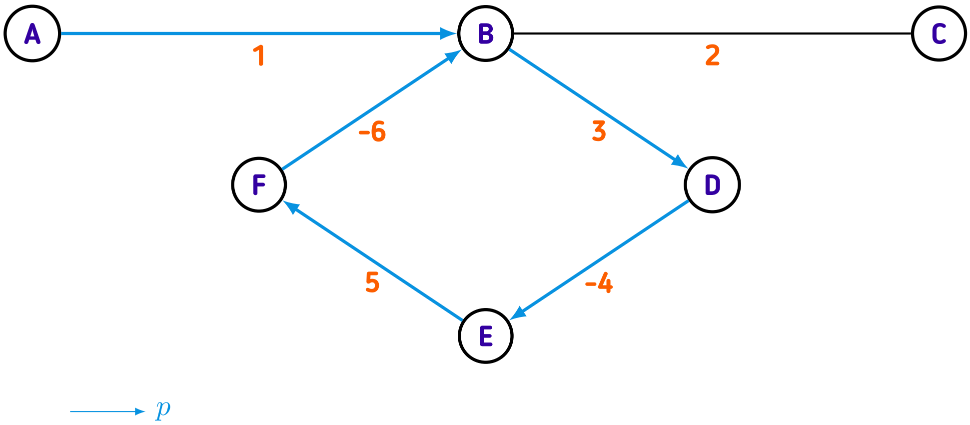

A
B
C
1
2
-6
3
F
D
5
-4
E
p

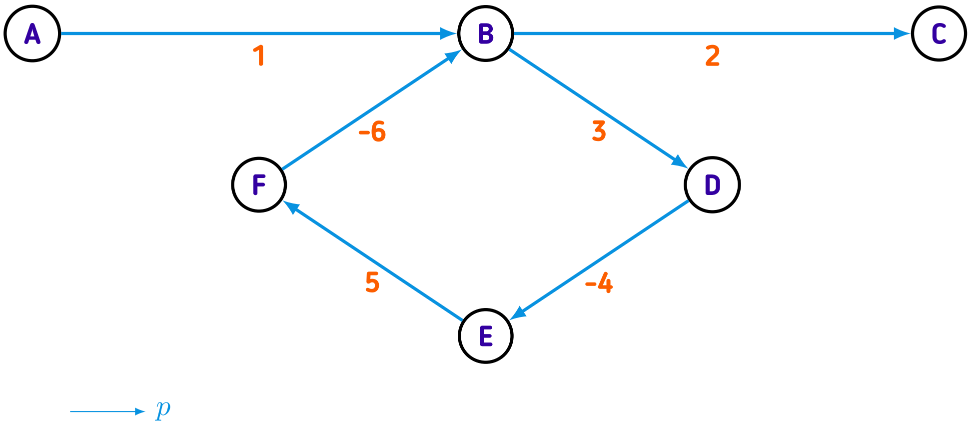

A
B
C
D
E
F
1
2
3
-6
5
-4
p

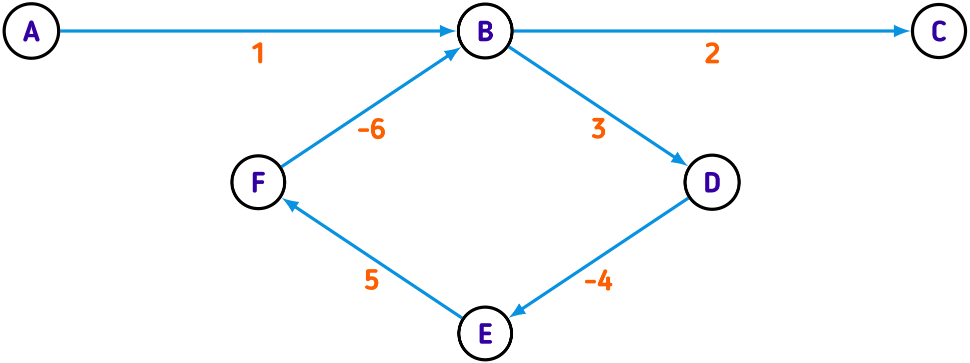

A
B
C
F
D
E
1
2
-6
3
5
-4
p


A
B
C
D
E
F
1
2
-6
3
5
-4
p

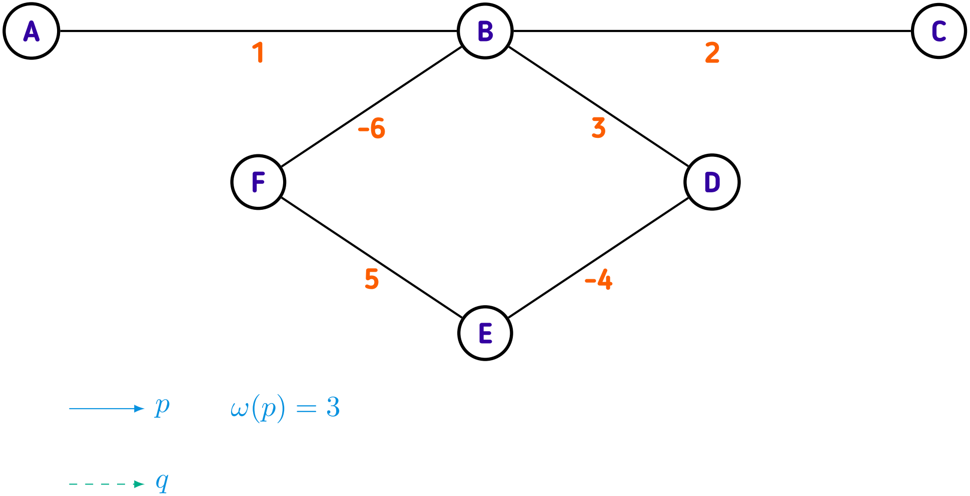

A
B
C
D
E
F
1
2
-6
3
5
-4
p

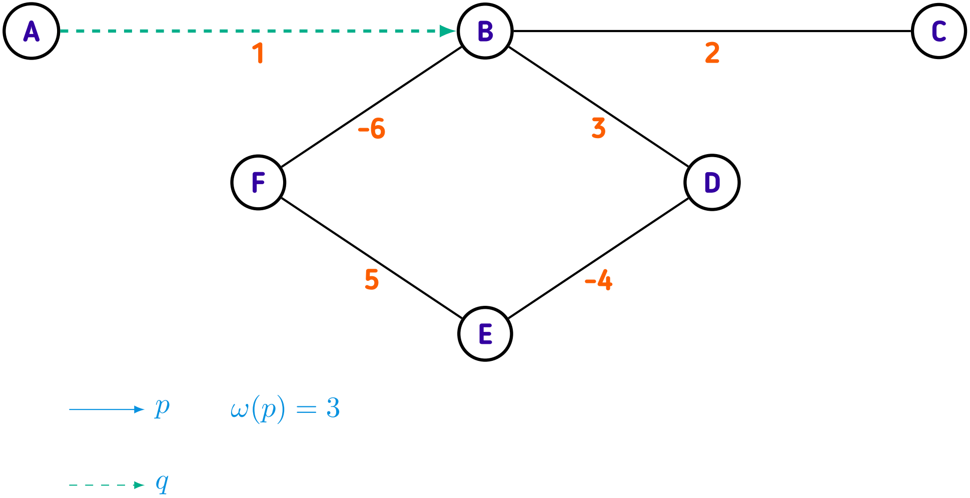

A
B
C
D
E
F
1
2
3
-4
5
-6
p


A
B
C
D
E
F
1
2
3
-4
5
-6
p

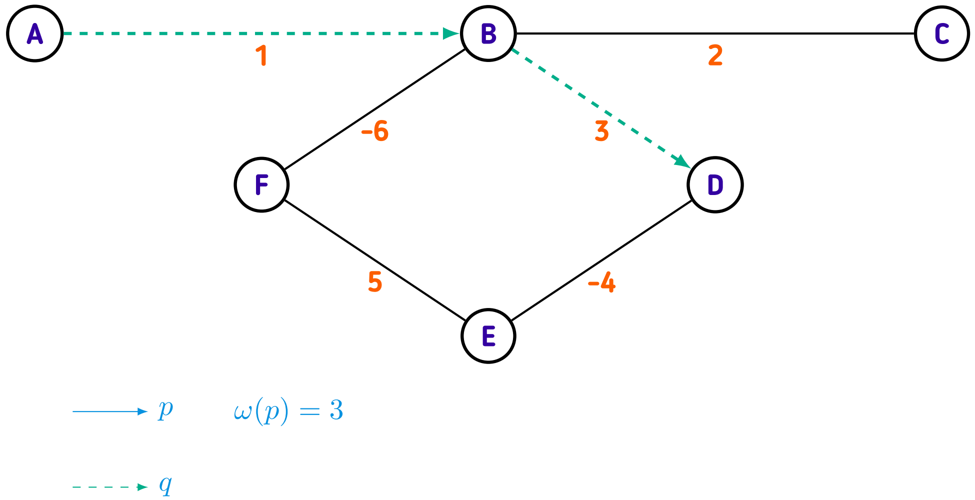


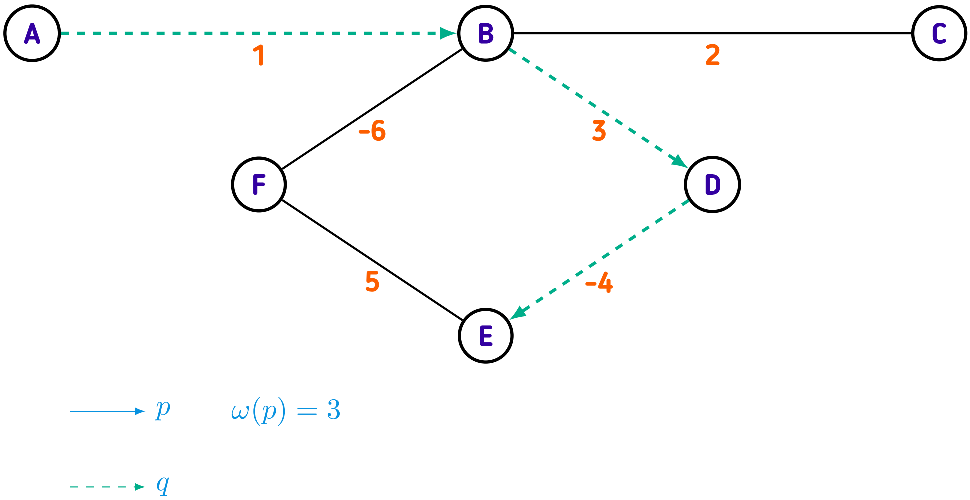
$\longrightarrow p \qquad \omega(p) = 3$


A
B
C
D
E
F
1
2
-6
3
5
-4
p
ω(p) = 3
q

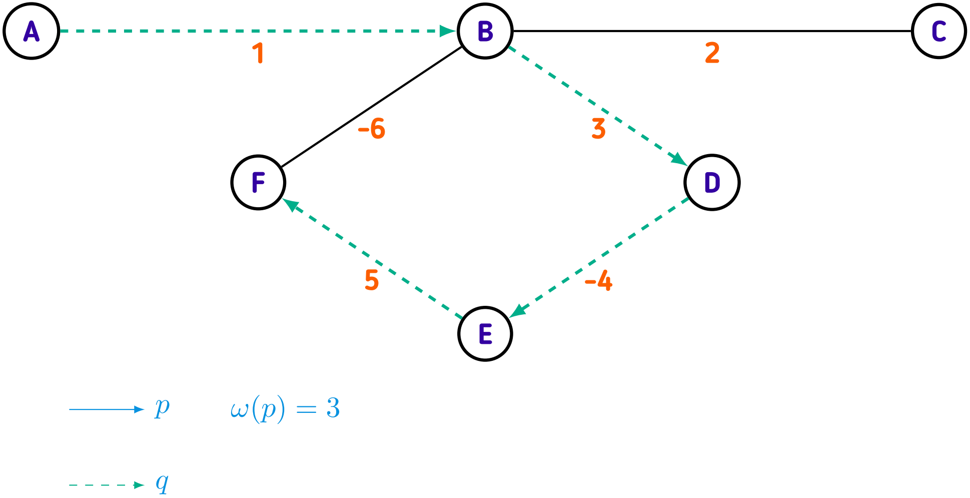


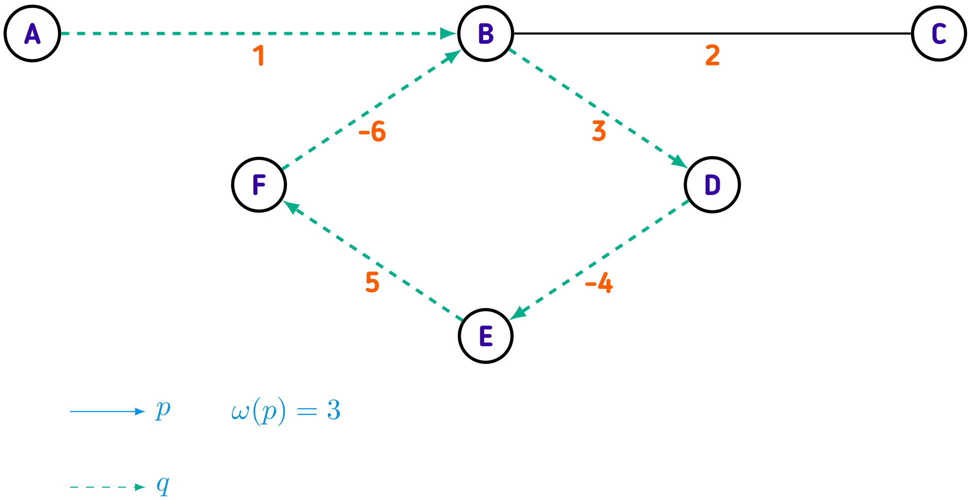
A
B
C
1
2
-6
3
F
D
5
-4
E
p
ω(p) = 3
q

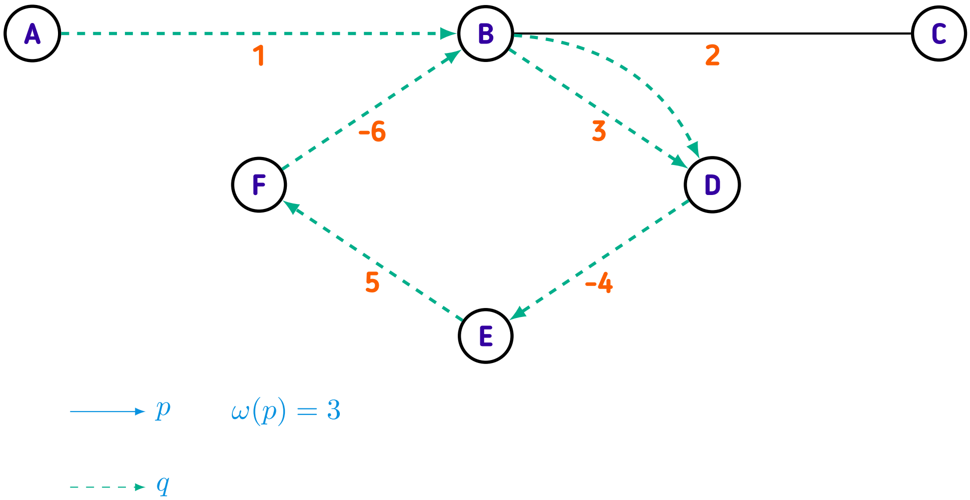

A
B
C
D
E
F
1
2
-6
3
5
-4
p
ω(p) = 3
q

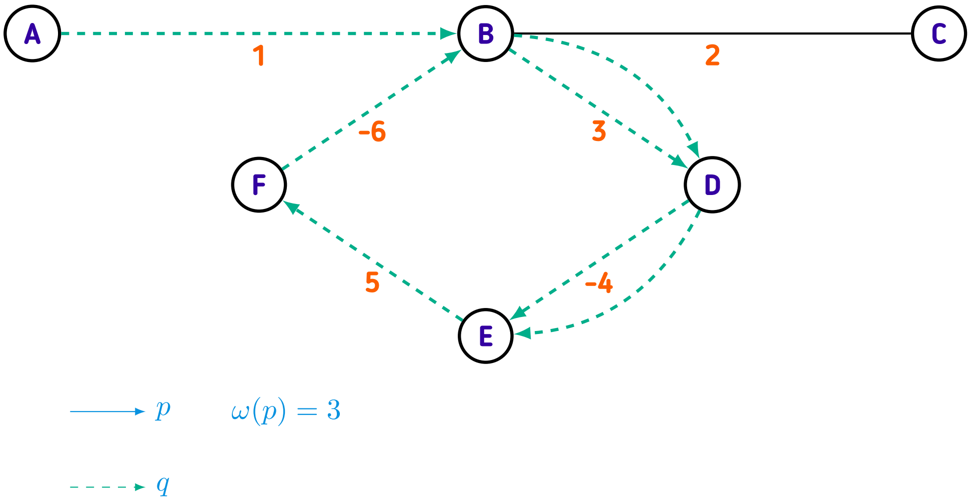


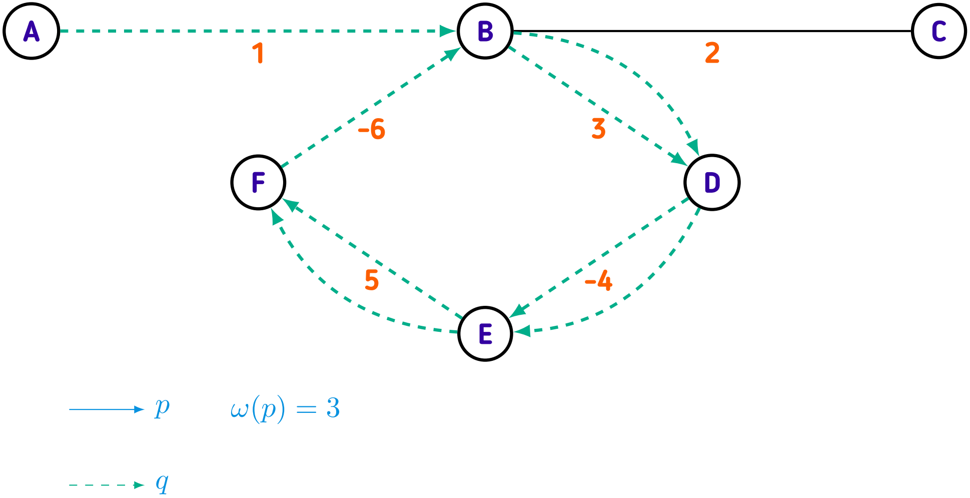
A
B
C
D
E
F
1
2
-6
3
5
-4
p
$\omega(p) = 3$
q

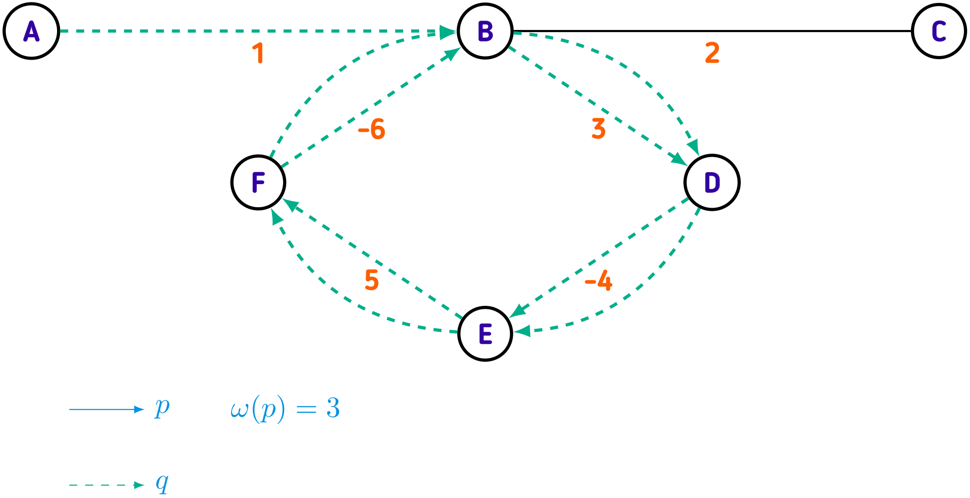

A
B
C
D
E
F
1
2
3
-4
5
-6
p
ω(p) = 3
q

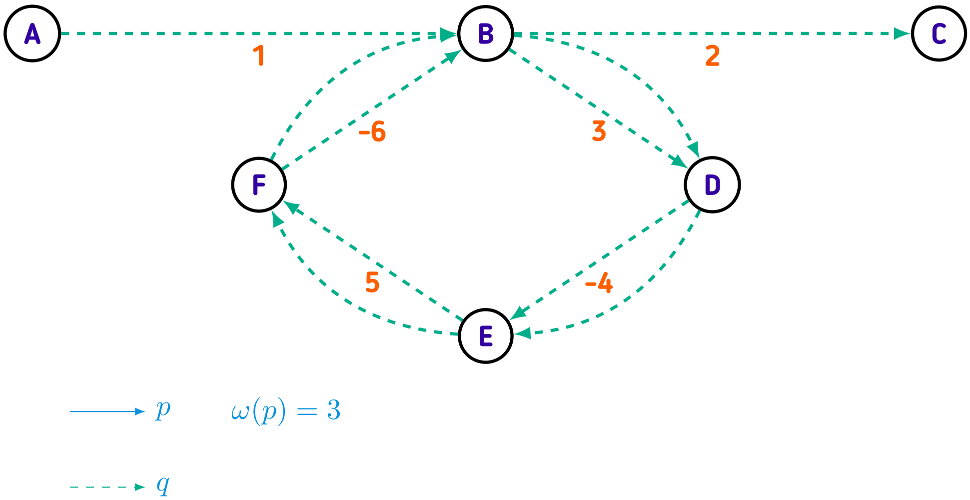

A
B
C
D
E
F
1
2
3
-4
5
-6
p
$\omega(p) = 3$
q

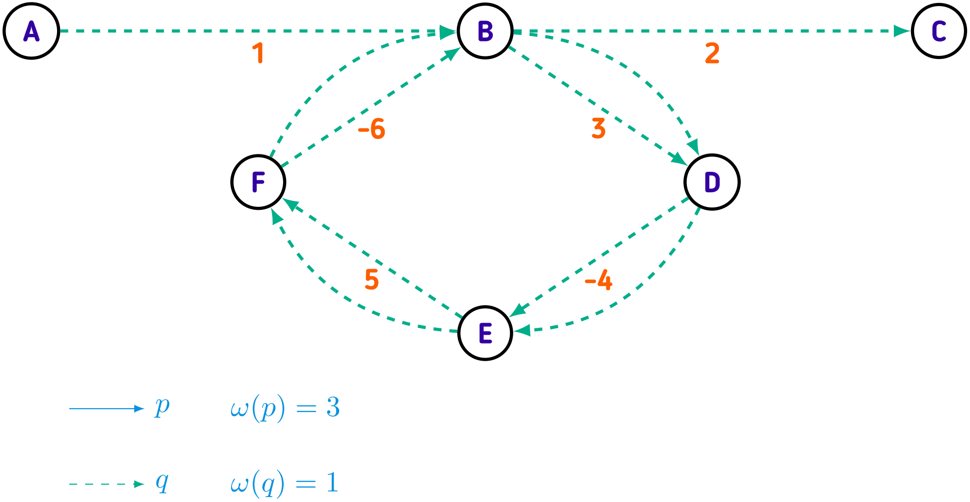

A
B
C
D
E
F
1
2
3
-4
5
-6
p
ω(p) = 3
q


A
B
C
D
E
F
1
2
3
-6
-4
5
p
$\omega(p) = 3$
q


A
B
C
D
E
F
1
2
3
-6
-4
5
p
$\omega(p) = 3$
q


A
B
C
D
E
F
1
2
3
-6
5
-4
p
ω(p) = 3
q


A
B
C
D
E
F
1
2
3
-6
-4
5
p
ω(p) = 3
q


A
B
C
1
2
-6
3
F
D
5
-4
E
p
ω(p) = 3
q
ω(q) = 1

Número de rodadas do algoritmo de Bellman-Ford

Número de rodadas do algoritmo de Bellman-Ford

Teorema. Seja $G(V, E)$ um grafo cujos pesos de suas arestas sejam todos não-negativos. Então para qualquer $v \in V$, o caminho mínimo de s a u identificado pelo algoritmo de Bellman-Ford tem, no máximo, $|V| - 1$ arestas.

Detecção de ciclos negativos

Detecção de ciclos negativos

Teorema. **Seja $G(V, E)$ um grafo. Se a $|V|$-ésima rodada do algoritmo de Bellman-Ford atualizar o vetor d ao menos uma vez, então G possui pelo menos um ciclo negativo.**

```cpp
bool has_negative_cycle(int s, int N, const vector<edge>& edges)
{
    const int oo { 1000000010 };

    vector<int> dist(N + 1, oo);
    dist[s] = 0;

    for (int i = 1; i <= N - 1; i++)
        for (auto [u, v, w] : edges)
            if (dist[u] < oo and dist[v] > dist[u] + w)
                dist[v] = dist[u] + w;

    for (auto [u, v, w] : edges)
        if (dist[u] < oo and dist[v] > dist[u] + w)
            return true;

    return false;
}
```

Problemas sugeridos

1. AtCoder Beginner Contest 137 – Problem E: Coin Respawn
2. CSES 1673 – High Score
3. OJ 423 – MPI Maelstrom
4. OJ 534 – Frogger

Referências

1. **HALIM**, **Felix**; **HALIM**, **Steve**. *Competitive Programming 3*, **2010.**
2. **LAAKSONEN**, **Antti**. *Competitive Programmer's Handbook*, **2018.**
3. **SKIENA**, **Steven**; **REVILLA**, **Miguel**. *Programming Challenges*, **2003.**
4. **Wikipédia**, *Bellman-Ford algorithm.* **Acesso em 07/07/2021.**
5. **Wikipédia**, *L. R. Ford Jr.* **Acesso em 07/07/2021.**
6. **Wikipédia**, *Richard E. Bellman.* **Acesso em 07/07/2021.**