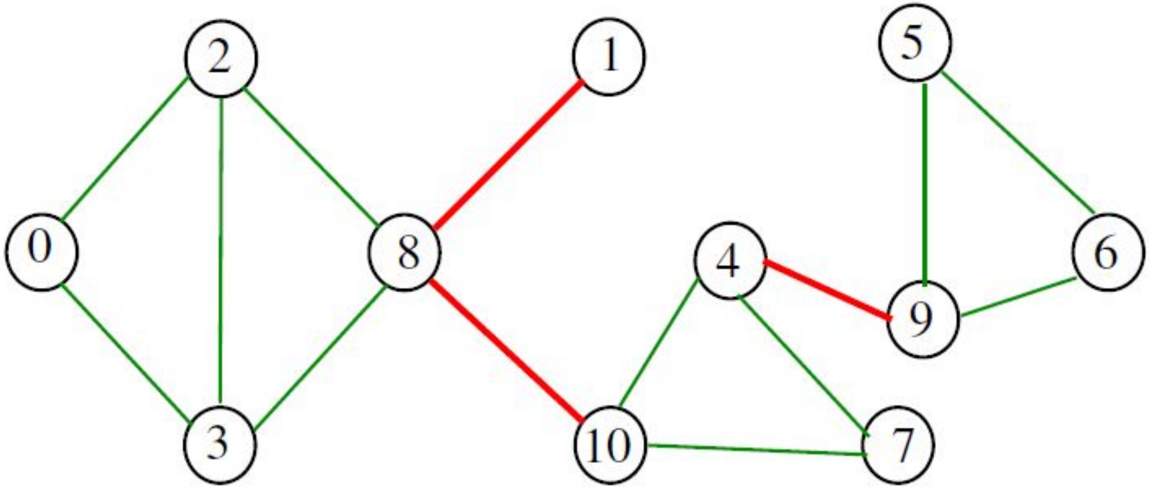
Grafos: pontes e arcos negativos

Laboratório de Programação Competitiva - 2020

Pedro Henrique Paiola

Pontes em grafos

- Seja um grafo não direcionado. Uma ponte é definida como uma aresta que, quando removida, desconecta o grafo (aumenta o número de componentes conexas).
- **Objetivo**: encontrar todas as pontes em um grafo



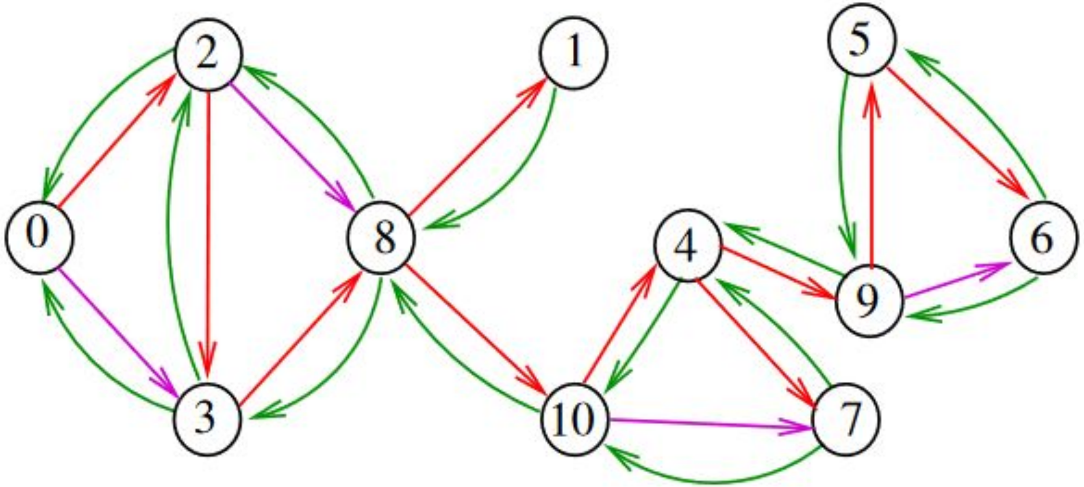
Pontes em grafos

- O algoritmo para isto usa uma DFS, partindo de um vértice arbitrário, em que vale a seguinte observação:
- Supondo que estamos iterando sobre as arestas de um vértice **v**
  - a aresta **(v, to)** é uma ponte see nenhum dos vértices **to** e seus descendentes tiver uma *back-edge* (aresta de retorno) para o vértice **v** ou qualquer um de seus ancestrais.
  - Isso garante que não existe outra maneira de voltar para **v** de **to,** exceto pela própria aresta **(v, to)**

Pontes em grafos

- Para isso, vamos realizar a DFS anotando o tempo de entrada ou número de pré-ordem (**pre**) em cada vértice:

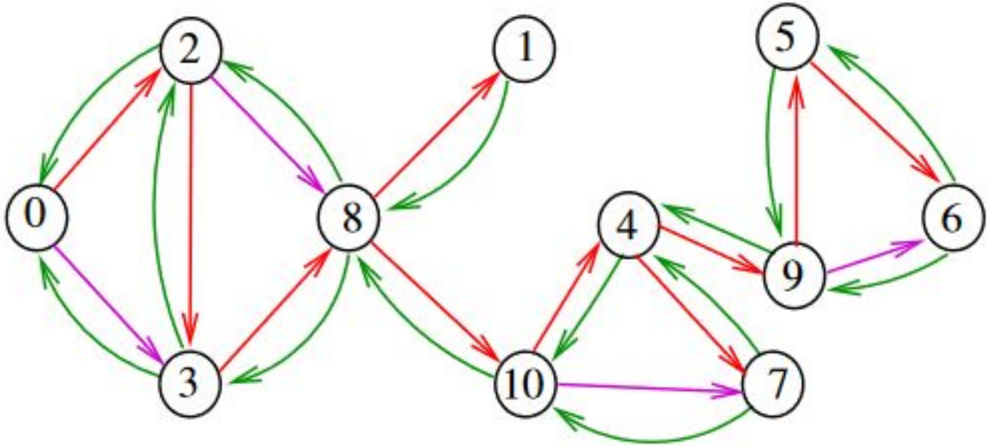
v	0	1	2	3	4	5	6	7	8	9	10
pre[v]	0	**4**	1	2	6	8	9	10	3	**7**	**5**



Pontes em grafos

- Além disso, vamos manter um vetor **low,** armazenando o menor tempo de entrada alcançado utilizando arcos da arborescência e ATÉ UM arco de retorno.

v	0	1	2	3	4	5	6	7	8	9	10
pre[v]	0	4	1	2	6	8	9	10	3	7	5
low[v]	0	4	0	0	5	7	7	5	1	7	5



Pontes em grafos

- Para toda aresta **(v, w)**
  - Se **(v, w)** é uma aresta de arborescência, então **low[v] <= low[w]**
  - Se **(v, w)** é uma aresta de retorno, então **low[v] <= pre[w]**
  - E **(v, w)** é uma ponte see **low[w] > pre[v]**

$$low[v] = min \begin{cases} pre[v] & \\ pre[p] & \text{para todo p em que (v, p) é uma aresta de retorno} \\ low[w] & \text{para todo w em que (v, w) é uma aresta de arborescência} \end{cases}$$

Pontes em grafos

```
dfs(v, pai)
    visitado = true
    pre[v] = low[v] = ordemVisitacao++
    para cada aresta(v, w) com w != pai
        se w já foi visitado
            low[v] = min(low[v], pre[w])
        senão
            dfs(w, v)
            low[v] = min(low[v], low[w])
            se (low[w] > pre[v])
                eh_ponte(v, w)
```

Implementação em C++: https://cp-algorithms-brasil.com/grafos/pontes.html

Caminho mínimo com arcos negativos

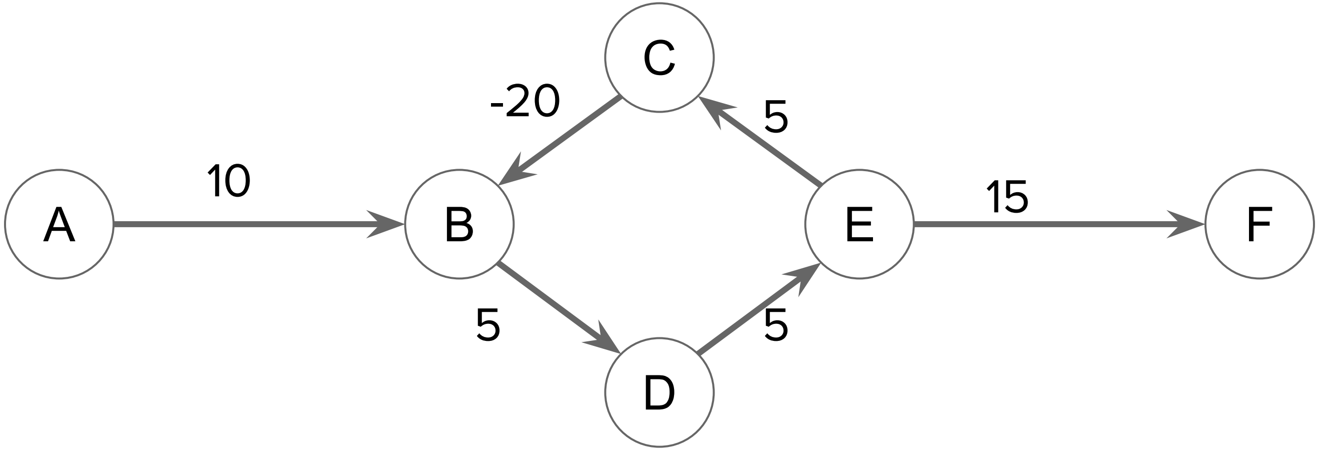
- Anteriormente, já apresentamos o problema do caminho mínimo em um grafo.
- Trabalhamos com o clássico algoritmo de Dijkstra, porém este método NÃO aceita que o grafo tenha arcos negativos
- Para isso temos como opção o algoritmo de Bellman-Ford
    - Por que não o usamos sempre? Ele possui maior complexidade

Caminho mínimo com arcos negativos

- Quando pode aparecer arcos negativos em problemas de caminho mínimo?
  - Parece não fazer muito sentido falar em “distância” com arcos negativos, mas podemos ter diversos tipos de outros problemas em que caímos neste caso.
  - Por exemplo: problemas envolvendo dinheiro, onde arcos positivos representam gastos e arcos negativos representam lucro. Nesse caso, um caminho mínimo maximiza o lucro.

Caminho mínimo com arcos negativos

- Caso insolúvel: presença de ciclos negativos
- Nesta situação o problema se torna NP-difícil



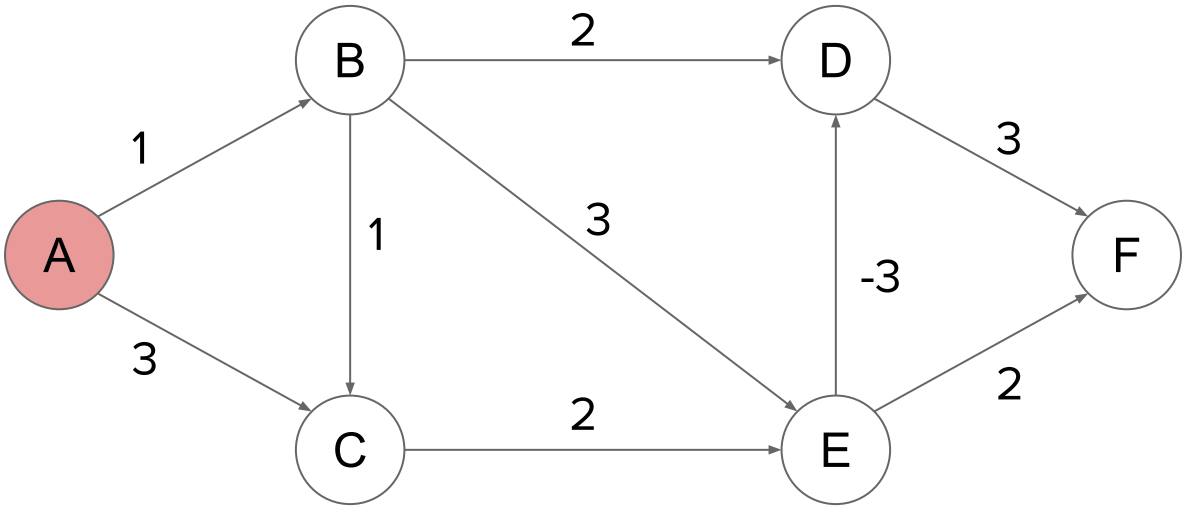
Bellman-Ford

- O algoritmo Bellman-Ford é dividido em três etapas:
  - Inicialização: padronização das distâncias
  - Relaxamento: cálculo efetivo dos caminhos mínimos
  - Verificação de ciclos negativos

Bellman-Ford

- Inicialização: como no Dijkstra, a distância até a origem é inicializada com 0 e as outras como infinito.

Bellman-Ford



Vértices	A	B	C	D	E	F
Estimativas	0	∞	∞	∞	∞	∞
Precedentes	-	-	-	-	-	-

Bellman-Ford

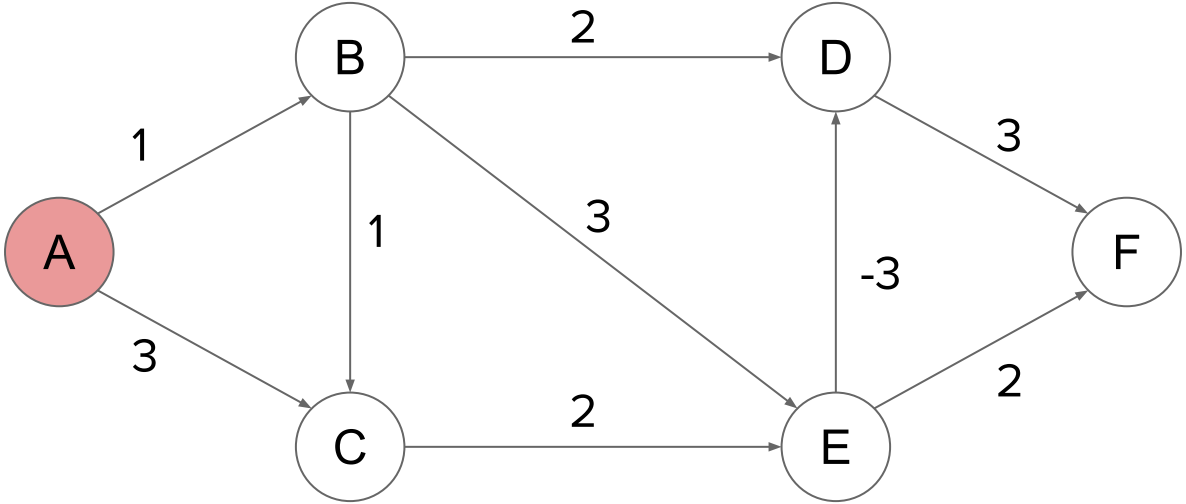
- Relaxamento: a técnica do relaxamento consiste em verificar se pode ser encontrado um caminho mais curto para v passando por um certo vértice u

```
se d[u] + peso(u,v) < d[v] então
    d[v] = d[u] + peso(u,v)
    p[v] = u
```

- De forma semelhante ao Dijkstra, isso será feito N (número de vértices) - 1 vezes, porém considerando TODAS as arestas, e não apenas as incidentes no último vértice "fechado".

Bellman-Ford

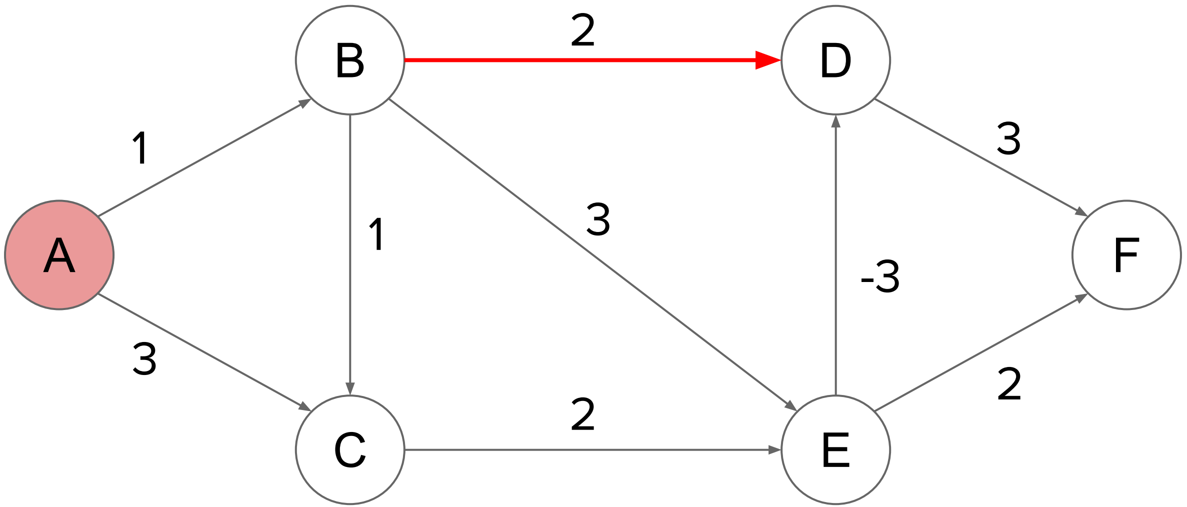
Ordem de relaxamento:
(B,D)
(A,B)
(D,F)
(E,D)
(E,F)
(C,E)
(B,E)
(B,C)
(A,C)



Vértices	**A**	**B**	**C**	**D**	**E**	**F**
Estimativas	0	∞	∞	∞	∞	∞
Precedentes	-	-	-	-	-	-

Bellman-Ford

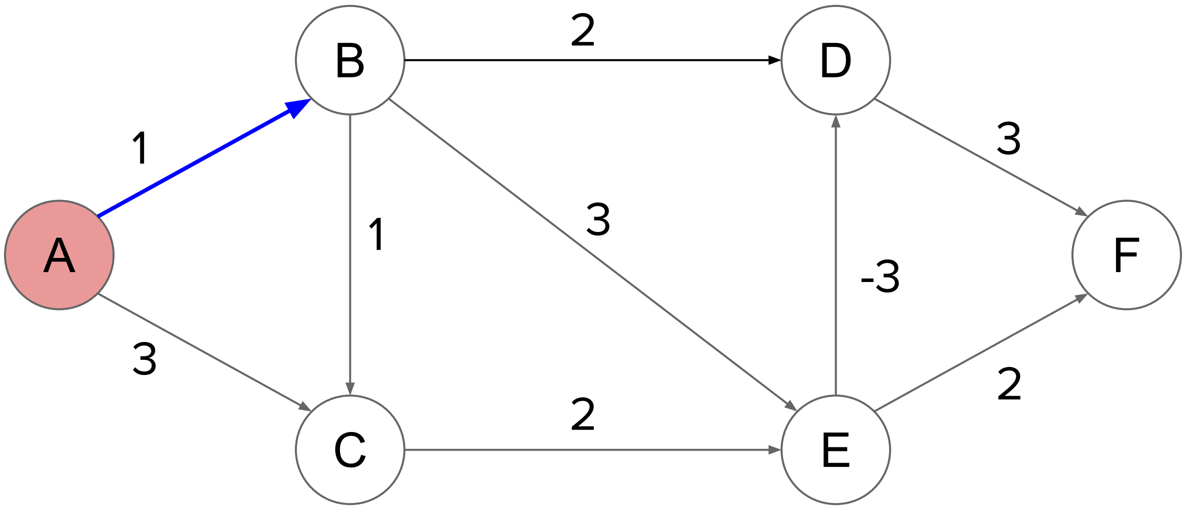
Ordem de relaxamento:
(B,D)
(A,B)
(D,F)
(E,D)
(E,F)
(C,E)
(B,E)
(B,C)
(A,C)



Vértices	**A**	**B**	**C**	**D**	**E**	**F**
Estimativas	0	∞	∞	∞	∞	∞
Precedentes	-	-	-	-	-	-

Bellman-Ford

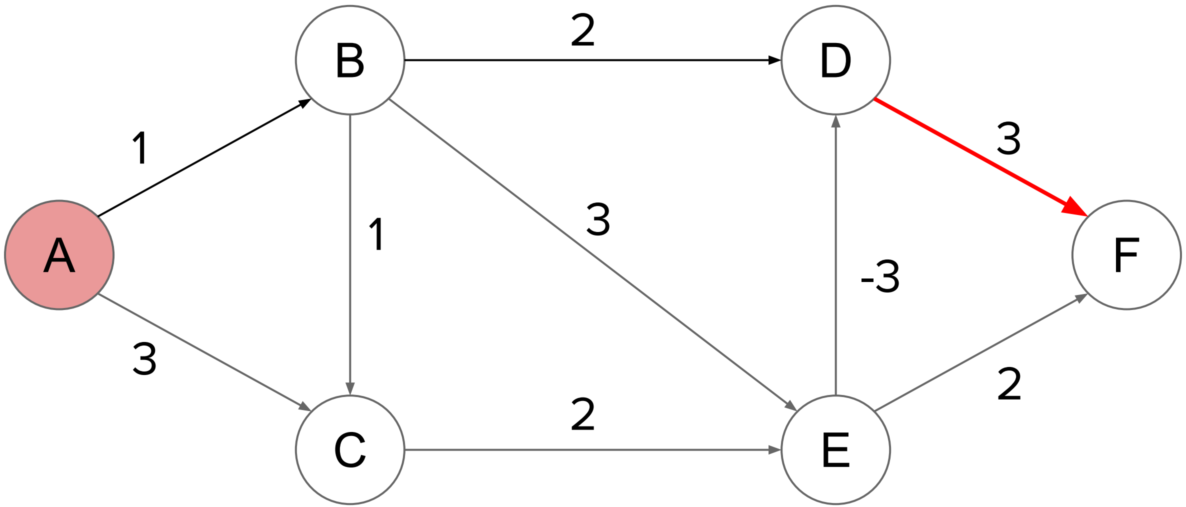
Ordem de relaxamento:
(B,D)
(A,B)
(D,F)
(E,D)
(E,F)
(C,E)
(B,E)
(B,C)
(A,C)



Vértices	**A**	**B**	**C**	**D**	**E**	**F**
Estimativas	0	1	∞	∞	∞	∞
Precedentes	-	A	-	-	-	-

Bellman-Ford

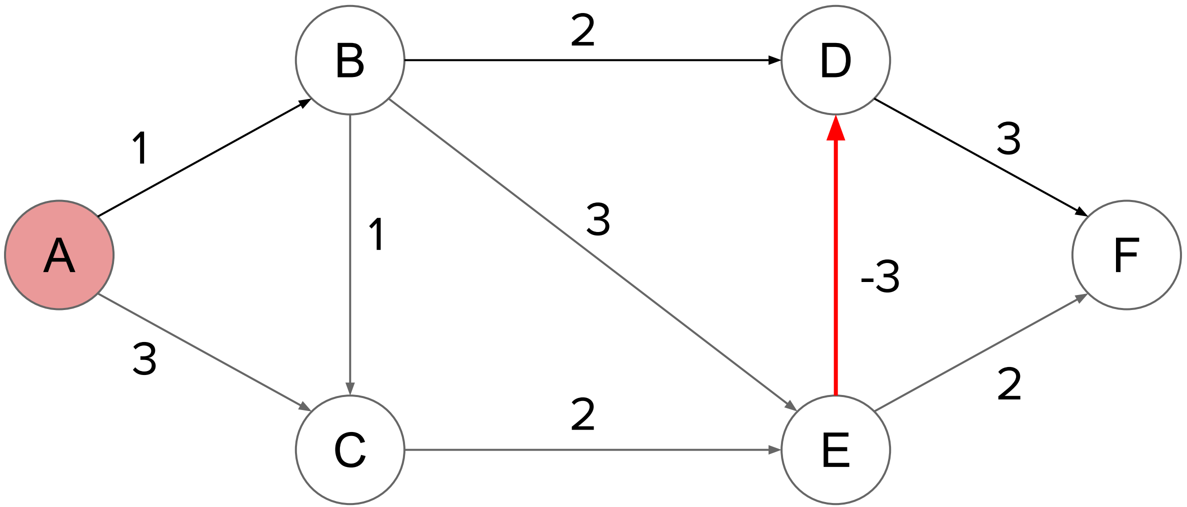
Ordem de relaxamento:
(B,D)
(A,B)
(D,F)
(E,D)
(E,F)
(C,E)
(B,E)
(B,C)
(A,C)



Vértices	**A**	**B**	**C**	**D**	**E**	**F**
Estimativas	0	1	∞	∞	∞	∞
Precedentes	-	A	-	-	-	-

Bellman-Ford

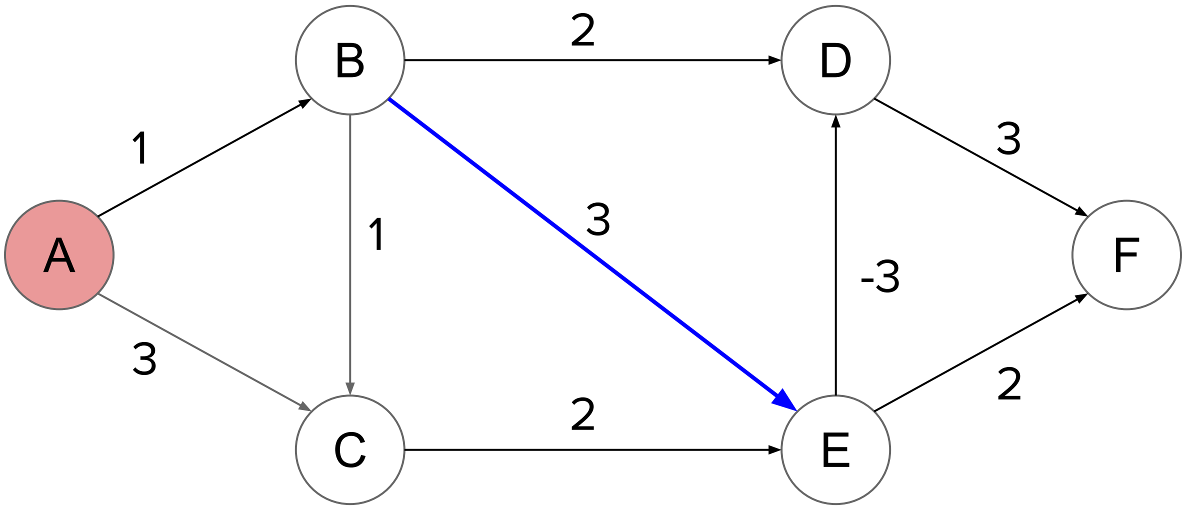
Ordem de relaxamento:
(B,D)
(A,B)
(D,F)
(E,D)
(E,F)
(C,E)
(B,E)
(B,C)
(A,C)



Vértices	**A**	**B**	**C**	**D**	**E**	**F**
Estimativas	0	1	∞	∞	∞	∞
Precedentes	-	A	-	-	-	-

Bellman-Ford

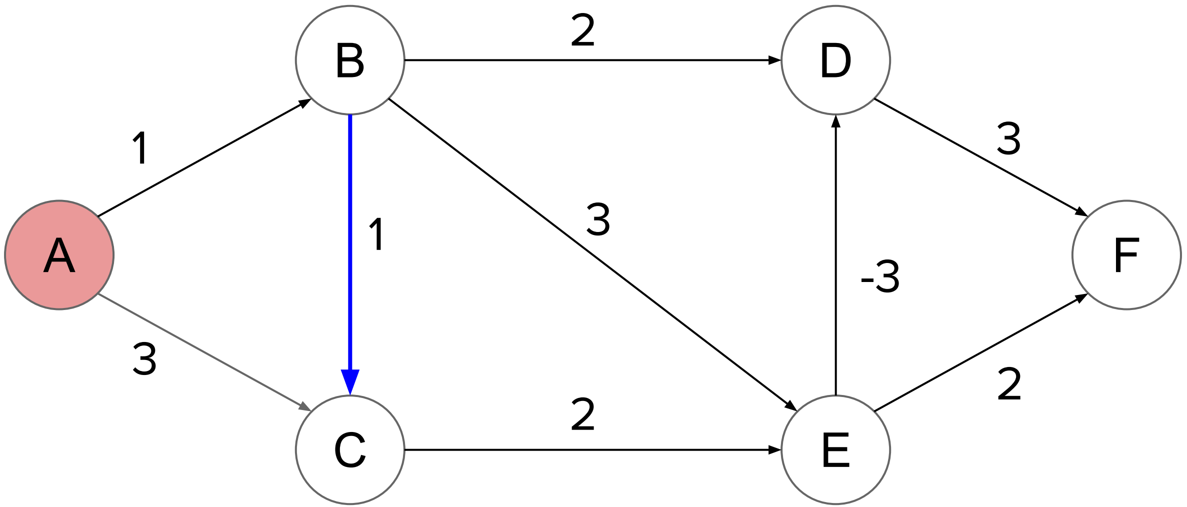
Ordem de relaxamento:
(B,D)
(A,B)
(D,F)
(E,D)
(E,F)
(C,E)
(B,E)
(B,C)
(A,C)



Vértices	A	B	C	D	E	F
Estimativas	0	1	∞	∞	4	∞
Precedentes	-	A	-	-	B	-

Bellman-Ford

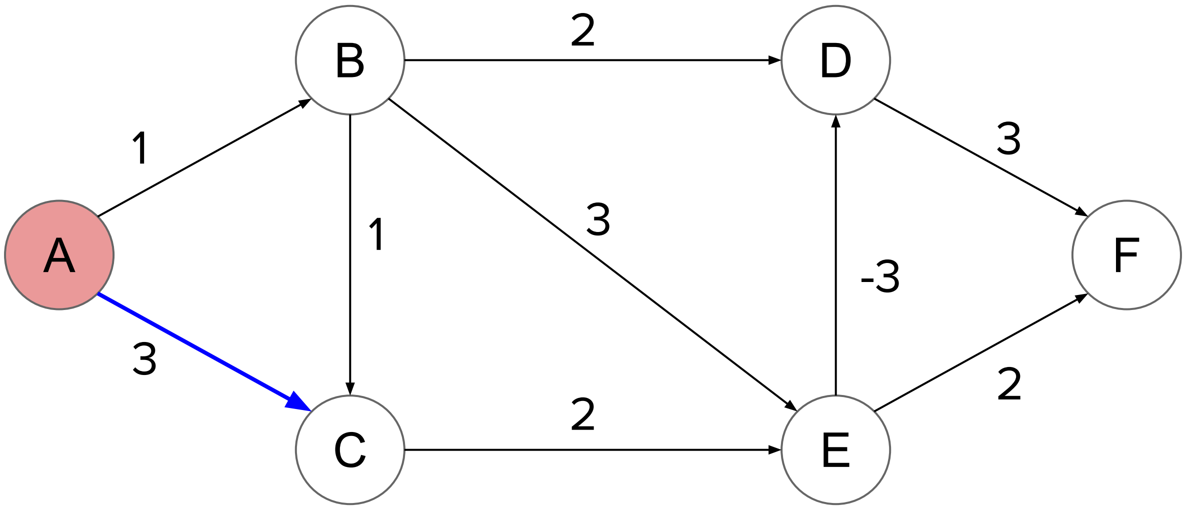
Ordem de relaxamento:
(B,D)
(A,B)
(D,F)
(E,D)
(E,F)
(C,E)
(B,E)
(B,C)
(A,C)



Vértices	**A**	**B**	**C**	**D**	**E**	**F**
Estimativas	0	1	2	∞	4	∞
Precedentes	-	A	B	-	B	-

Bellman-Ford

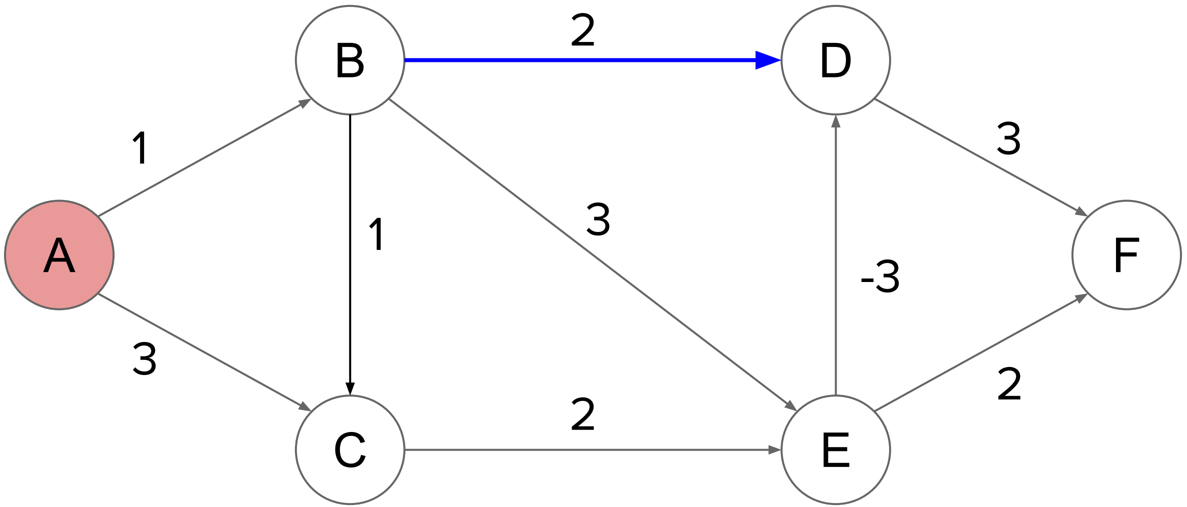
Ordem de relaxamento:
(B,D)
(A,B)
(D,F)
(E,D)
(E,F)
(C,E)
(B,E)
(B,C)
(A,C)



Vértices	**A**	**B**	**C**	**D**	**E**	**F**
Estimativas	0	1	2	∞	4	∞
Precedentes	-	A	B	-	B	-

Bellman-Ford

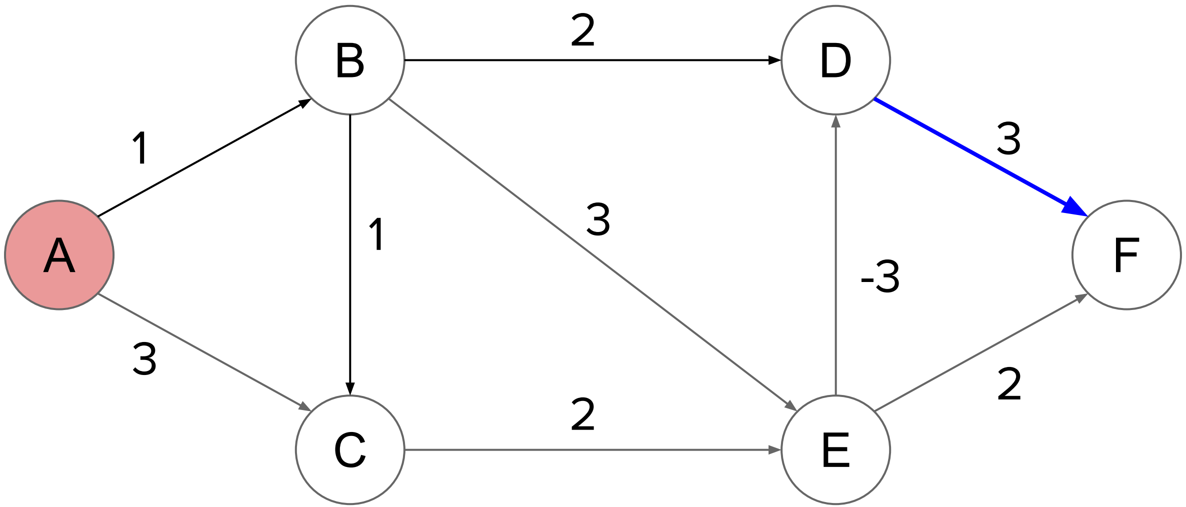
Ordem de relaxamento:
(B,D)
(A,B)
(D,F)
(E,D)
(E,F)
(C,E)
(B,E)
(B,C)
(A,C)



Vértices	**A**	**B**	**C**	**D**	**E**	**F**
Estimativas	0	1	2	3	4	∞
Precedentes	-	A	B	B	B	-

Bellman-Ford

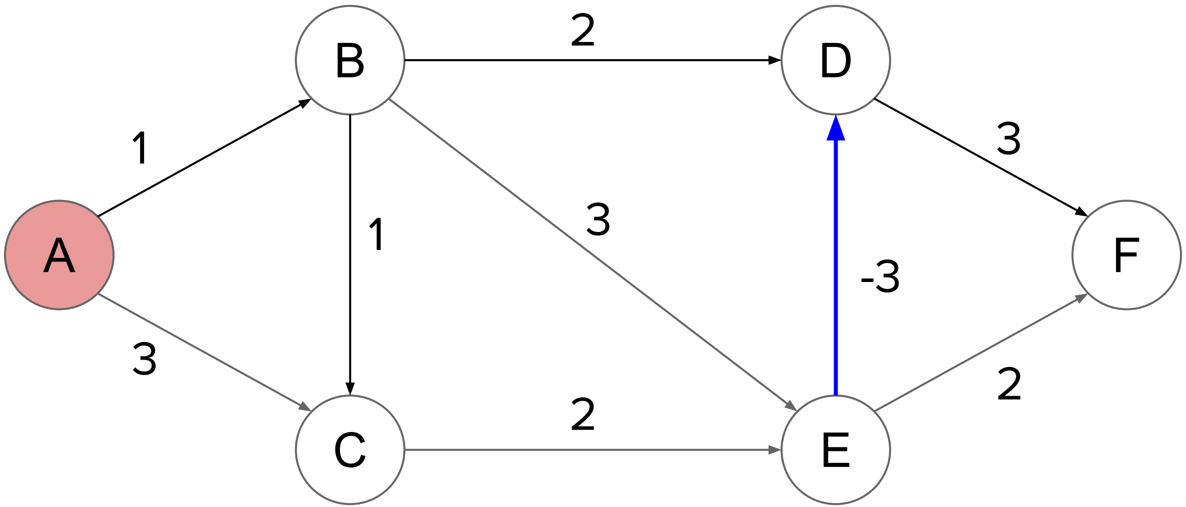
Ordem de relaxamento:
(B,D)
(A,B)
(D,F)
(E,D)
(E,F)
(C,E)
(B,E)
(B,C)
(A,C)



Vértices	**A**	**B**	**C**	**D**	**E**	**F**
Estimativas	0	1	2	3	4	6
Precedentes	-	A	B	B	B	D

Bellman-Ford

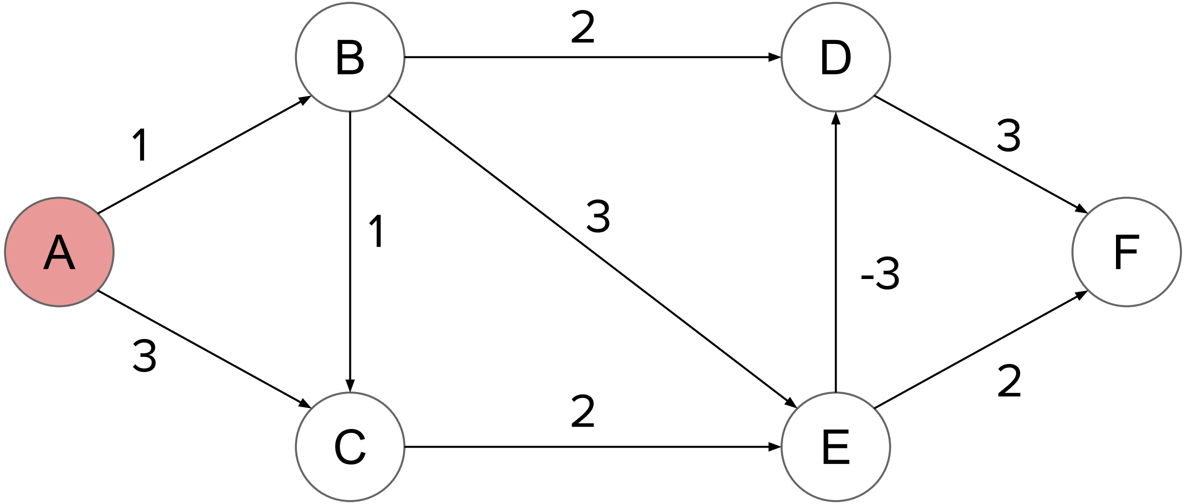
Ordem de relaxamento:
(B,D)
(A,B)
(D,F)
(E,D)
(E,F)
(C,E)
(B,E)
(B,C)
(A,C)



Vértices	**A**	**B**	**C**	**D**	**E**	**F**
Estimativas	0	1	2	1	4	6
Precedentes	-	A	B	E	B	D

Bellman-Ford

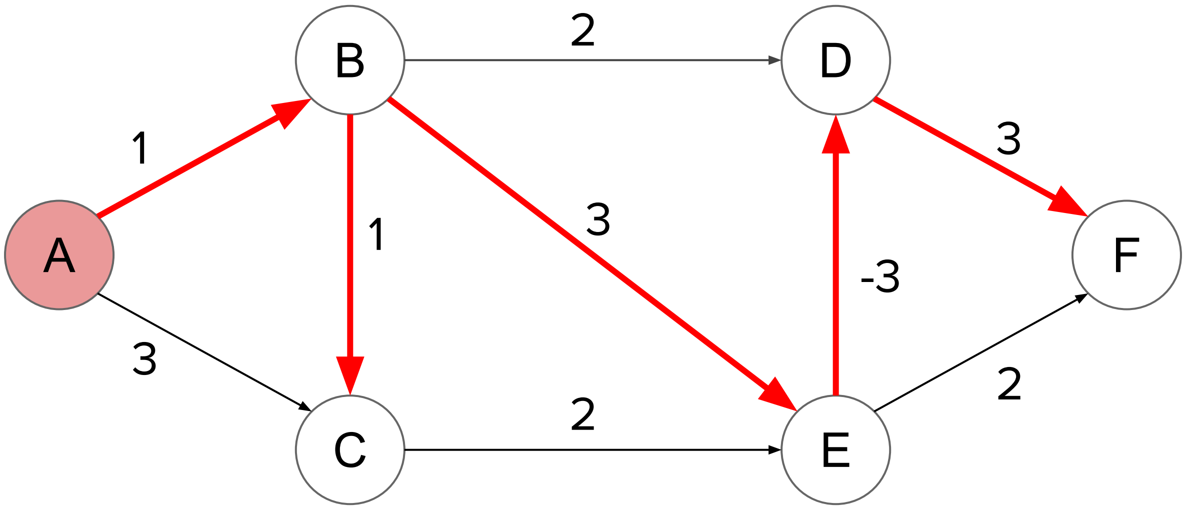
Ordem de relaxamento:
(B,D)
(A,B)
(D,F)
(E,D)
(E,F)
(C,E)
(B,E)
(B,C)
(A,C)



Vértices	**A**	**B**	**C**	**D**	**E**	**F**
Estimativas	0	1	2	1	4	6
Precedentes	-	A	B	E	B	D

No final

Bellman-Ford



Vértices	A	B	C	D	E	F
Estimativas	0	1	2	1	4	4
Precedentes	-	A	B	E	B	D

Bellman-Ford

- Checagem de ciclos negativos: o relaxamento é aplicado mais uma vez, se houver alguma situação em que se encontre um caminho melhor, é por que temos a presença de um ciclo negativo (caso em que SEMPRE pode-se encontrar um ciclo menor, ao “andar” mais uma vez pelo ciclo)

Bellman-Ford

```
BellmanFord(G, origem)
    d[v] = infinito, para todo v
    p[v] = -1, para todo v
    d[origem] = 0
    para i de 1 até |V(G)| - 1 faça
        para cada aresta (u,v) de G faça
            relax(u, v, w)
    para cada aresta (u,v) de G faça
        se d[v] > d[u] + peso(u,v)
            retorna FALSE
    retorna TRUE
```

Referências

https://www.ime.usp.br/~am/328-12/aulas/aula-0327h.pdf
https://cp-algorithms-brasil.com/grafos/pontes.html
https://www.geeksforgeeks.org/bridge-in-a-graph/
https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/cheapestpaths.html#sec:min-path
https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/bellman-ford.html
https://www.ic.unicamp.br/~rezende/ensino/mo417/2010s2/Slides/Aula23.pdf
http://www.dt.fee.unicamp.br/~ricfow/IA881/caminhoMinimo.pdf
https://pt.slideshare.net/jackocap/anlise-de-algoritmos-problemas-em-grafos-caminho-mnimo-algoritmo-de-bellmanford