Curso Estruturas de Dados e Algoritmos Expert

Prof. Nelio Alves

Grafos (parte 4)

DEVSUPERIOR

https://devsuperior.com.br

1

Algoritmos de Menor Caminho

https://devsuperior.com.br

Prof. Dr. Nelio Alves

2

O problema do menor caminho

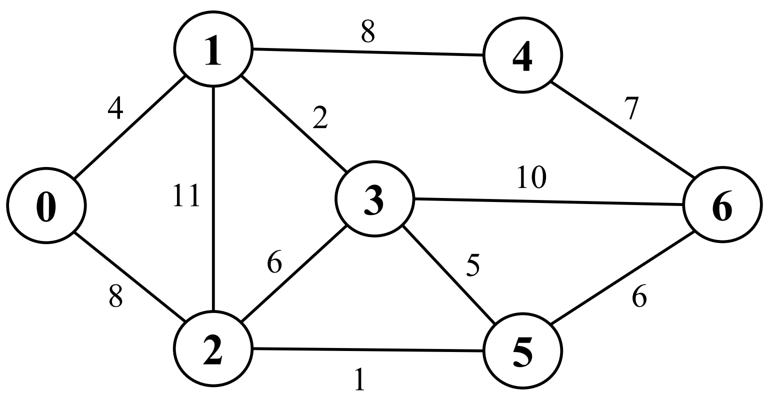
- Um motorista deseja encontrar o caminho mais curto possível entre duas cidades.

- Ele consulta um aplicativo de mapas: a partir de um mapa de sua região, que representa rodovias e distâncias entre cada par de cidades, como determinar uma rota mais curta entre as cidades?

- Podemos modelar esse problema com o auxílio de grafos
  - Representamos cidades como vértices
  - Representamos rodovias como arestas
  - Dois vértices estão ligados se duas cidades estão unidas por rodovia
  - O peso das arestas indica o custo associado

O problema do menor caminho

- Qual o menor caminho entre a cidade 0 e a cidade 6?

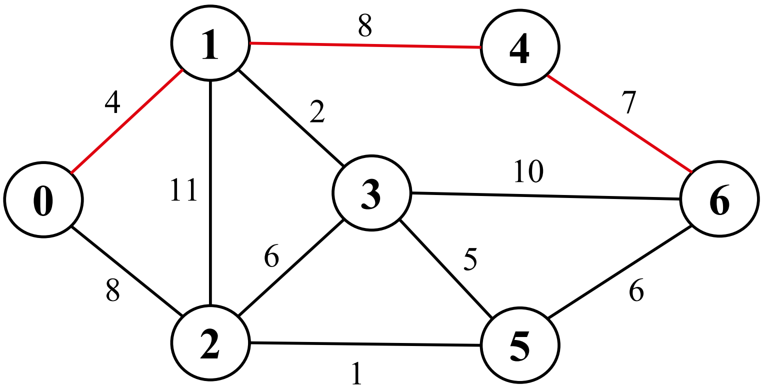


O problema do menor caminho

- Qual o menor caminho entre a cidade 0 e a cidade 6?

Possibilidade 1

dist(0, 6) = 4 + 8 + 7
dist(0, 6) = 19



O problema do menor caminho

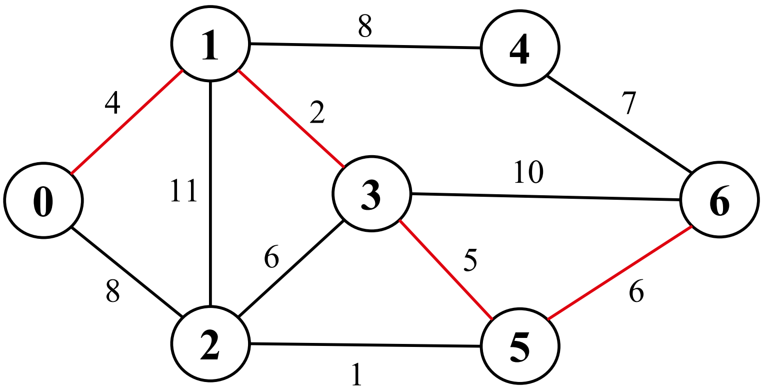
- Qual o menor caminho entre a cidade 0 e a cidade 6?

Possibilidade 1

dist(0, 6) = 4 + 8 + 7
dist(0, 6) = 19

Possibilidade 2

dist(0, 6) = 4 + 2 + 5 + 6
dist(0, 6) = 17



O problema do menor caminho

- Qual o menor caminho entre a cidade 0 e a cidade 6?

Possibilidade 1

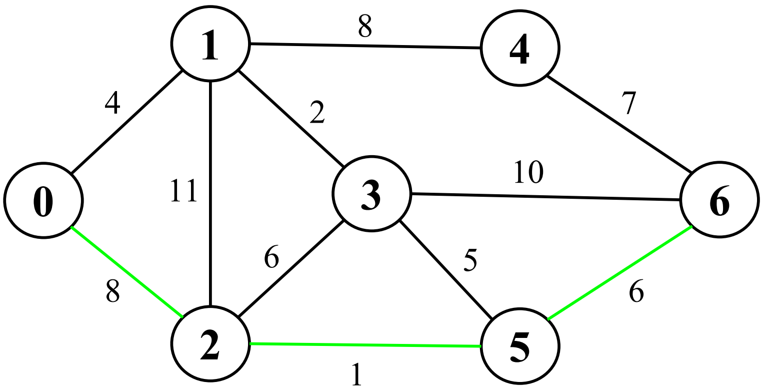
dist(0, 6) = 4 + 8 + 7
dist(0, 6) = 19

Possibilidade 2

dist(0, 6) = 4 + 2 + 5 + 6
dist(0, 6) = 17

Possibilidade 3

dist(0, 6) = 8 + 1 + 6
dist(0, 6) = 15



Menor caminho neste grafo!

O problema do menor caminho

- Soluções?
  - Enumerar todas as rotas e selecionar a mais curta?
- Problemas
  - **Complexidade Pior Caso**
    - Se considerarmos um grafo completo não direcionado, o número de rotas pode ser na ordem de $O(n^n)$, **superexponencial**!
    - Para n = 15, precisaríamos de cerca de 138 anos para concluir uma execução.
  - **Grafo Direcionado com Ciclos**
    - O número de caminhos é potencialmente infinito sem restrições na geração do caminho.

Surgem então algoritmos para resolver o problema com tempo melhor!

Variações do problema do menor caminho

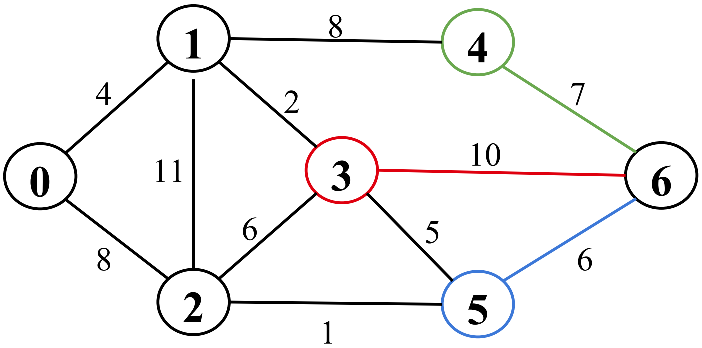
O problema do caminho pode aparecer de diferentes maneiras:

- Caminhos mais curtos de **origem única**
  - Algoritmo de Dijkstra
  - Algoritmo de Bellman-Ford
- Caminhos mais curtos de **todos os pares**
  - Algoritmo de Floyd-Warshall
- Caminhos mais curtos de **destino único**
  - Equivalente a origem única no grafo invertido
- Caminho mais curto de **par único**
  - Também usa métodos de origem única para resolver

Conceitos fundamentais

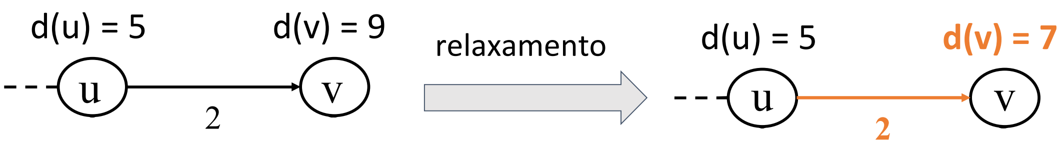
- Partindo de um vértice inicial s, podemos associar cada vértice a um valor d(v), que nos diz o valor do menor caminho de s até v.
- d(v) mantém a estimativa do custo do menor caminho

Exemplo: **d(6) = min(d(4)+7, d(3)+10, d(5)+6)**



Conceitos fundamentais

- Inicialmente faremos um estimativa pessimista para o valor do caminho mínimo até cada vértice: d(v) = ∞
- **Relaxamento de arestas**
  - Processo no qual verificamos se podemos melhorar nossa estimativa pessimista, fazendo um caminho que passa pela aresta (u, v)
  - Conceito <u>central</u> em algoritmos de caminho mínimo



Sub-rotina de relaxamento de arestas

Pseudocódigo

```
relax(u, v, w)
  if d[v] > d[u] + w(u, v)
    d[v] = d[u] + w(u, v)
    antecessor[v] = u
```

onde

- (u, v) é a aresta
- w é seu peso
- d é a estimativa de distância

Algoritmo de Dijkstra

https://devsuperior.com.br

Prof. Dr. Nelio Alves

Algoritmo de Dijkstra

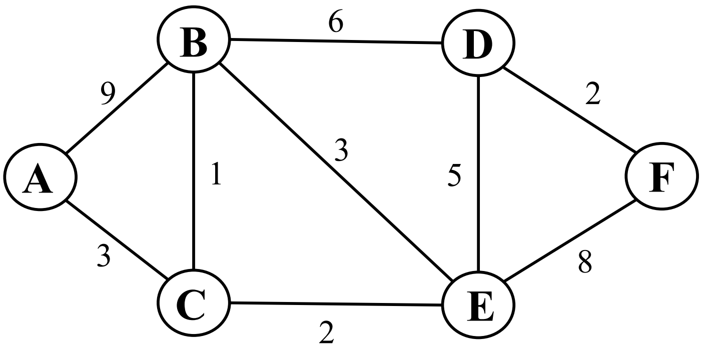
Características

- Caminhos mais curtos de origem única
- Admite ciclos
- Só admite pesos positivos

Estratégia

1. Começa a partir de um nó inicial e faz estimativas pessimistas (infinito)
2. Seleciona o vértice u mais próximo não visitado para processar
3. Relaxa as arestas adjacentes a u
4. Marca u como visitado
5. Volta ao passo 2 até que visitemos todo o grafo

Algoritmo de Dijkstra

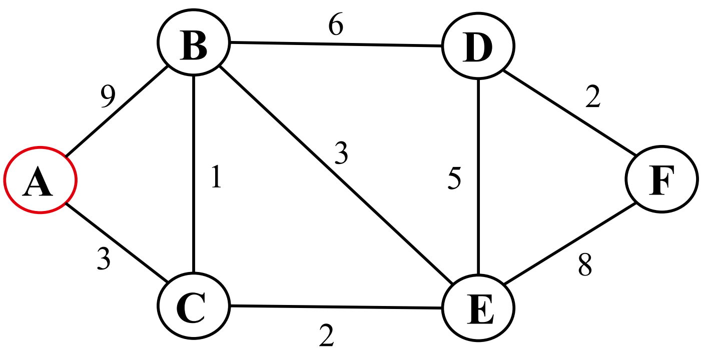


Vértice	Menor distância	Vértice Anterior
A		
B		
C		
D		
E		
F		

visitados = []

Algoritmo de Dijkstra

Começar em A e fazer estimativas pessimistas

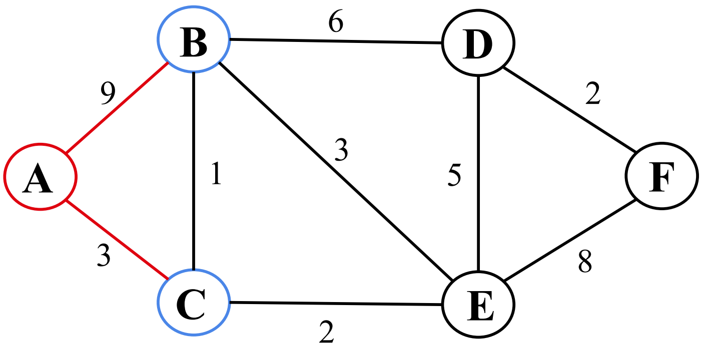


Vértice	Menor distância	Vértice Anterior
A	0	A
B	∞	
C	∞	
D	∞	
E	∞	
F	∞	

visitados = []

Algoritmo de Dijkstra

Relaxa vértices adjacentes ao A

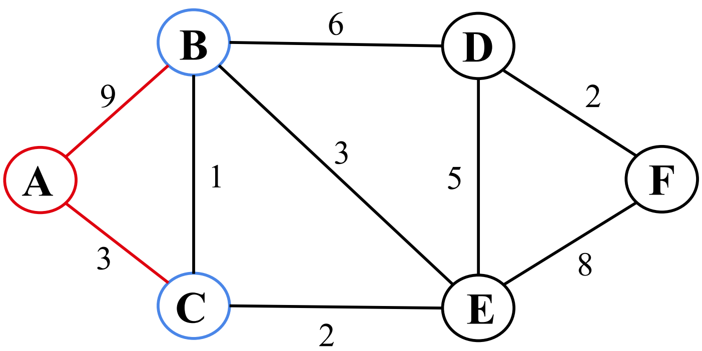


Vértice	Menor distância	Vértice Anterior
A	0	A
B	∞	
C	∞	
D	∞	
E	∞	
F	∞	

visitados = []

Algoritmo de Dijkstra

Relaxa vértices adjacentes ao A

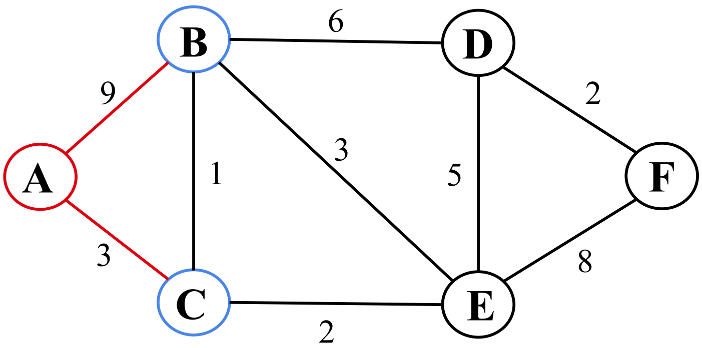


Vértice	Menor distância	Vértice Anterior
A	0	A
B	9	A
C	3	A
D	∞	
E	∞	
F	∞	

visitados = []

Algoritmo de Dijkstra

Relaxa vértices adjacentes ao A

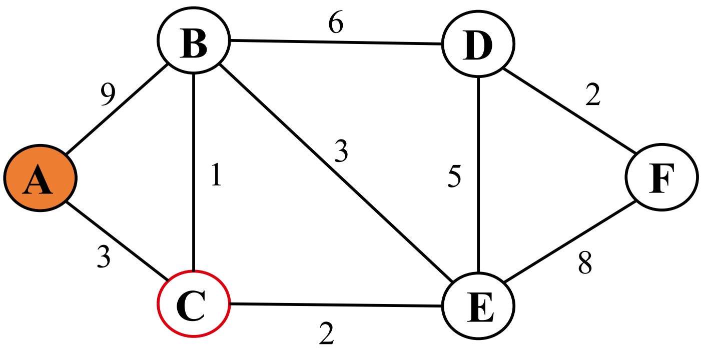


Vértice	Menor distância	Vértice Anterior
A	0	A
B	9	A
C	3	A
D	∞	
E	∞	
F	∞	

visitados = []

Algoritmo de Dijkstra

Marca A como visitado e vai ao vértice mais próximo

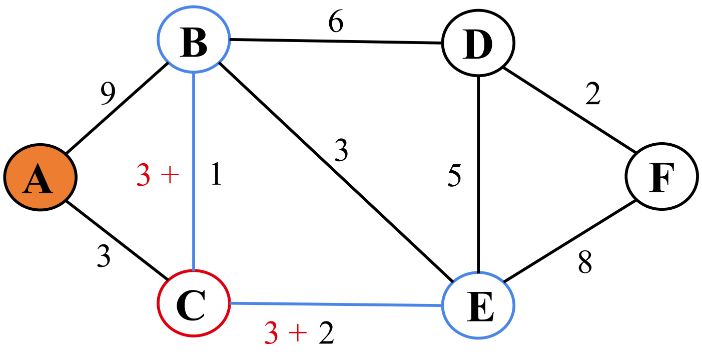


Vértice	Menor distância	Vértice Anterior
A	0	A
B	9	A
C	3	A
D	∞	
E	∞	
F	∞	

visitados = [A]

Algoritmo de Dijkstra

Relaxa vértices adjacentes a C

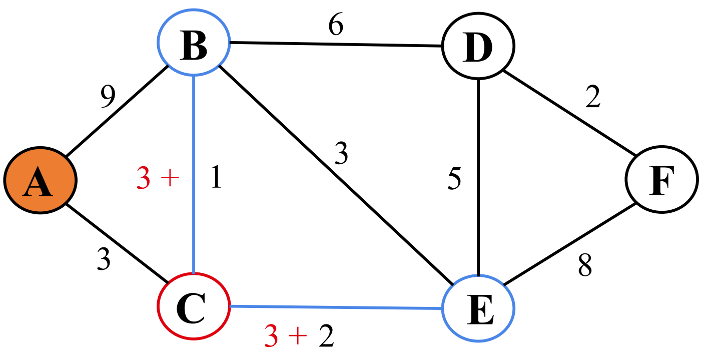


Vértice	Menor distância	Vértice Anterior
A	0	A
B	9	A
C	3	A
D	∞	
E	∞	
F	∞	

visitados = [A]

Algoritmo de Dijkstra

Atualiza vértices B e E pois achou distância menor

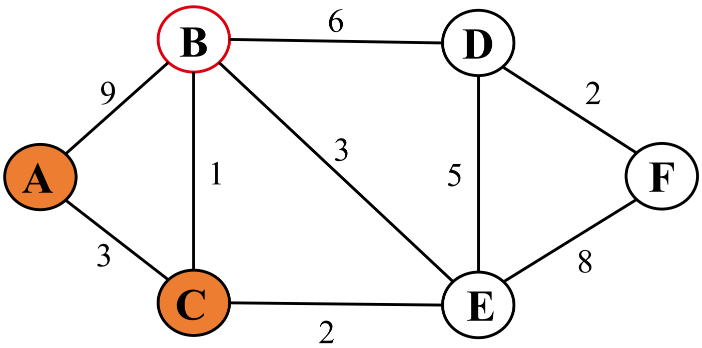


Vértice	Menor distância	Vértice Anterior
A	0	A
B	4	C
C	3	A
D	∞	
E	5	C
F	∞	

visitados = [A]

Algoritmo de Dijkstra

Marca C como visitado e vai ao vértice mais próximo

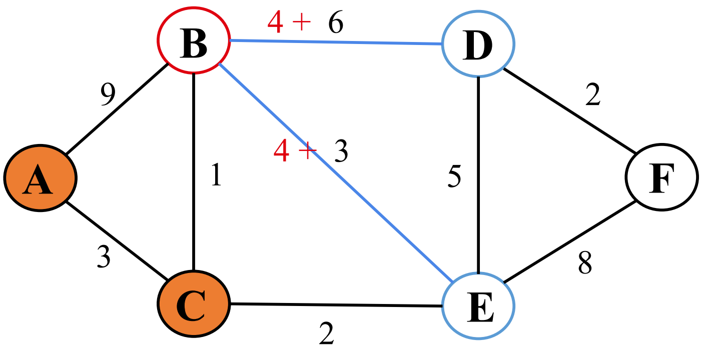


Vértice	Menor distância	Vértice Anterior
A	0	A
B	4	C
C	3	A
D	∞	
E	5	C
F	∞	

visitados = [A, C]

Algoritmo de Dijkstra

Relaxa vértices adjacentes a B

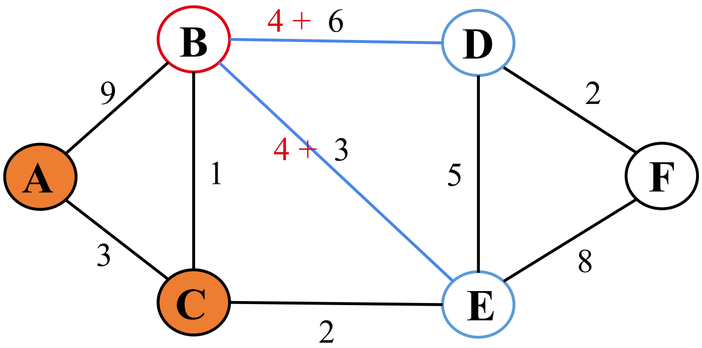


Vértice	Menor distância	Vértice Anterior
A	0	A
B	4	C
C	3	A
D	∞	
E	5	C
F	∞	

visitados = [A, C]

Algoritmo de Dijkstra

Atualiza vértice D pois achou distância menor

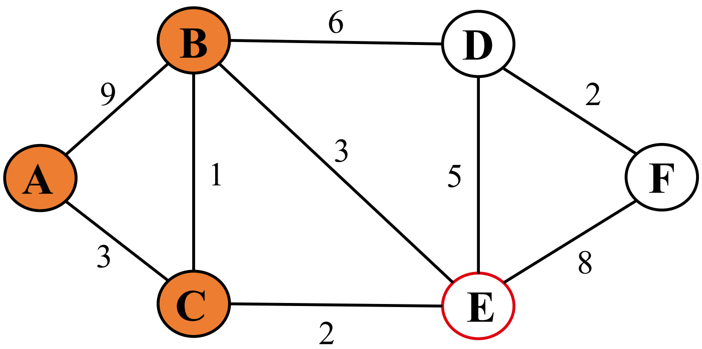


Vértice	Menor distância	Vértice Anterior
A	0	A
B	4	C
C	3	A
D	10	B
E	5	C
F	∞	

visitados = [A, C]

Algoritmo de Dijkstra

Marca B como visitado e vai ao vértice mais próximo

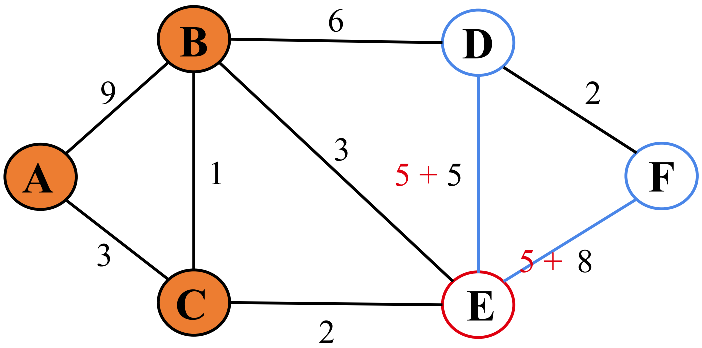


Vértice	Menor distância	Vértice Anterior
A	0	A
B	4	C
C	3	A
D	10	B
E	5	C
F	∞	

visitados = [A, C, B]

Algoritmo de Dijkstra

Relaxa vértices adjacentes a E

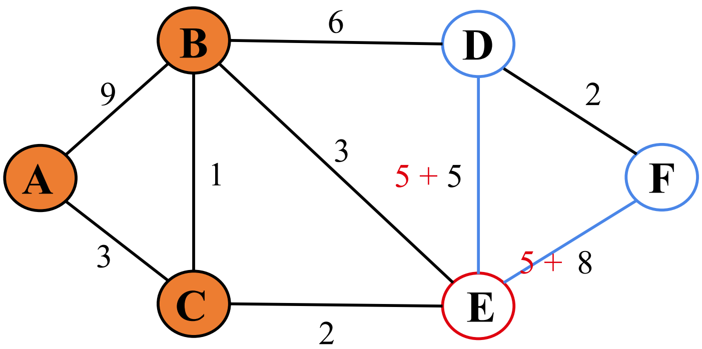


Vértice	Menor distância	Vértice Anterior
A	0	A
B	4	C
C	3	A
D	10	B
E	5	C
F	∞	

visitados = [A, C, B]

Algoritmo de Dijkstra

Atualiza vértice F pois achou distância menor

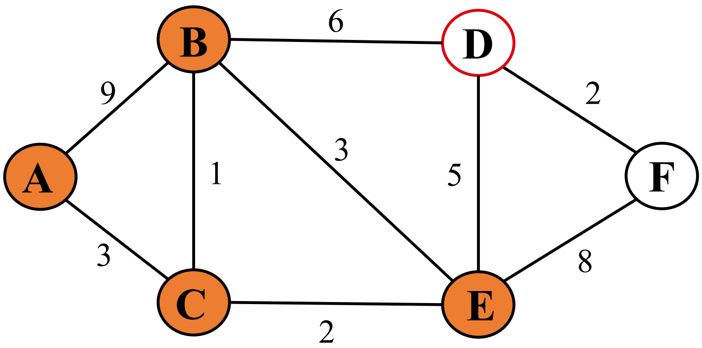


Vértice	Menor distância	Vértice Anterior
A	0	A
B	4	C
C	3	A
D	10	B
E	5	C
F	13	E

visitados = [A, C, B]

Algoritmo de Dijkstra

Marca E como visitado e vai ao vértice mais próximo

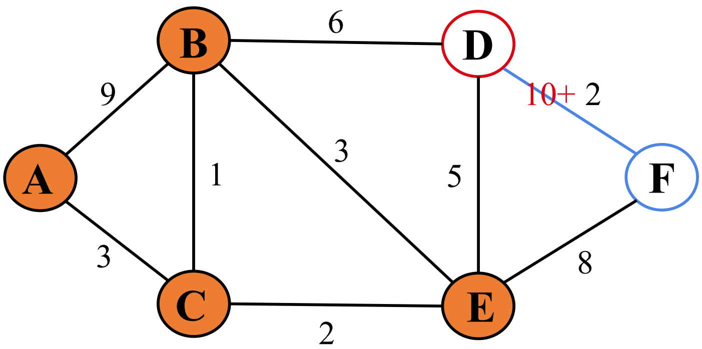


Vértice	Menor distância	Vértice Anterior
A	0	A
B	4	C
C	3	A
D	10	B
E	5	C
F	13	F

visitados = [A, C, B, E]

Algoritmo de Dijkstra

Relaxa vértices adjacentes a F

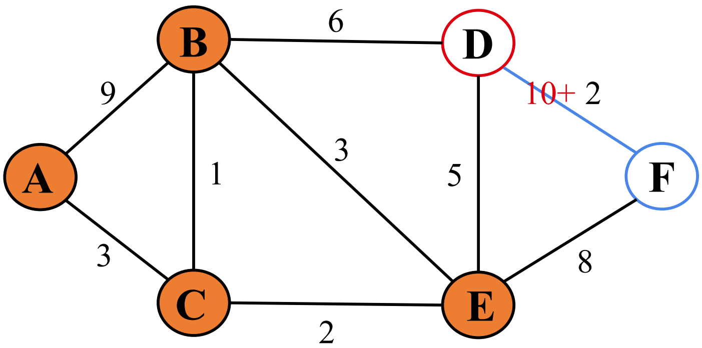


Vértice	Menor distância	Vértice Anterior
A	0	A
B	4	C
C	3	A
D	10	B
E	5	C
F	13	F

visitados = [A, C, B, E]

Algoritmo de Dijkstra

Atualiza vértice F pois achou distância menor

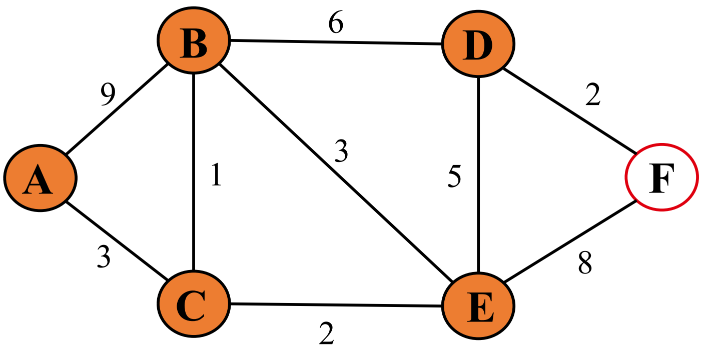


Vértice	Menor distância	Vértice Anterior
A	0	A
B	4	C
C	3	A
D	10	B
E	5	C
F	12	D

visitados = [A, C, B, E]

Algoritmo de Dijkstra

Marca D como visitado a vai ao vértice mais próximo

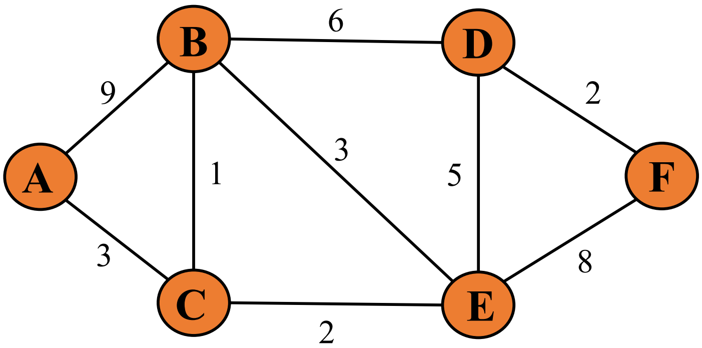


Vértice	Menor distância	Vértice Anterior
A	0	A
B	4	C
C	3	A
D	10	B
E	5	C
F	12	D

visitados = [A, C, B, E, D]

Algoritmo de Dijkstra

Não há vizinhos válidos para relaxar, marca F e finaliza

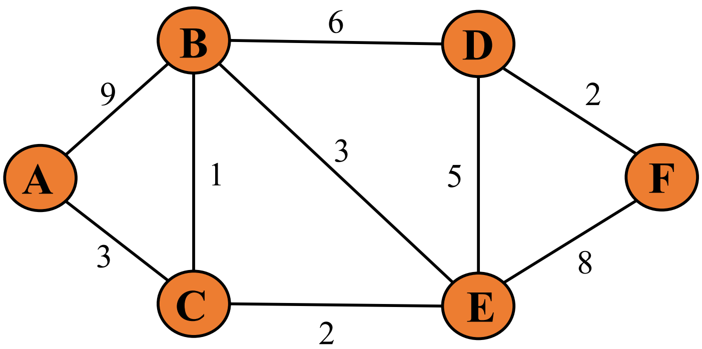


Vértice	Menor distância	Vértice Anterior
A	0	A
B	4	C
C	3	A
D	10	B
E	5	C
F	12	D

visitados = [A, C, B, E, D, F]

Algoritmo de Dijkstra

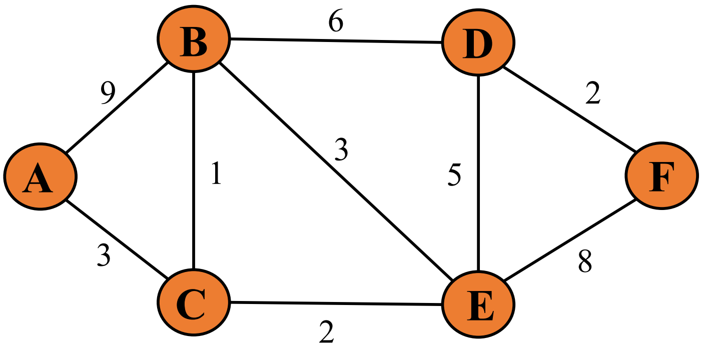
Q: Como recuperar caminho a partir do vértice anterior?



Vértice	Menor distância	Vértice Anterior
A	0	A
B	4	C
C	3	A
D	10	B
E	5	C
F	12	D

Algoritmo de Dijkstra

Q: Como recuperar caminho a partir do vértice anterior?
Percorremos vértices anteriores até achar origem.

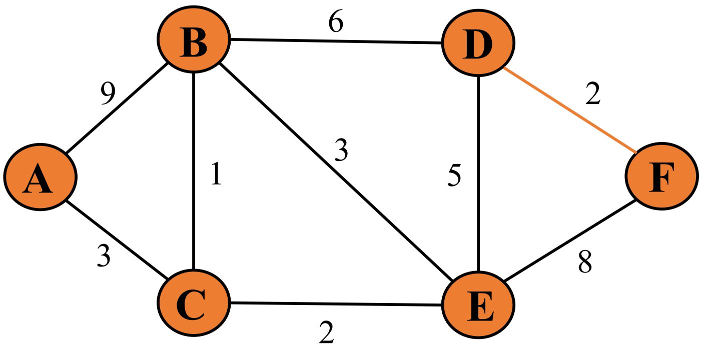


Vértice	Menor distância	Vértice Anterior
A	0	A
B	4	C
C	3	A
D	10	B
E	5	C
F	12	D

Caminho até F = [F]

Algoritmo de Dijkstra

Q: Como recuperar caminho a partir do vértice anterior?
Percorremos vértices anteriores até achar origem.

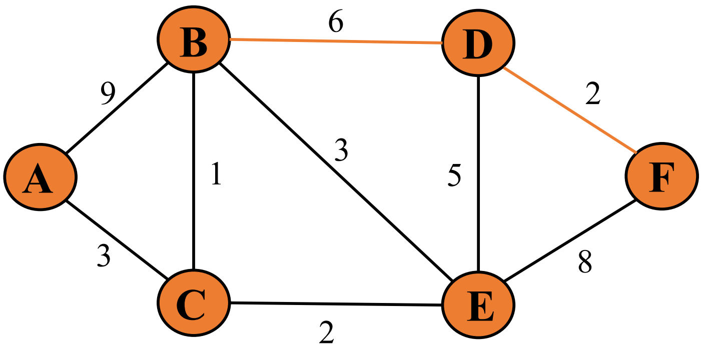


Vértice	Menor distância	Vértice Anterior
A	0	A
B	4	C
C	3	A
D	10	B
E	5	C
F	12	D

Caminho até F = [F, D]

Algoritmo de Dijkstra

Q: Como recuperar caminho a partir do vértice anterior?
Percorremos vértices anteriores até achar origem.

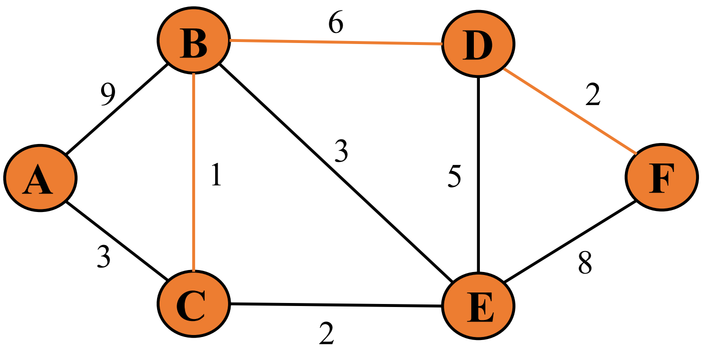


Vértice	Menor distância	Vértice Anterior
A	0	A
B	4	C
C	3	A
D	10	B
E	5	C
F	12	D

Caminho até F = [F, D, B]

Algoritmo de Dijkstra

Q: Como recuperar caminho a partir do vértice anterior?
Percorremos vértices anteriores até achar origem.

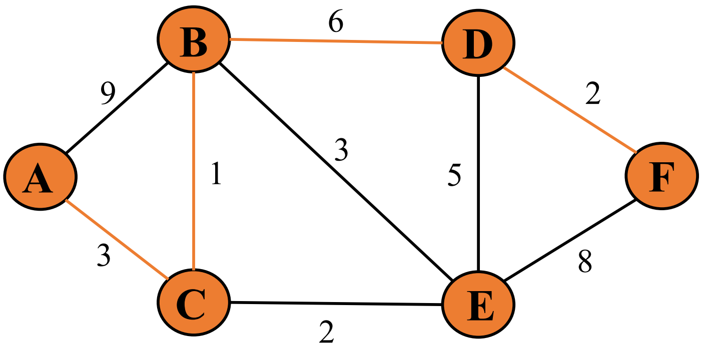


Vértice	Menor distância	Vértice Anterior
A	0	A
B	4	C
C	3	A
D	10	B
E	5	C
F	12	D

Caminho até F = [F, D, B, C]

Algoritmo de Dijkstra

Q: Como recuperar caminho a partir do vértice anterior?
Percorremos vértices anteriores até achar origem.

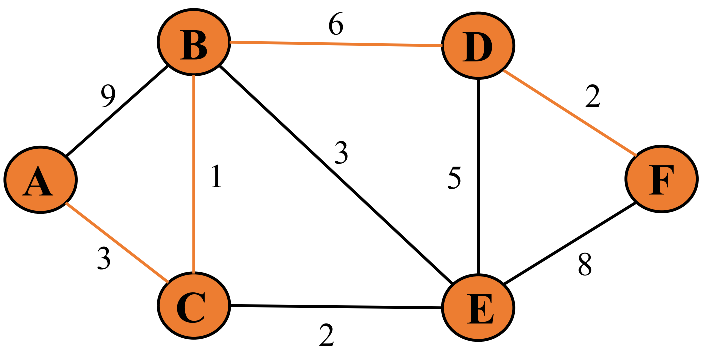


Vértice	Menor distância	Vértice Anterior
A	0	A
B	4	C
C	3	A
D	10	B
E	5	C
F	12	D

Caminho até F = [F, D, B, C, A]

Algoritmo de Dijkstra

Q: Como recuperar caminho a partir do vértice anterior?
Percorremos vértices anteriores até achar origem.



Vértice	Menor distância	Vértice Anterior
A	0	A
B	4	C
C	3	A
D	10	B
E	5	C
F	12	D

Caminho até F = [F, D, B, C, A] = A ⟶ C ⟶ B ⟶ D ⟶ F

Algoritmo de Dijkstra

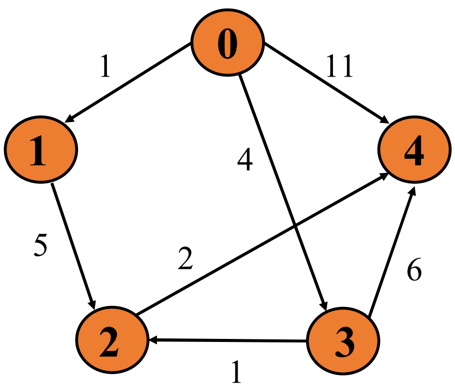
- **Produz solução ótima?**

  Sim, produz sempre o caminho mínimo em um grafo com pesos positivos.

- **Por que funciona?**
  - Pois sempre busca pelo vértice mais leve (abordagem gulosa).
  - Como o algoritmo examina os vértices na ordem de distância da origem, uma vez que um vértice é processado, sua distância final à origem está determinada e não será reconsiderada.

Exercício de fixação

Calcule os caminhos mínimos para o grafo abaixo a partir do vértice 0 aplicando o algoritmo de Dijkstra.



Algoritmo de Dijkstra - Implementação

- Possui algumas formas de implementar, cada um com seu melhor caso de uso

- Implementação para grafos densos
    - Procura próxima visita em array ordenado
    - Complexidade: $O(V^2 + EV)$

- Implementação para grafos esparsos
    - Procura próxima visita em fila de prioridade (heap binário)
    - Complexidade: $O((V+E) \log V)$
    - Mais usada em geral

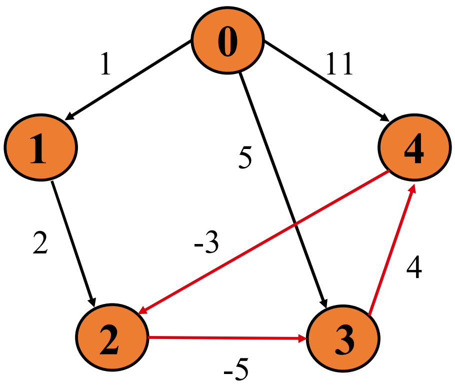
Algoritmo de Bellman-Ford

https://devsuperior.com.br

Prof. Dr. Nelio Alves

Motivação

- Como lidar com ciclos negativos em um grafo?



Caminho mínimo direto:
- $0 \longrightarrow 1 \longrightarrow 2$
- Peso: 1 + 2 = 3

Caminho mínimo com ciclo negativo:
- $0 \longrightarrow 1 \longrightarrow 2 \longrightarrow 3 \longrightarrow 4 \longrightarrow 2$
- Peso: 1 + 2 - 5 + 4 - 3 = -1

Conseguimos melhorar a resposta infinitamente ao incluir ciclos negativos, logo não existe caminho válido.

Possível solução: Algoritmo de Bellman-Ford!

Algoritmo de Bellman-Ford

Características
- Caminhos mais curtos de origem única
- Arestas podem ter peso negativo
- Detecta ciclos negativos

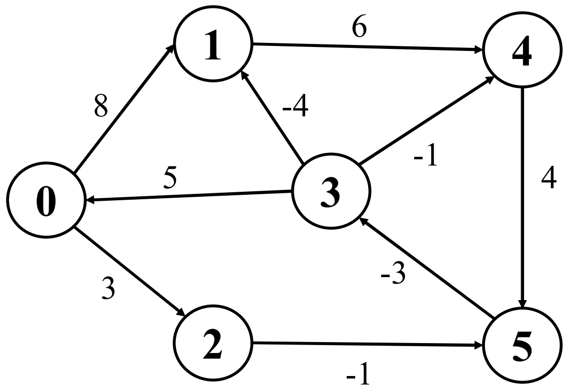
Ideia
- **Caminho simples:** um caminho sem ciclos tem no máximo (|V| - 1) arestas
- Logo, em no máximo (|V| - 1) passos é possível encontrar o menor caminho para todos os vértices

Método
- Parte de um vértice inicial e faz estimativas pessimistas para cada vértice
- Relaxa **todas** as arestas (|V| - 1) vezes
- Se após isso os pesos continuarem diminuindo, detectamos um ciclo negativo

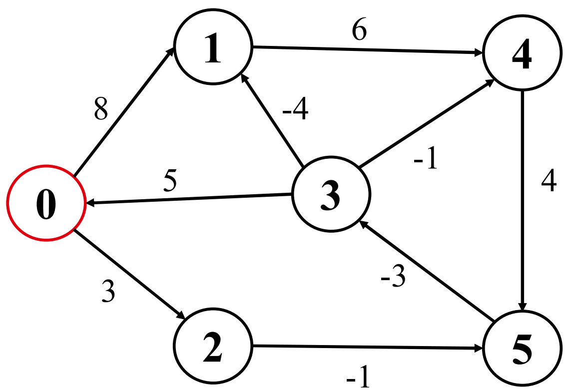
Algoritmo de Bellman-Ford

Exemplo

Quais são os menores caminhos com origem em 0?



Algoritmo de Bellman-Ford

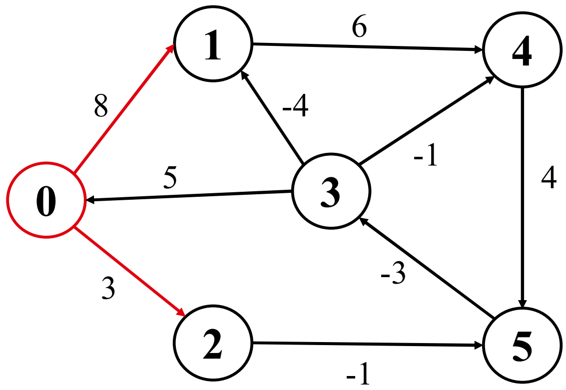


	0	1	2	3	4	5
dist	0	∞	∞	∞	∞	∞

Começando em 0 e fazendo estimativas pessimistas...

Algoritmo de Bellman-Ford

Iteração 1

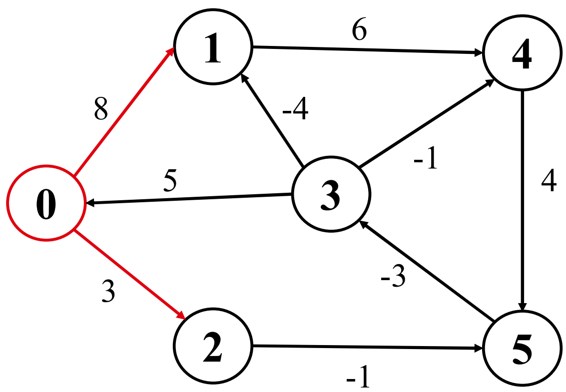


	0	1	2	3	4	5
dist	0	∞	∞	∞	∞	∞

Relaxando a partir de 0

Algoritmo de Bellman-Ford

Iteração 1

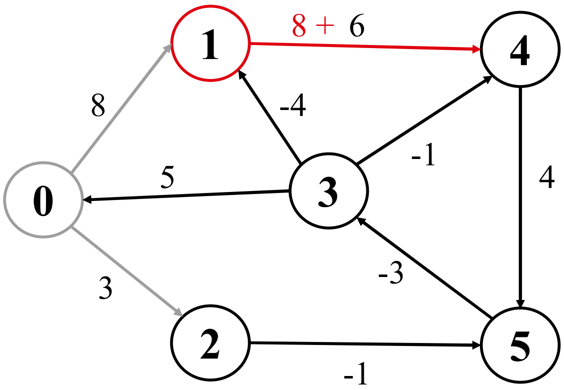


	0	1	2	3	4	5
dist	0	8	3	∞	∞	∞

Atualizamos vértices 1 e 2

Algoritmo de Bellman-Ford

Iteração 1

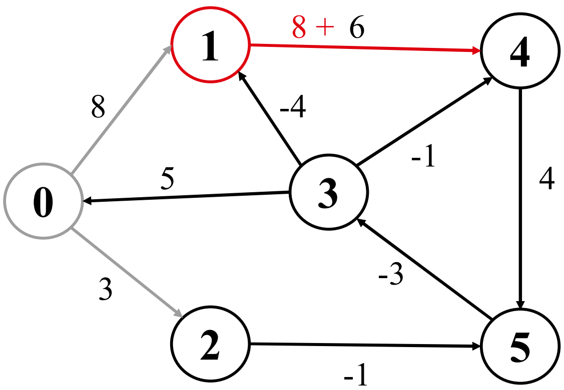


	0	1	2	3	4	5
dist	0	8	3	∞	∞	∞

Relaxando a partir de 1

Algoritmo de Bellman-Ford

Iteração 1

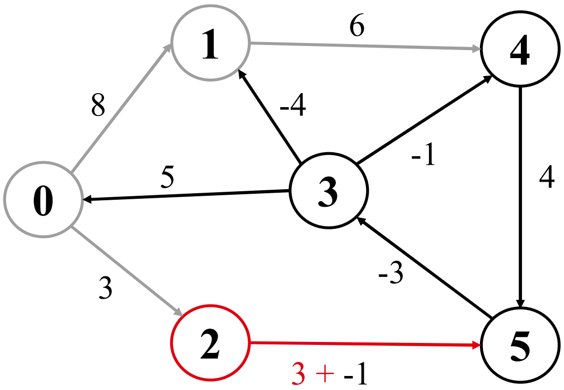


	0	1	2	3	4	5
dist	0	8	3	∞	14	∞

Atualizamos vértice 4

Algoritmo de Bellman-Ford

Iteração 1

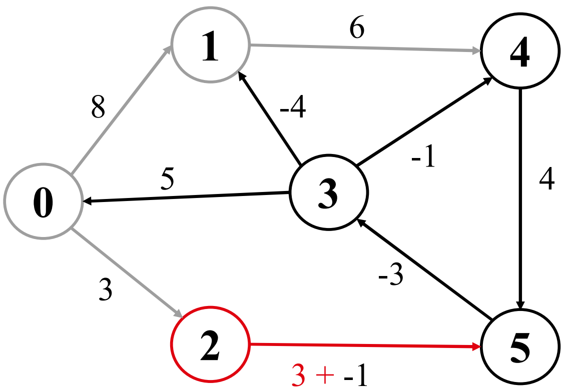


	0	1	**2**	3	4	5
dist	0	8	3	∞	14	∞

Relaxando a partir de 2

Algoritmo de Bellman-Ford

Iteração 1

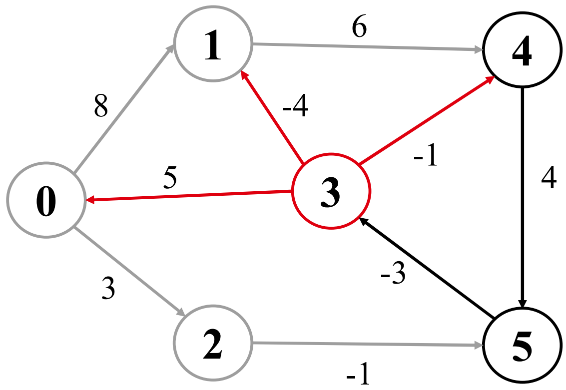


	0	1	**2**	3	4	5
dist	0	8	3	∞	14	2

Atualizamos vértice 5

Algoritmo de Bellman-Ford

Iteração 1

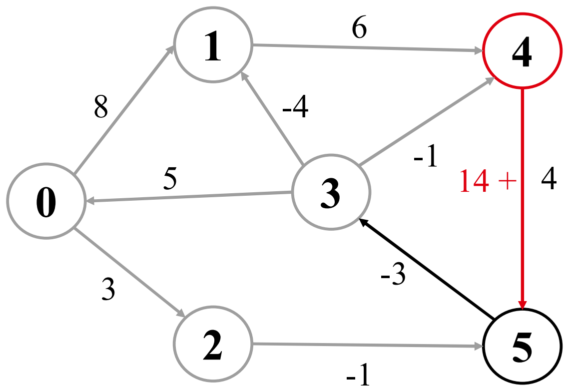


	0	1	2	**3**	4	5
dist	0	8	3	∞	14	2

Não conseguimos relaxar a partir de 3, pois não o descobrimos ainda

Algoritmo de Bellman-Ford

Iteração 1

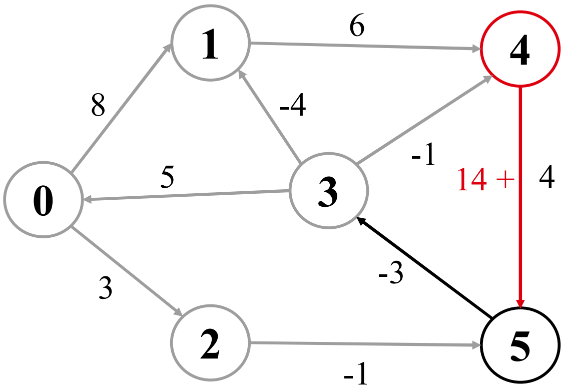


	0	1	2	3	**4**	5
dist	0	8	3	∞	14	2

Relaxando a partir de 4

Algoritmo de Bellman-Ford

Iteração 1

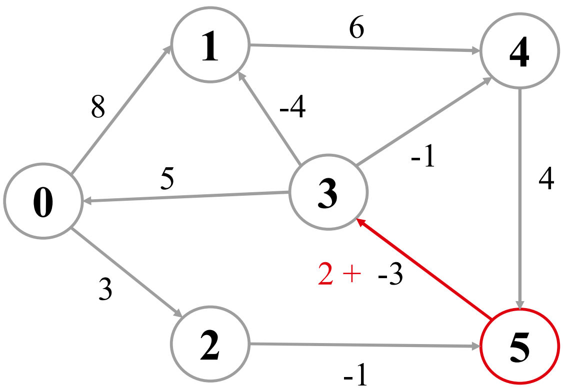


	0	1	2	3	**4**	5
dist	0	8	3	∞	14	2

Não atualizamos ninguém

Algoritmo de Bellman-Ford

Iteração 1

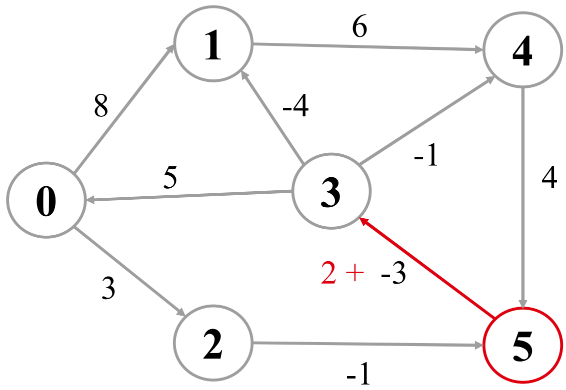


	0	1	2	3	4	**5**
dist	0	8	3	∞	14	2

Relaxando a partir de 5

Algoritmo de Bellman-Ford

Iteração 1



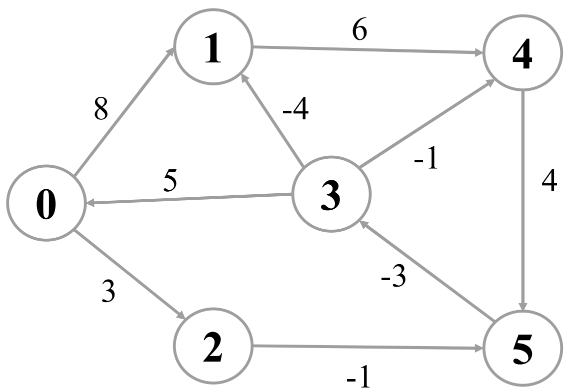
	0	1	2	3	4	5
dist	0	8	3	-1	14	2

Atualizamos vértice 3

59

Algoritmo de Bellman-Ford

Iteração 1



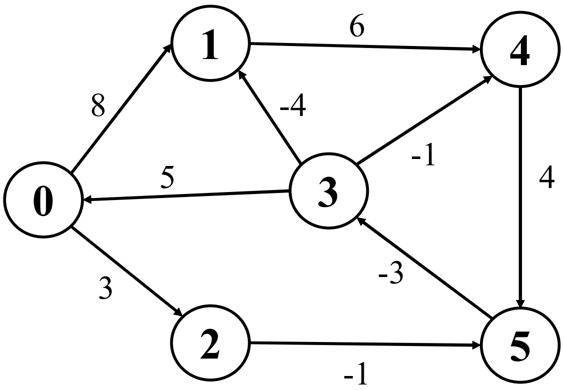
	0	1	2	3	4	5
dist	0	8	3	-1	14	2

Finalizada primeira iteração

60

Algoritmo de Bellman-Ford

Iteração 2

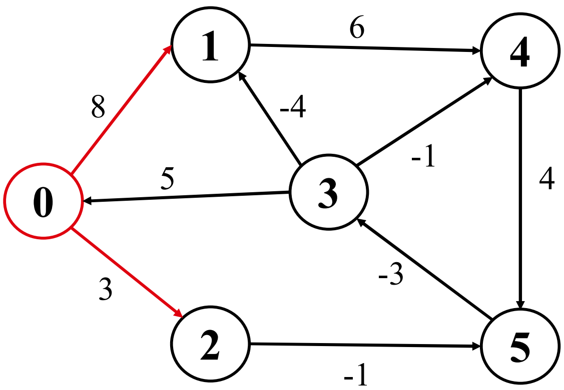


	0	1	2	3	4	5
dist	0	8	3	-1	14	2
dist	0	8	3	-1	14	2

Faremos outra iteração para tentar melhorar custos

Algoritmo de Bellman-Ford

Iteração 2

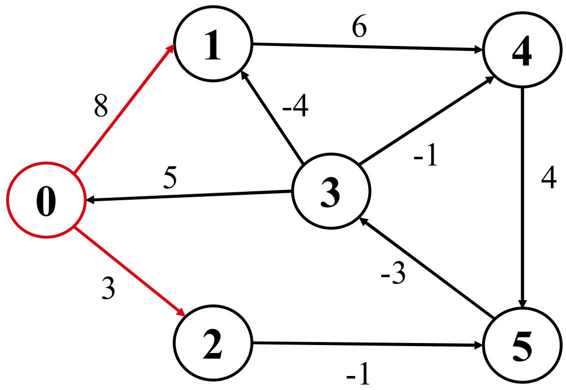


	0	1	2	3	4	5
dist	0	8	3	-1	14	2
dist	0	8	3	-1	14	2

Relaxando a partir de 0

Algoritmo de Bellman-Ford

Iteração 2

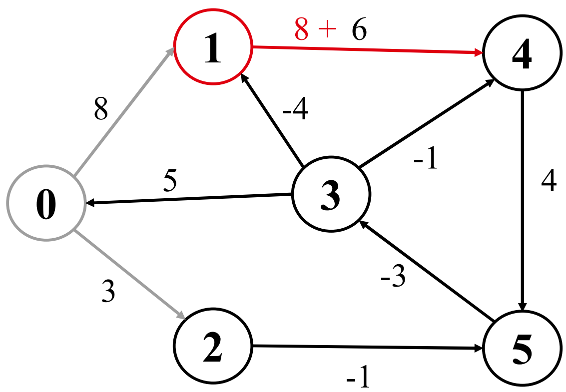


	0	1	2	3	4	5
dist	0	8	3	-1	14	2
dist	0	8	3	-1	14	2

Não atualizamos ninguém

Algoritmo de Bellman-Ford

Iteração 2

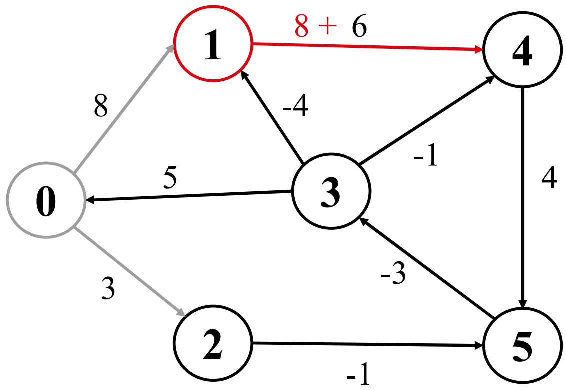


	0	**1**	2	3	4	5
dist	0	8	3	-1	14	2
dist	0	8	3	-1	14	2

Relaxando a partir de 1

Algoritmo de Bellman-Ford

Iteração 2

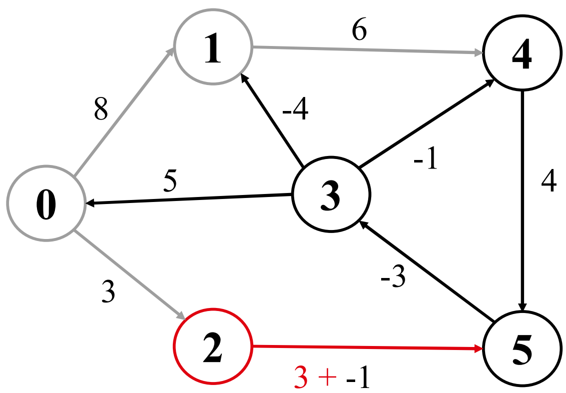


	0	1	2	3	4	5
dist	0	8	3	-1	14	2
dist	0	8	3	-1	14	2

Não atualizamos ninguém

Algoritmo de Bellman-Ford

Iteração 2

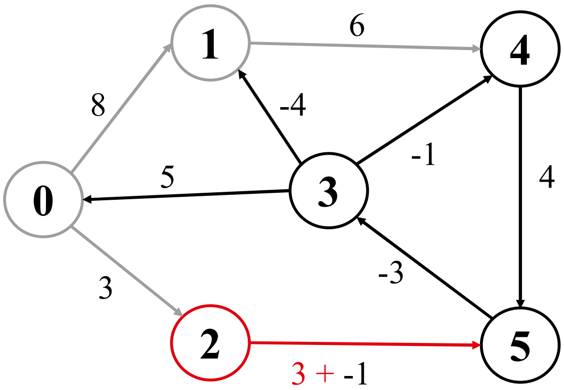


	0	1	2	3	4	5
dist	0	8	3	-1	14	2
dist	0	8	3	-1	14	2

Relaxando a partir de 2

Algoritmo de Bellman-Ford

Iteração 2

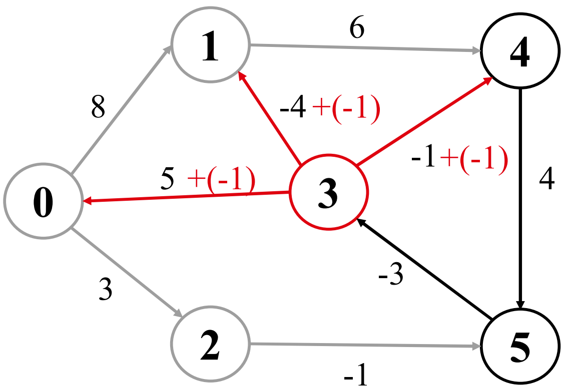


	0	1	**2**	3	4	5
dist	0	8	3	-1	14	2
dist	0	8	3	-1	14	2

Não atualizamos ninguém

Algoritmo de Bellman-Ford

Iteração 2

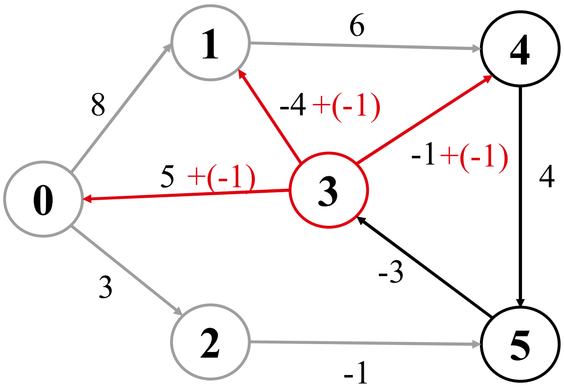


	0	1	2	**3**	4	5
dist	0	8	3	-1	14	2
dist	0	8	3	-1	14	2

Relaxando a partir de 3

Algoritmo de Bellman-Ford

Iteração 2

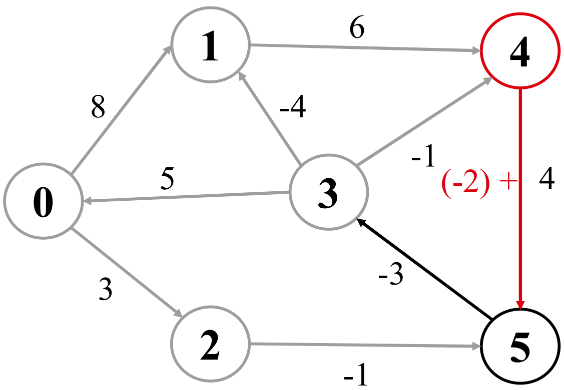


	0	1	2	**3**	4	5
dist	0	8	3	-1	14	2
dist	0	-5	3	-1	-2	2

Atualizamos vértices 1 e 4

Algoritmo de Bellman-Ford

Iteração 2

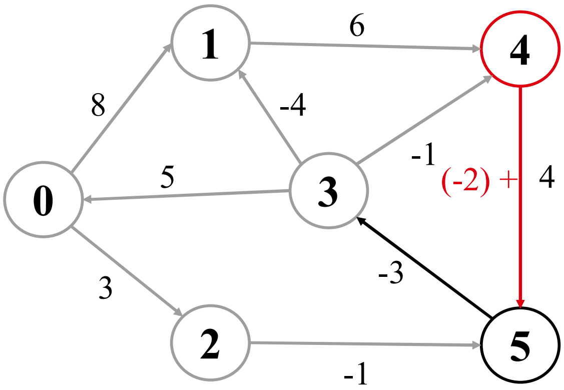


	0	1	2	3	**4**	5
dist	0	8	3	-1	14	2
dist	0	-5	3	-1	-2	2

Relaxando a partir de 4

Algoritmo de Bellman-Ford

Iteração 2

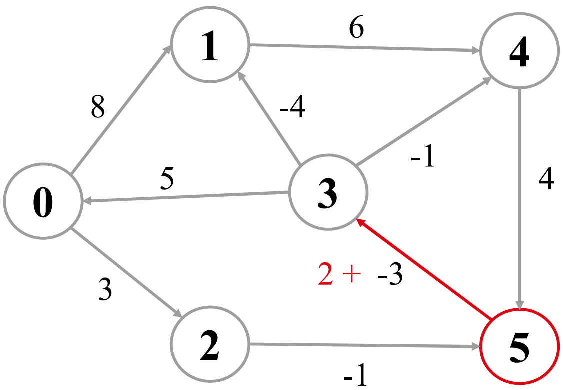


	0	1	2	3	**4**	5
dist	0	8	3	-1	14	2
dist	0	-5	3	-1	-2	2

Não atualizamos ninguém

Algoritmo de Bellman-Ford

Iteração 2

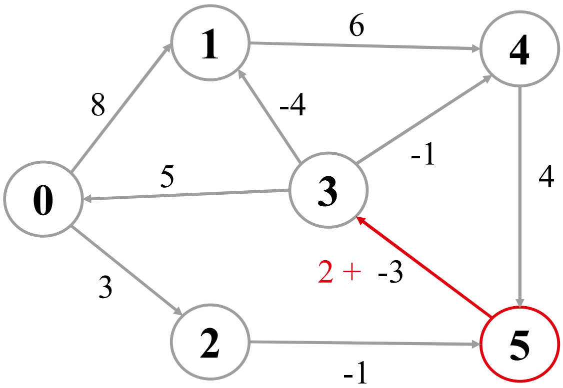


	0	1	2	3	4	**5**
dist	0	8	3	-1	14	2
dist	0	-5	3	-1	-2	2

Relaxando a partir de 5

Algoritmo de Bellman-Ford

Iteração 2

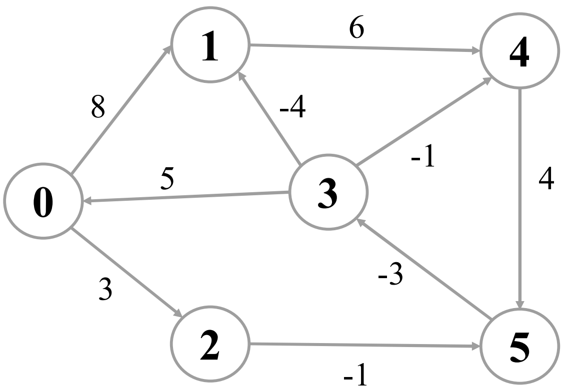


	0	1	2	3	4	5
dist	0	8	3	-1	14	2
dist	0	-5	3	-1	-2	2

Não atualizamos ninguém

Algoritmo de Bellman-Ford

Iteração 2

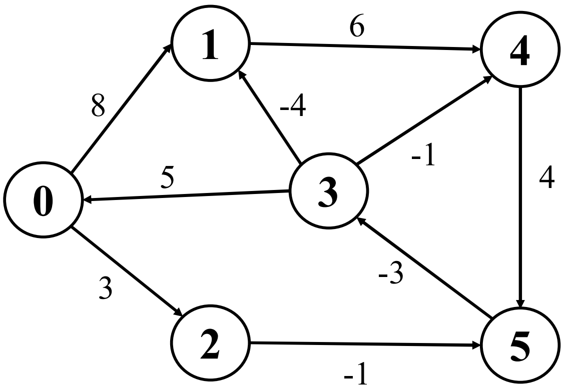


	0	1	2	3	4	5
dist	0	8	3	-1	14	2
dist	0	-5	3	-1	-2	2

Finalizamos segunda iteração

Algoritmo de Bellman-Ford

Iteração 3

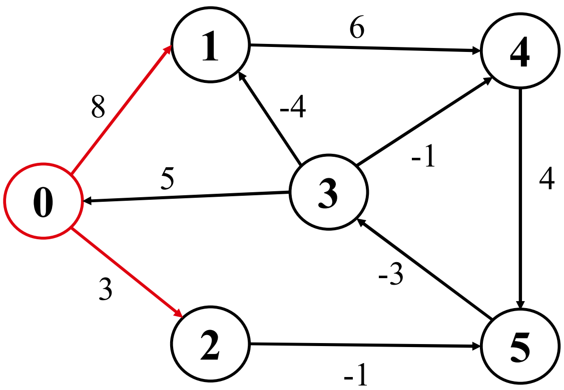


	0	1	2	3	4	5
dist	0	8	3	-1	14	2
dist	0	-5	3	-1	-2	2
dist	0	-5	3	-1	-2	2

Faremos outra iteração para tentar melhorar custos

Algoritmo de Bellman-Ford

Iteração 3

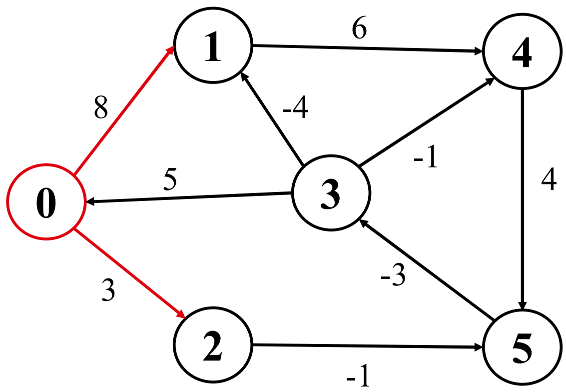


	0	1	2	3	4	5
dist	0	8	3	-1	14	2
dist	0	-5	3	-1	-2	2
dist	0	-5	3	-1	-2	2

Relaxando a partir de 0

Algoritmo de Bellman-Ford

Iteração 3



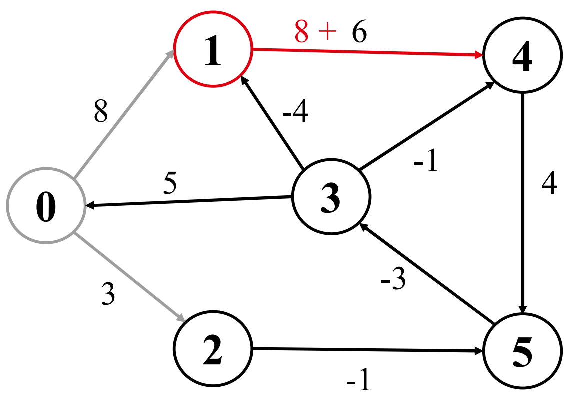
	0	1	2	3	4	5
dist	0	8	3	-1	14	2
dist	0	-5	3	-1	-2	2
dist	0	-5	3	-1	-2	2

Não atualizamos ninguém

77

Algoritmo de Bellman-Ford

Iteração 3



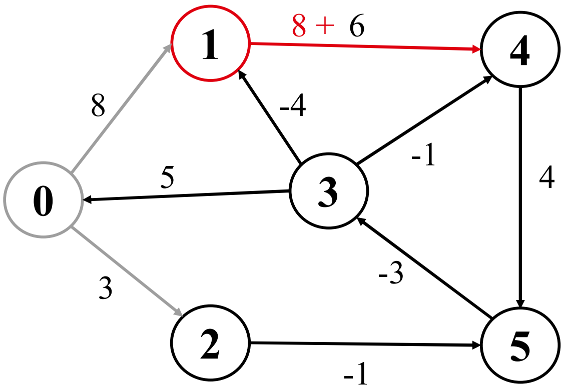
	0	**1**	2	3	4	5
dist	0	8	3	-1	14	2
dist	0	-5	3	-1	-2	2
dist	0	-5	3	-1	-2	2

Relaxando a partir de 1

78

Algoritmo de Bellman-Ford

Iteração 3

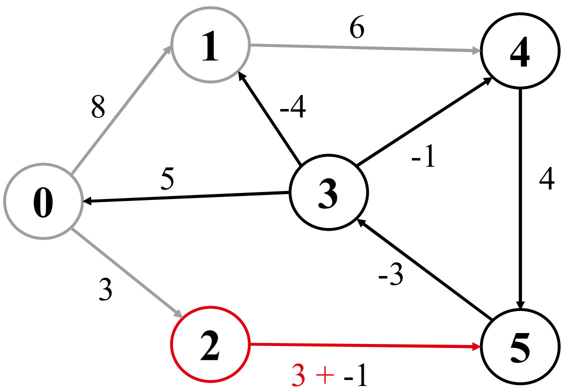


	0	**1**	2	3	4	5
dist	0	8	3	-1	14	2
dist	0	-5	3	-1	-2	2
dist	0	-5	3	-1	-2	2

Não atualizamos ninguém

Algoritmo de Bellman-Ford

Iteração 3

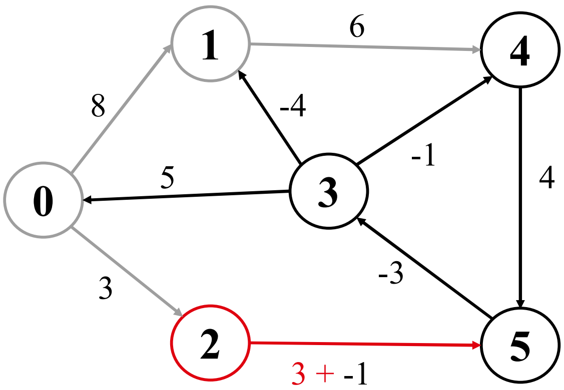


	0	1	**2**	3	4	5
dist	0	8	3	-1	14	2
dist	0	-5	3	-1	-2	2
dist	0	-5	3	-1	-2	2

Relaxando a partir de 2

Algoritmo de Bellman-Ford

Iteração 3

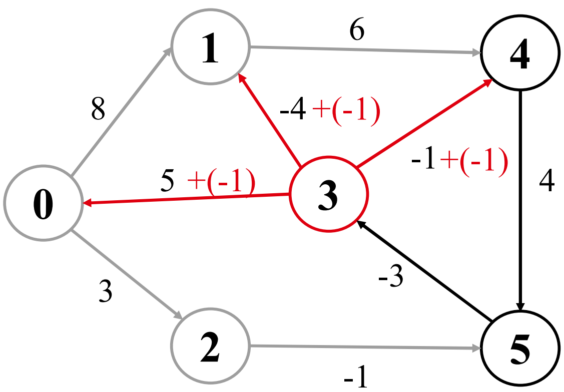


	0	1	**2**	3	4	5
dist	0	8	3	-1	14	2
dist	0	-5	3	-1	-2	2
dist	0	-5	3	-1	-2	2

Não atualizamos ninguém

Algoritmo de Bellman-Ford

Iteração 3

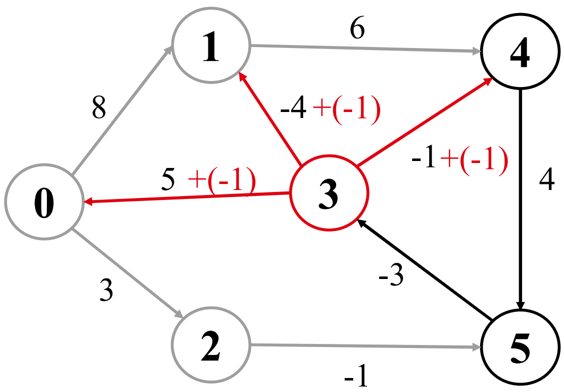


	0	1	2	**3**	4	5
dist	0	8	3	-1	14	2
dist	0	-5	3	-1	-2	2
dist	0	-5	3	-1	-2	2

Relaxando a partir de 3

Algoritmo de Bellman-Ford

Iteração 3

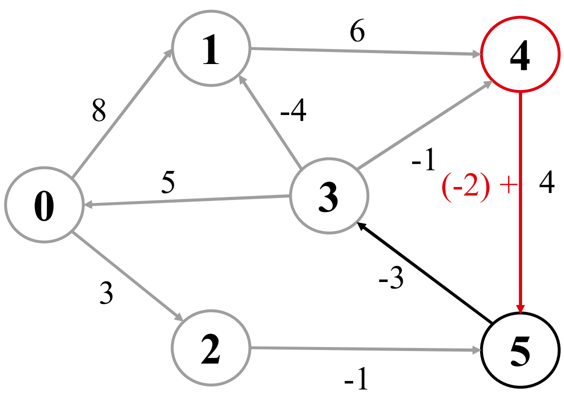


	0	1	2	**3**	4	5
dist	0	8	3	-1	14	2
dist	0	-5	3	-1	-2	2
dist	0	-5	3	-1	-2	2

Não atualizamos ninguém

Algoritmo de Bellman-Ford

Iteração 3

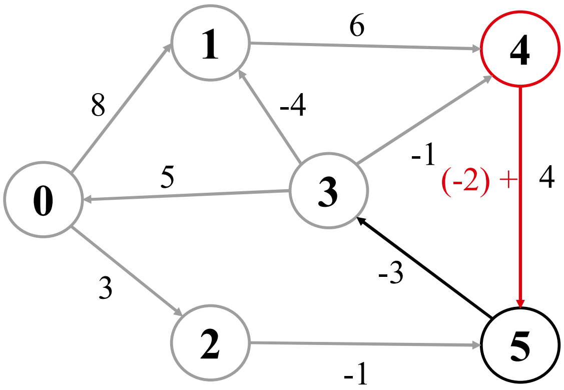


	0	1	2	3	**4**	5
dist	0	8	3	-1	14	2
dist	0	-5	3	-1	-2	2
dist	0	-5	3	-1	-2	2

Relaxando a partir de 4

Algoritmo de Bellman-Ford

Iteração 3

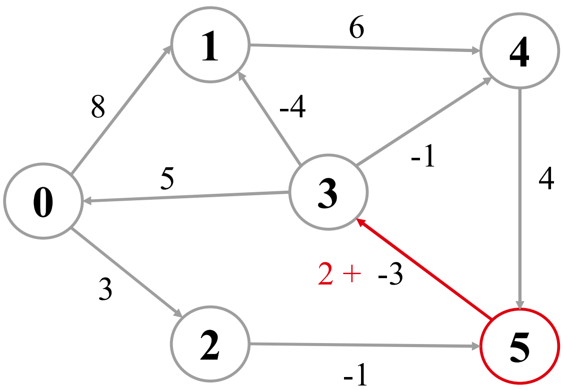


	0	1	2	3	**4**	5
dist	0	8	3	-1	14	2
dist	0	-5	3	-1	-2	2
dist	0	-5	3	-1	-2	2

Não atualizamos ninguém

Algoritmo de Bellman-Ford

Iteração 3

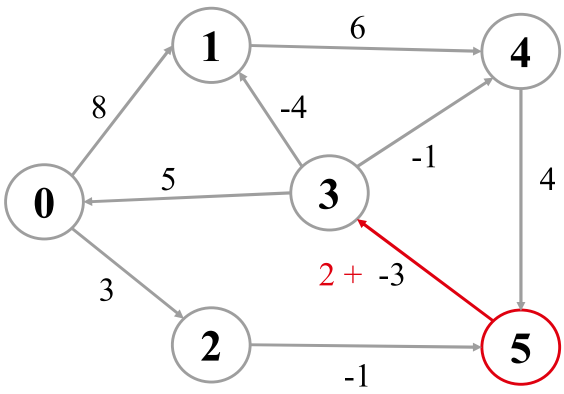


	0	1	2	3	4	**5**
dist	0	8	3	-1	14	2
dist	0	-5	3	-1	-2	2
dist	0	-5	3	-1	-2	2

Relaxando a partir de 5

Algoritmo de Bellman-Ford

Iteração 3

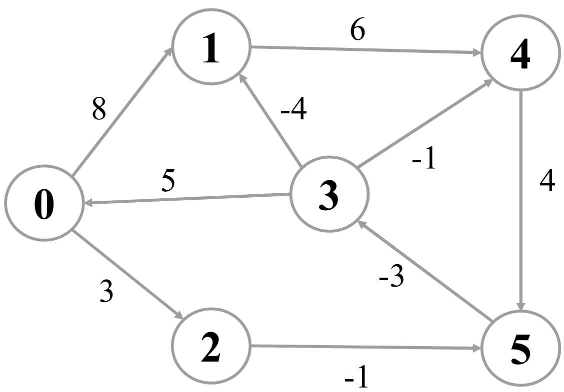


	0	1	2	3	4	**5**
dist	0	8	3	-1	14	2
dist	0	-5	3	-1	-2	2
dist	0	-5	3	-1	-2	2

Não atualizamos ninguém

Algoritmo de Bellman-Ford

Iteração 3

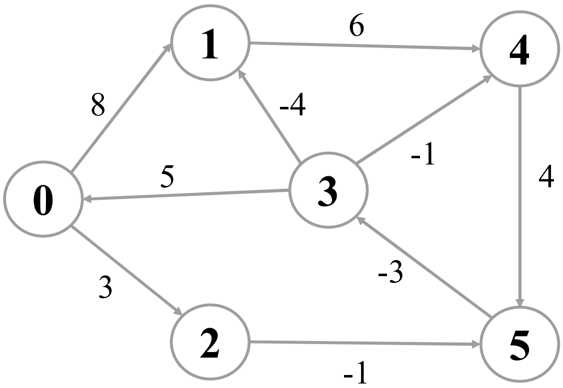


	0	1	2	3	4	5
dist	0	8	3	-1	14	2
dist	0	-5	3	-1	-2	2
dist	0	-5	3	-1	-2	2

Finalizamos terceira iteração

Algoritmo de Bellman-Ford

Iteração 3

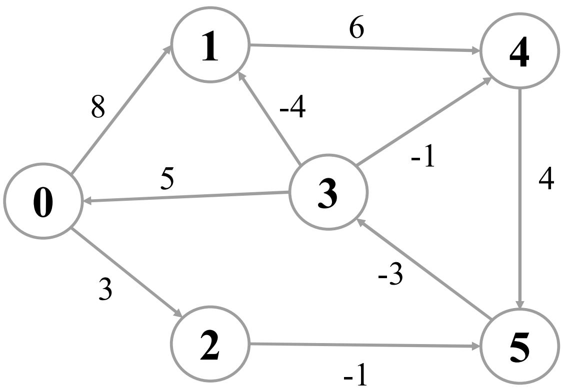


	0	1	2	3	4	5
dist	0	8	3	-1	14	2
dist	0	-5	3	-1	-2	2
dist	0	-5	3	-1	-2	2

Os valores não mudaram desde a última iteração, podemos finalizar pois já convergiu!

Algoritmo de Bellman-Ford

Iteração 3

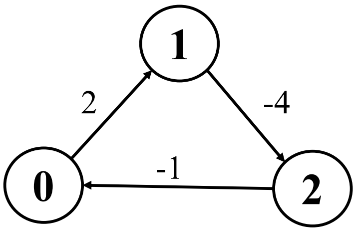


	0	1	2	3	4	5
dist	0	8	3	-1	14	2
dist	0	-5	3	-1	-2	2
dist	0	-5	3	-1	-2	2

Estes são os valores dos menores caminhos a partir de 0.

Como o Bellman-Ford detecta ciclos negativos?

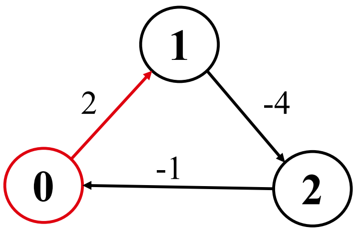
Iteração 1



	0	1	2
dist	0	∞	∞

Como o Bellman-Ford detecta ciclos negativos?

Iteração 1

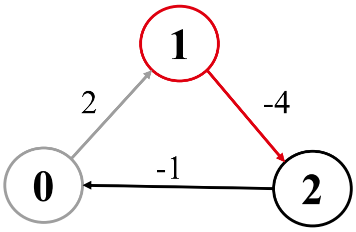


	0	1	2
dist	0	2	∞

Relaxa a partir do vértice 0, atualiza vértice 1

Como o Bellman-Ford detecta ciclos negativos?

Iteração 1

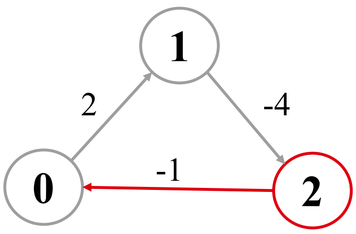


	0	**1**	2
dist	0	2	-2

Relaxa a partir do vértice 1, atualiza vértice 2

Como o Bellman-Ford detecta ciclos negativos?

Iteração 1

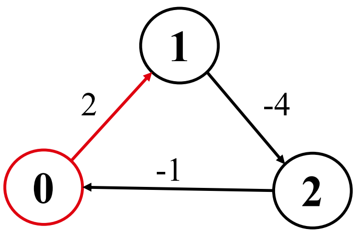


	0	1	**2**
dist	-3	2	-2

Relaxa a partir do vértice 2, atualiza vértice 0

Como o Bellman-Ford detecta ciclos negativos?

Iteração 2

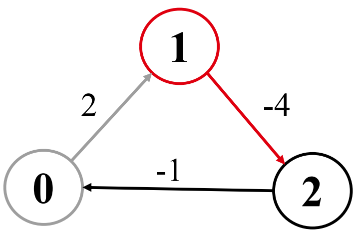


	0	1	2
dist	-3	2	-2
dist	-3	-1	-2

Relaxa a partir do vértice 0, atualiza vértice 1

Como o Bellman-Ford detecta ciclos negativos?

Iteração 2

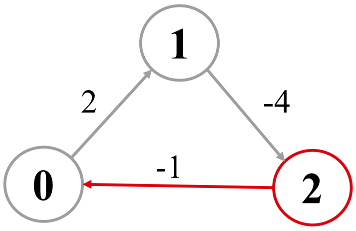


	0	**1**	2
dist	-3	2	-2
dist	-3	-1	-5

Relaxa a partir do vértice 1, atualiza vértice 2

Como o Bellman-Ford detecta ciclos negativos?

Iteração 2

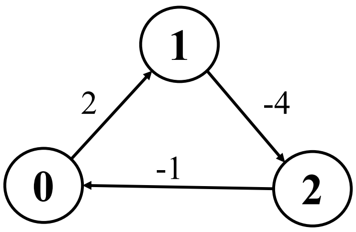


	0	1	**2**
dist	-3	2	-2
dist	-6	-1	-5

Relaxa a partir do vértice 2, atualiza vértice 0

Como o Bellman-Ford detecta ciclos negativos?

Iteração 3

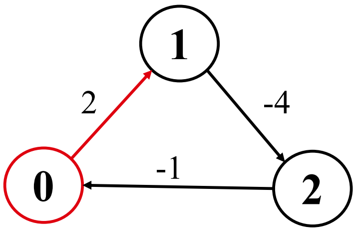


	0	1	2
dist	-3	2	-2
dist	-6	-1	-5
dist	-6	-1	-5

Já concluímos |V| - 1 iterações, vamos testar se ainda é possível melhorar a resposta.

Como o Bellman-Ford detecta ciclos negativos?

Iteração 3



	0	1	2
dist	-3	2	-2
dist	-6	-1	-5
dist	-6	-4	-5

Tentamos relaxar a partir do 0 e conseguimos atualizar!
Logo o grafo possui ciclo negativo.

Algoritmo de Bellman-Ford

- **Produz solução ótima?**

  Sim, produz sempre o caminho mínimo em um grafo com pesos positivos ou negativos, em até (|V| - 1) iterações.

- **Por que funciona?**
  - No pior caso, o caminho mais curto pode ter no máximo |V| - 1 arestas, pois selecionando mais formaremos ciclos
  - Relaxando arestas |V| - 1 vezes garantimos que as estimativas foram atualizadas a valores ótimos
  - Após |V| - 1 um ciclo é detectado se os valores continuarem a mudar

Algoritmo de Bellman-Ford - Implementação

- Em geral, utiliza uma lista de arestas para iterar sobre todas a cada ciclo de execução.
- Percorre todas as arestas |V| - 1 vezes, realizando relaxamento a partir delas.

Algoritmo de Bellman-Ford - Complexidade

- As operações envolvidas são:
  - Percorrer todos as arestas: O(|A|)
  - Fazer |V| - 1 iterações: O(|V|)

- Como percorremos todas as arestas |V| - 1 vezes, a complexidade de tempo é:
  - O(VE)
    - V = número de vértices
    - E = número de arestas

Algoritmo de Floyd-Warshall

https://devsuperior.com.br

Prof. Dr. Nelio Alves

Motivação

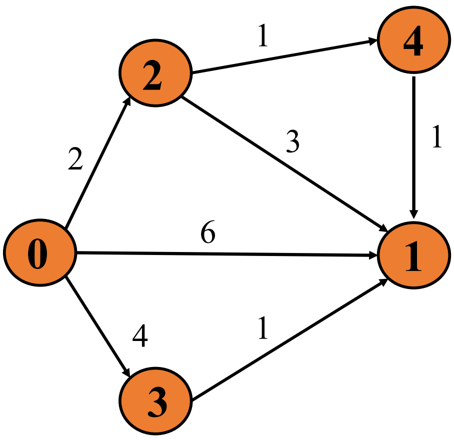
- Suponha que um grafo orientado ponderado representa os possíveis vôos de uma companhia aérea conectando pares de cidades
- Queremos construir uma tabela com as **melhores rotas**, ou os menores caminhos, entre todas as cidades

Como resolver esse problema?

Caminhos Mais Curtos de Todos os Pares

- Soluções?
  - Utilizar o algoritmo de Dijkstra, calculando as distâncias considerando cada vértice como uma origem.
- Solução mais direta
  - **Algoritmo de Floyd-Warshall**

Vértices intermediários em caminhos



Menor caminho entre o vértice 0 e o vértice 1?

- (0 ⟶ 1) = ?

Podemos tentar os seguintes caminhos:

- (0 ⟶ 1) = 6
- (0 ⟶ 1) = (0 ⟶ **2**) + (**2** ⟶ 1) = 5
- (0 ⟶ 1) = (0 ⟶ **3**) + (**3** ⟶ 1) = 5
- (0 ⟶ 1) = (0 ⟶ **4**) + (**4** ⟶ 1) = 4

↓

(0 ⟶ **2**) + (**2** ⟶ 4) = 3

Ou seja, um caminho de i até j, pode passar por **vértices intermediários** k!

dist[i][j] = dist[i][k] + dist[k][j]

Algoritmo de Floyd-Warshall

Características

- Caminhos mais curtos de <u>todos os pares</u>
- Arestas podem ter <u>peso negativo</u>
- Utiliza uma matriz A_{ij} de tamanho |V|x|V| para calcular o custo entre vértices i e j

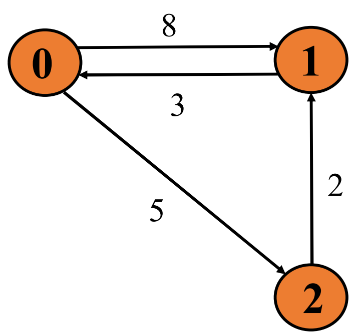
Ideia

- Um caminho mais curto de um vértice i até outro vértice j pode passar por um vértice intermediário k
- Verificar então se um caminho passando por k é mais curto que o caminho direto entre i e j (relaxamento)

Método

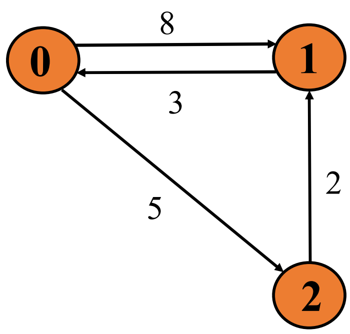
- Preencher matriz com distâncias conhecidas, e infinitos para distâncias desconhecidas
- Iterar sobre todos os pares possíveis (i, j), para cada vértice intermediário k
- Atualizar matriz de distâncias se o caminho passando por k for mais curto

Algoritmo de Floyd-Warshall



	0	1	2
0			
1			
2			

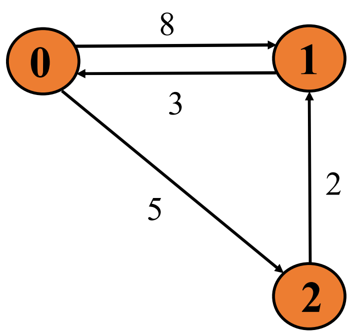
Algoritmo de Floyd-Warshall



- Iremos preencher uma matriz A inicial com os custos conhecidos.
- Diagonal é zerada

	0	1	2
0			
1			
2			

Algoritmo de Floyd-Warshall

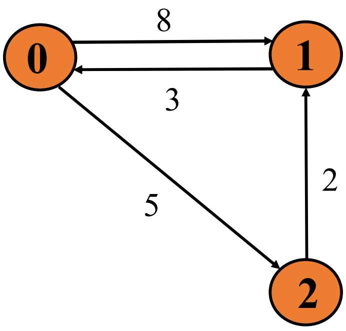


- Iremos preencher uma matriz $A^{(0)}$ com os custos conhecidos.
- Diagonal é zerada

$A^{(0)}$

	0	1	2
0	0	8	5
1	3	0	∞
2	∞	2	0

Algoritmo de Floyd-Warshall



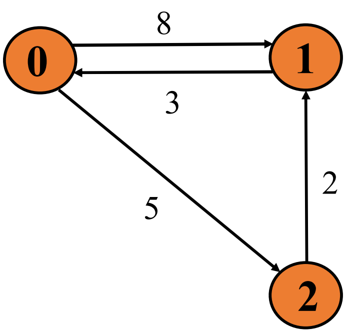
$A^{(0)}$

	0	1	2
0	0	8	5
1	3	0	∞
2	∞	2	0

- Matriz é percorrida n = |V| vezes
- A cada iteração tentaremos usar um vértice k ($0 \le k < n$) como **intermediário**
- Verificaremos se um caminho entre um par de vértices (i, j) pode ser encurtado passando por k

$$\boxed{A[i][j] = \min(A[i][j], A[i][k] + A[k][j])}$$

Algoritmo de Floyd-Warshall



$A^{(0)}$

	0	1	2
0	0	8	5
1	3	0	∞
2	∞	2	0

Resumindo:

$$\boxed{A[i][j] = \min(A[i][j], A[i][k] + A[k][j])}$$

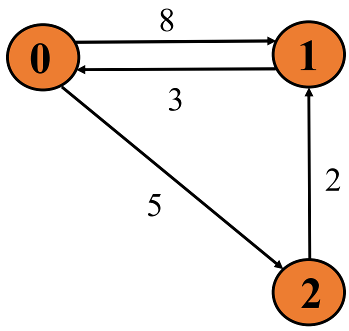
- Iterações:

  k = 1, 2, 3

- Em cada iteração calculamos:

  $A^{(1)}, A^{(2)}, A^{(3)}$

Algoritmo de Floyd-Warshall



$A[i][j] = \min(A[i][j], A[i][k] + A[k][j])$

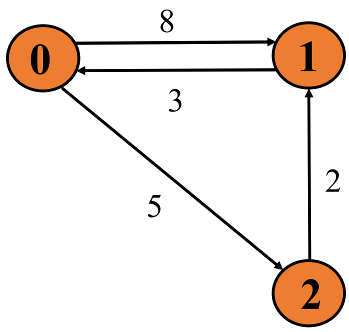
$A[0][0] = \min(A[0][0], A[0][0] + A[0][0])$

$A^{(1)}$

	0	1	2
0	0	8	5
1	3	0	∞
2	∞	2	0

i = 0, j = 0, k = 0

Algoritmo de Floyd-Warshall



$A[i][j] = \min(A[i][j], A[i][k] + A[k][j])$

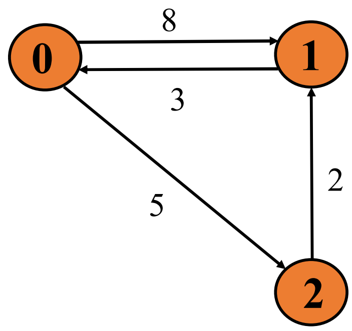
$A[0][0] = \min(A[0][1], A[0][0] + A[0][1])$

$A^{(1)}$

	0	1	2
0	0	8	5
1	3	0	∞
2	∞	2	0

i = 0, j = 1, k = 0

Algoritmo de Floyd-Warshall



A[i][j] = min(A[i][j], A[i][**k**] + A[**k**][j])

↓

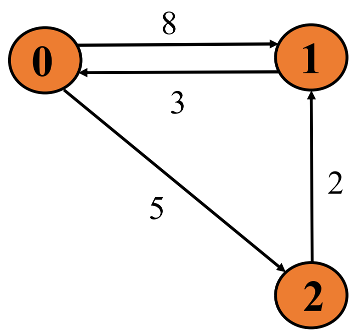
A[0][0] = min(A[0][2], A[0][**0**] + A[**0**][2])

$A^{(1)}$

	0	1	2
0	0	8	5
1	3	0	∞
2	∞	2	0

i = 0, j = 2, k = 0

Algoritmo de Floyd-Warshall



A[i][j] = min(A[i][j], A[i][**k**] + A[**k**][j])

↓

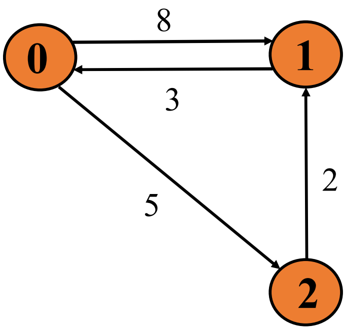
A[0][0] = min(A[1][0], A[1][**0**] + A[**0**][0])

$A^{(1)}$

	0	1	2
0	0	8	5
1	3	0	∞
2	∞	2	0

i = 1, j = 0, k = 0

Algoritmo de Floyd-Warshall



$A^{(1)}$

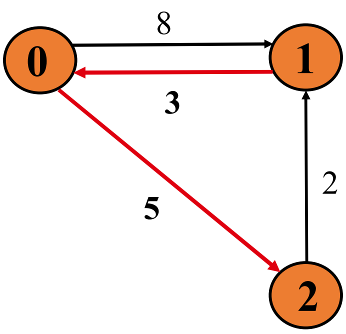
	0	1	2
0	0	8	5
1	3	0	∞
2	∞	2	0

A[i][j] = min(A[i][j], A[i][k] + A[k][j])

A[0][0] = min(A[1][1], A[1][0] + A[0][1])

i = 1, j = 1, k = 0

Algoritmo de Floyd-Warshall



$A^{(1)}$

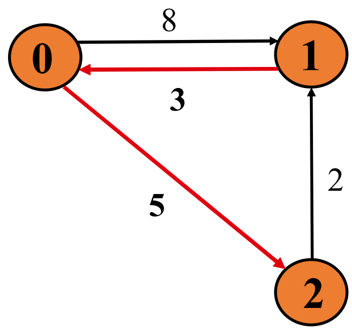
	0	1	2
0	0	8	**5**
1	**3**	0	∞
2	∞	2	0

A[i][j] = min(A[i][j], A[i][k] + A[k][j])

A[0][0] = min(A[1][2], A[1][0] + A[0][2])

i = 1, j = 2, k = 0

Algoritmo de Floyd-Warshall



A[i][j] = min(A[i][j], A[i][**k**] + A[**k**][j])

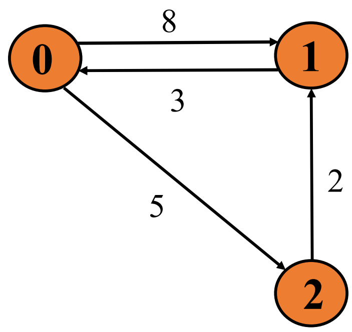
A[0][0] = min(A[1][2], A[1][**0**] + A[**0**][2])

$A^{(1)}$

	0	1	2
0	0	8	**5**
1	**3**	0	8
2	∞	2	0

i = 1, j = 2, k = 0

Algoritmo de Floyd-Warshall



A[i][j] = min(A[i][j], A[i][**k**] + A[**k**][j])

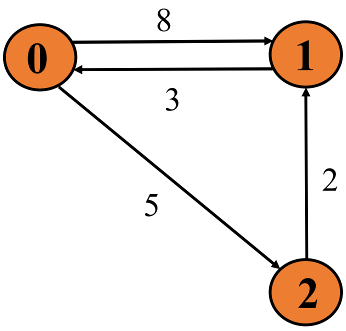
A[0][0] = min(A[2][0], A[2][**0**] + A[**0**][0])

$A^{(1)}$

	0	1	2
0	0	8	5
1	3	0	8
2	∞	2	0

i = 2, j = 0, k = 0

Algoritmo de Floyd-Warshall



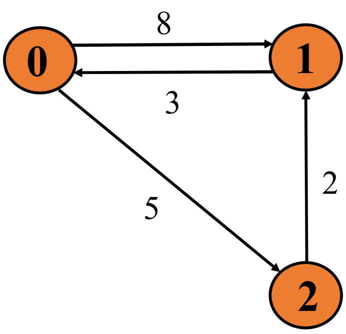
$$A[i][j] = \min(A[i][j], A[i][k] + A[k][j])$$

$$A[0][0] = \min(A[2][1], A[2][0] + A[0][1])$$

$A^{(1)}$	0	1	2
0	0	8	5
1	3	0	8
2	∞	2	0

i = 2, j = 1, k = 0

Algoritmo de Floyd-Warshall



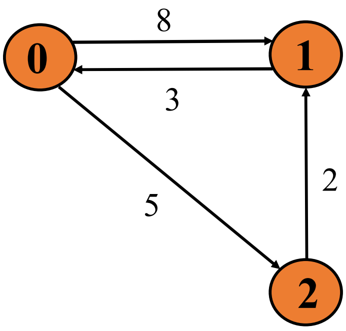
$$A[i][j] = \min(A[i][j], A[i][k] + A[k][j])$$

$$A[0][0] = \min(A[2][2], A[2][0] + A[0][2])$$

$A^{(1)}$	0	1	2
0	0	8	5
1	3	0	8
2	∞	2	0

i = 2, j = 2, k = 0

Algoritmo de Floyd-Warshall

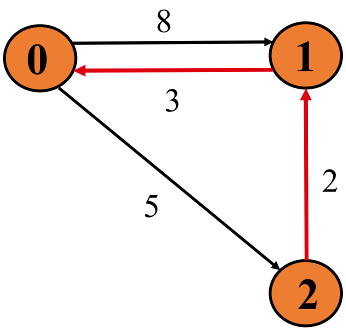


$A^{(1)}$

	0	1	2
0	0	8	5
1	3	0	8
2	∞	2	0

- Ao fim da iteração k = 0, temos os custos dos caminhos mais curtos entre i e j que passam pelo vértice 0
- Repetimos o processo para k = 1 e k = 2

Algoritmo de Floyd-Warshall



$A^{(2)}$

	0	1	2
0	0	8	5
1	3	0	8
2	∞	2	0

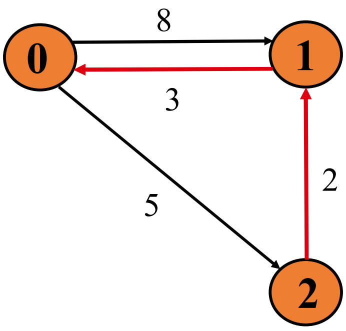
$$\boxed{A[i][j] = \min(A[i][j], A[i][k] + A[k][j])}$$

$$\downarrow$$

$$A[2][0] = \min(A[2][0], A[2][1] + A[1][0])$$

k = 1

Algoritmo de Floyd-Warshall



$A[i][j] = \min(A[i][j], A[i][k] + A[k][j])$

$$\downarrow$$

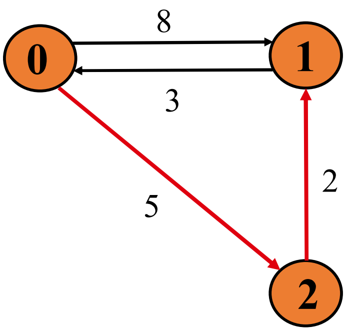
$A[2][0] = \min(A[2][0], A[2][1] + A[1][0])$

$A^{(2)}$

	0	1	2
0	0	8	5
1	3	0	8
2	5	2	0

k = 1

Algoritmo de Floyd-Warshall



$A[i][j] = \min(A[i][j], A[i][k] + A[k][j])$

$$\downarrow$$

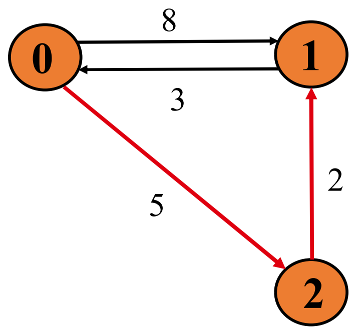
$A[0][1] = \min(A[0][1], A[0][2] + A[2][1])$

$A^{(3)}$

	0	1	2
0	0	8	5
1	3	0	8
2	5	2	0

k = 2

Algoritmo de Floyd-Warshall



$A^{(3)}$

	0	1	2
0	0	7	5
1	3	0	8
2	5	2	0

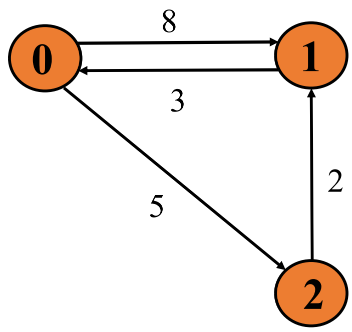
$$\boxed{A[i][j] = \min(A[i][j], A[i][k] + A[k][j])}$$

$$\downarrow$$

$$A[0][1] = \min(A[0][1], A[0][2] + A[2][1])$$

k = 2

Algoritmo de Floyd-Warshall



$A^{(3)}$

	0	1	2
0	0	7	5
1	3	0	8
2	5	2	0

- Na iteração k, a matriz $A^{(k)}$ diz os custos dos menores caminhos que passam pelos vértices 1, 2, …, k
- A matriz final $A^{(n)}$ diz os custos dos menores caminhos entre cada par de vértices (i, j)

Algoritmo de Floyd-Warshall - Implementação

Algoritmo

- Inicializar matriz com custos conhecidos e a diagonal zerada
- Fazem-se 3 fors aninhados para variar os valores (i, j, k)
  - k = [0, n - 1]
    - i = [0, n - 1]
      - j = [0, n - 1]

- Para cada trio de valores (i, j, k), tenta-se relaxamento segundo a regra:

A[i][j] = min(A[i][j], A[i][k] + A[k][j])

Algoritmo de Floyd-Warshall - Complexidade

Complexidade: $O(|V|^3)$

- Por que?
  - Operações envolvidas:
    - Percorrer a matriz $O(|V|^2)$
      - Relaxamento: $O(1)$
    - Realizar iterações $O(|V|)$

  - Logo, realizando-se $O(|V|)$ iterações, onde se percorre a matriz em cada uma…

$$O(|V|) * O(|V|^2)$$

$$O(|V|^3)$$

Algoritmo de Floyd-Warshall - Observações

- Que paradigma de resolução de problemas o algoritmo usa?
  - **Programação Dinâmica!**
    - Abordagem iterativa onde calculamos estados de uma função f(i, j, k) representando a distância ótima
- Para que tipo de grafos o Floyd-Warshall é bom?
  - **Grafos densos:** muito mais arestas que vértices
  - Para grafos esparsos o Algoritmo de Johnson é mais adequado
- É possível recuperar o caminho guardando o antecessor no relaxamento