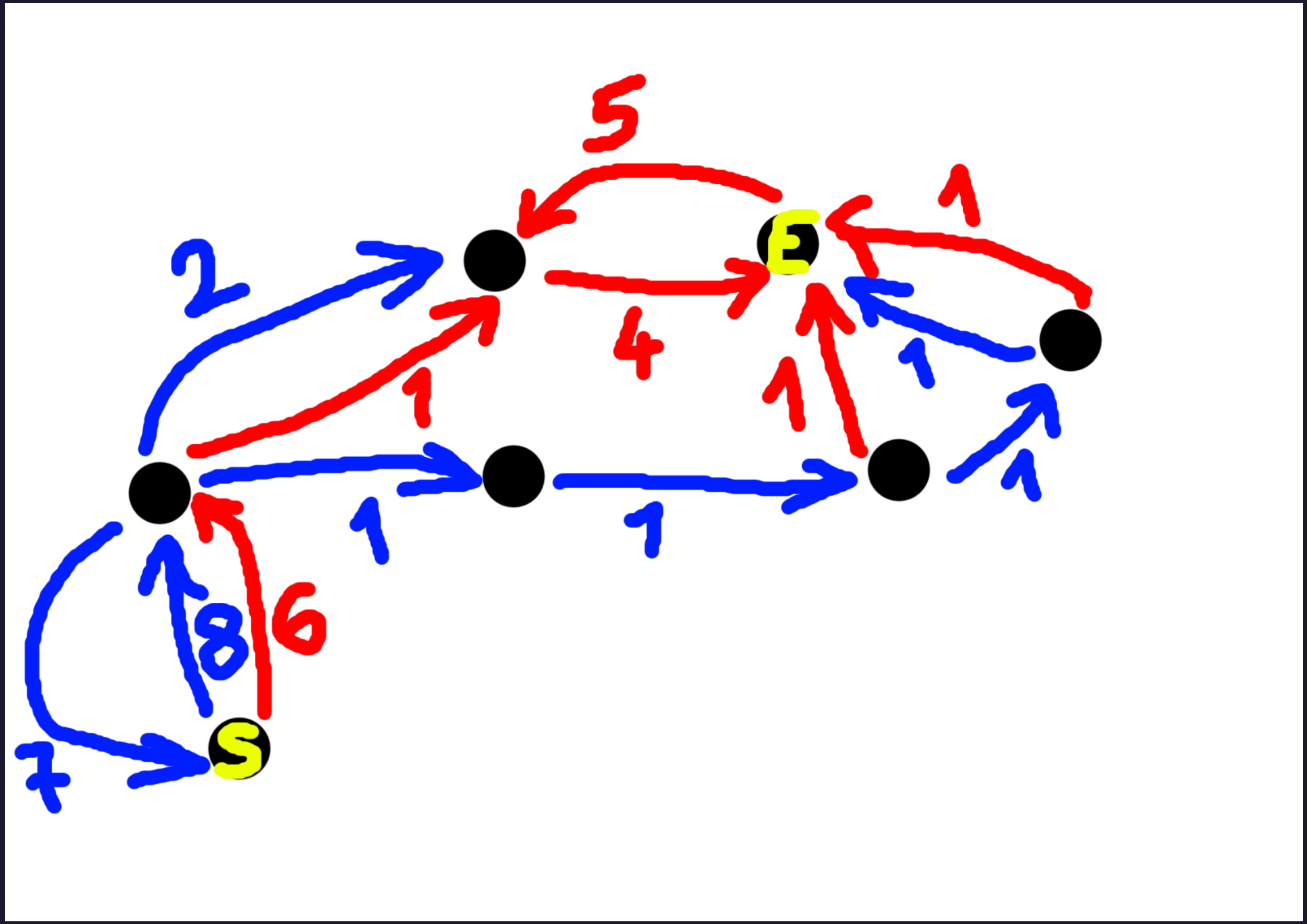


Caminho Mínimo de Fonte Única

Teoria dos Grafos

André Luiz Satoshi Kawamoto

AGENDA

- Motivação
- Conceito
- Raciocínio
- Algoritmo de Dijkstra
  - Limitação
- Algoritmo de Bellman-Ford
- Considerações Finais
- Exercícios

Motivação

- Estou numa cidade e quero chegar em outra percorrendo menos quilômetros (ou pagando menos pedágio, ou passando por menos buracos)
- Qual rota um pacote da Internet deveria seguir entre diversos dispositivos de comunicação para chegar com o menor tempo ao destino
- Basicamente qualquer situação em que se deseja traçar uma rota em que alguma característica quantificável associada a cada trecho individual deve ser minimizada (tempo, custo, distância, por exemplo)

Conceito

- O problema de encontrar o caminho mínimo consiste na minimização do *custo* de travessia de um grafo entre dois nós (ou vértices). Esse custo é dado pela soma dos pesos das arestas percorridas.

- Formalmente:

- Seja G um grafo valorado, e uma função de peso $f: A \rightarrow \mathbb{R}$ (uma função que mapeia cada aresta do conjunto A para um valor real)

- Dado um vértice qualquer v e um outro vértice v', e um caminho P entre v e v', o caminho mínimo é aquele tal que $\sum_{p \in P} f(p)$ é mínimo entre v e v'

- (A soma de todos os pesos pertencentes ao caminho P é mínima)

Raciocínio

- Em um grafo com arestas não valoradas, é fácil obter o caminho mais curto que sai de um vértice u para um vértice v
    - Busca em Largura a partir do vértice u.

- Para grafos com arestas valoradas, alguns algoritmos são bem conhecidos:
    - Dijkstra, quando todas as arestas possuem peso não-negativo
    - Bellman-Ford, quando existem arestas de peso negativo
    - Floyd-Warshall, para descobrir os caminhos mínimos entre todos os pares de vértices

Algoritmo de Dijkstra

- Edsger Dijkstra propôs um algoritmo que calcula o Caminho Mínimo em um grafo qualquer (dirigido ou não dirigido) com arestas de **peso não negativo**.

- Esse algoritmo é muito similar ao algoritmo de Prim (para MST)
  - 1 Vetor de distâncias V
  - 1 Vetor de predecessores π
  - 1 vértice inicial S

Algoritmo de Dijkstra

```
Início
        para todo v ∈ V[G]
                d[v]← ∞                         //distância do vértice v é infinita
                pred[v] ← -1                    //o predecessor de v no caminho é nulo
        d[s] ← 0                //a distância do vértice inicial para si mesmo é 0 (zero)
        Q ← V[G]                //conjunto de todos os nós do vértice
        enquanto Q ≠ ∅
                u ← extrair-min(Q)                      //Q ← Q - {u}
                para cada v adjacente a u
                        se d[v] > d[u] + w(u, v) *      //se a inclusão de v é favorável
                                d[v] ← d[u] + w(u, v)   //relaxe (u, v’)
                                pred[v] ← u
Fim
```

Algoritmo de Dijkstra

- A condicional do algoritmo de Dijkstra

```
se d[v] > d[u] + w(u, v) * //se a inclusão de v é favorável
   d[v] ← d[u] + w(u, v)   //relaxe (u, v')
```

- O termo 'relaxar' é uma analogia entre o caminho mais curto e uma mola. No início a mola está esticada. Na medida que se encontra caminhos mais curtos, essa mola fica mais 'relaxada'.

- Na prática, essa condição diz: "Se o peso para atingir o vértice v é **maior** que o peso até atingir o vértice u somado com o peso de u até v, então compensa atualizar o peso até v utilizando vértice u como intermediário no caminho"

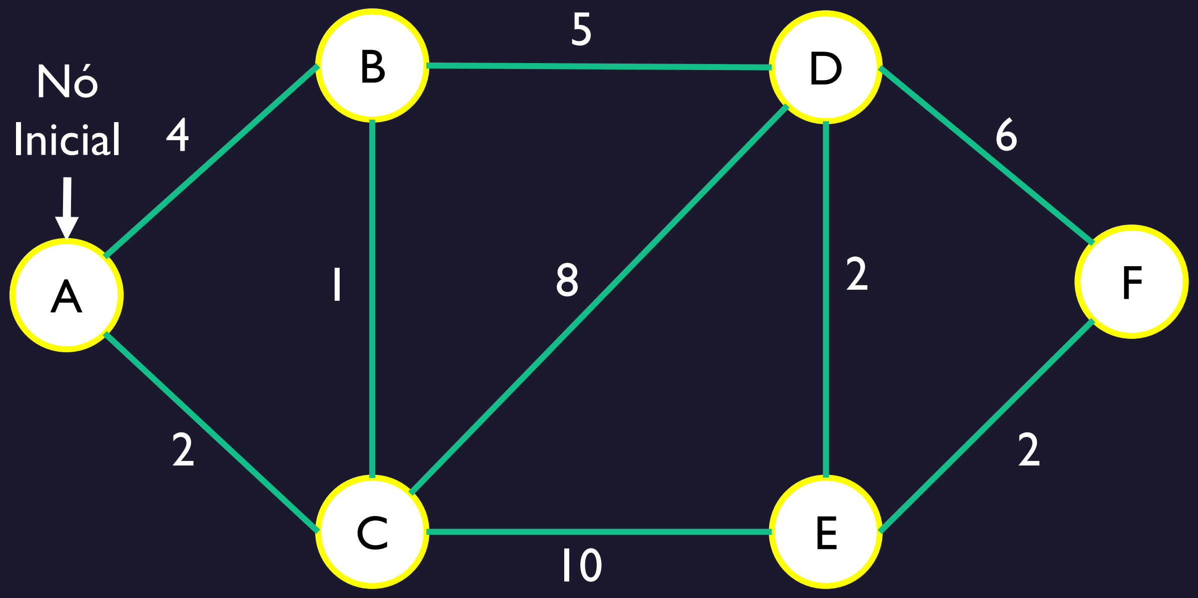
Algoritmo de Dijkstra

Distâncias

A	0
B	∞
C	∞
D	∞
E	∞
F	∞

Predecessores

A	-1
B	-1
C	-1
D	-1
E	-1
F	-1



S = A (nó inicial)

Q = {A, B, C, D, E, F}

Inicialização

Algoritmo de Dijkstra

1. u = A (nó com a menor distância no conjunto Q)

2. Q = {B, C, D, E, F} (removeu do conjunto)

3. Adjacentes do u {B, C}

Distâncias

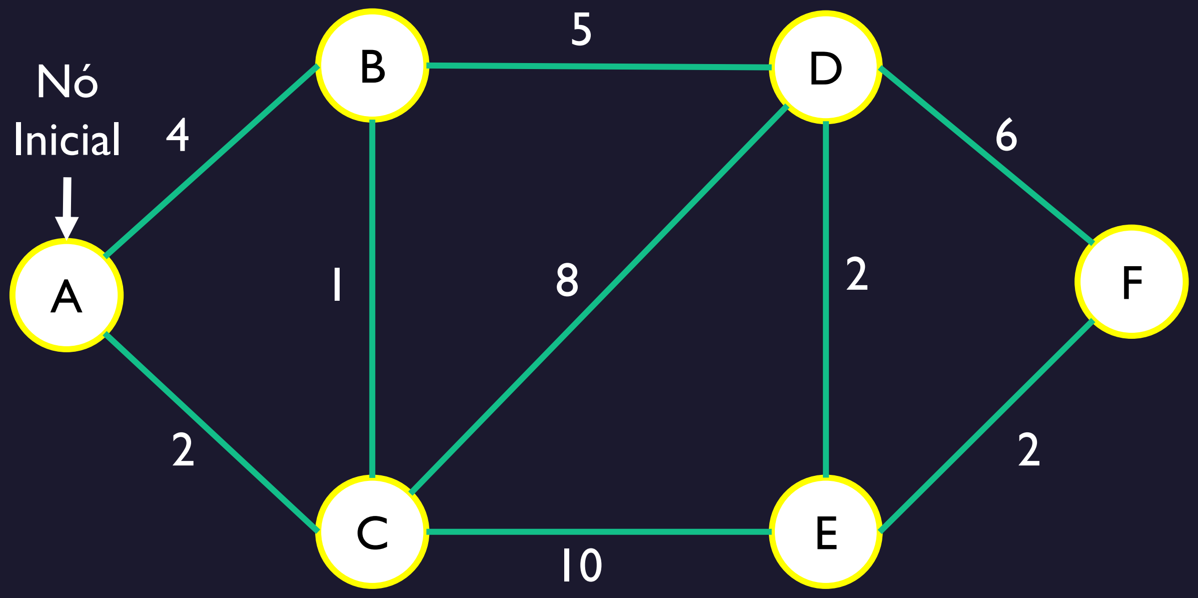
A	0
B	4
C	2
D	∞
E	∞
F	∞

3.1 – Relaxa B
Pred[B] ← A

3.2 – Relaxa C
Pred[C] ← A

Predecessores

A	-1
B	A
C	A
D	-1
E	-1
F	-1



Iteração

Algoritmo de Dijkstra

1. u = C (nó com a menor distância no conjunto Q)

2. Q = {B, D, E, F} (removeu do conjunto)

3. Adjacentes do u {B, D, E}

Distâncias

A	0
B	3
C	2
D	10
E	12
F	∞

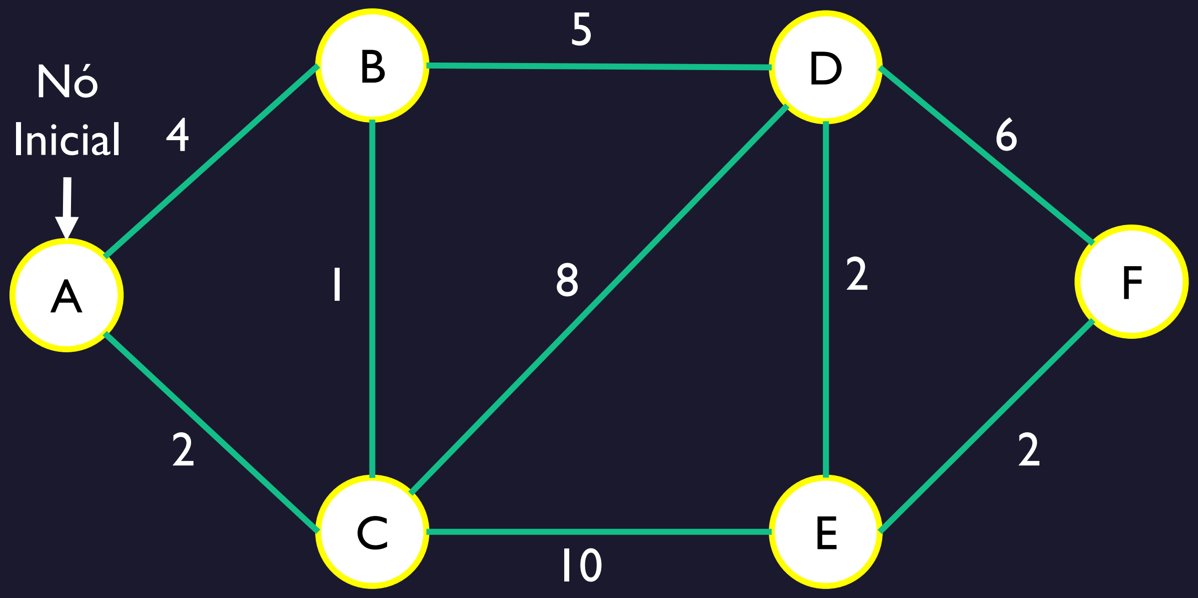
3.1 – Relaxa B
Pred[B] ← C

3.2 – Relaxa D
Pred[D] ← C

3.3 – Relaxa E
Pred[E] ← C

Predecessores

A	-1
B	C
C	A
D	C
E	C
F	-1



Iteração

Algoritmo de Dijkstra

1. u = B (nó com a menor distância no conjunto Q)

2. Q = {D, E, F} (removeu do conjunto)

3. Adjacentes do u {D}

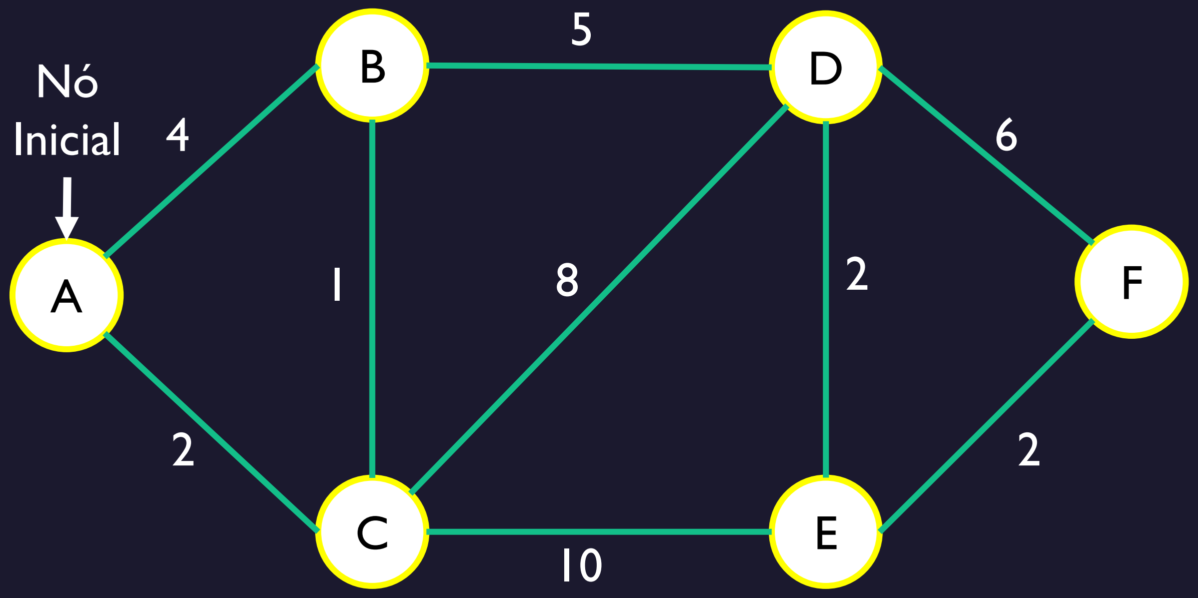
Distâncias

A	0
B	3
C	2
D	8
E	12
F	∞

3.1 – Relaxa D
Pred[D] ← B

Predecessores

A	-1
B	C
C	A
D	B
E	C
F	-1



Iteração

Algoritmo de Dijkstra

1. u = D (nó com a menor distância no conjunto Q)

2. Q = {E, F} (removeu do conjunto)

3. Adjacentes do u {E, F}

Distâncias

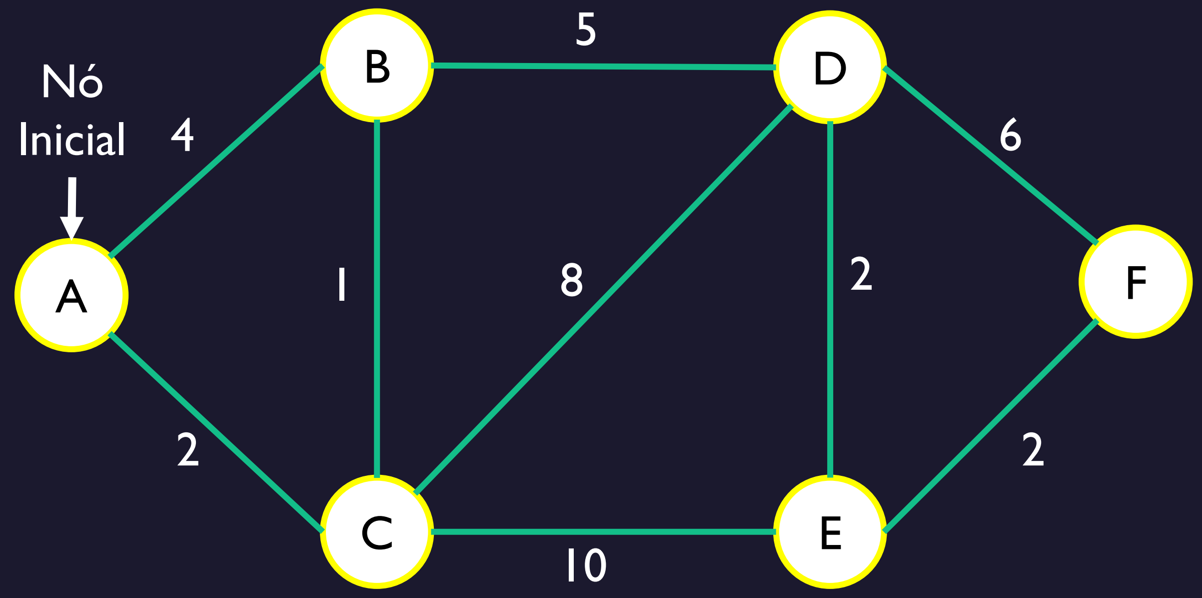
A	0
B	3
C	2
D	8
E	10
F	14

3.1 – Relaxa E
Pred[E] D

3.2 – Relaxa F
Pred[F] ← D

Predecessores

A	-1
B	C
C	A
D	B
E	D
F	D



Iteração

Algoritmo de Dijkstra

1. u = E (nó com a menor distância no conjunto Q)

2. Q = {F} (removeu do conjunto)

3. Adjacentes do u {F}

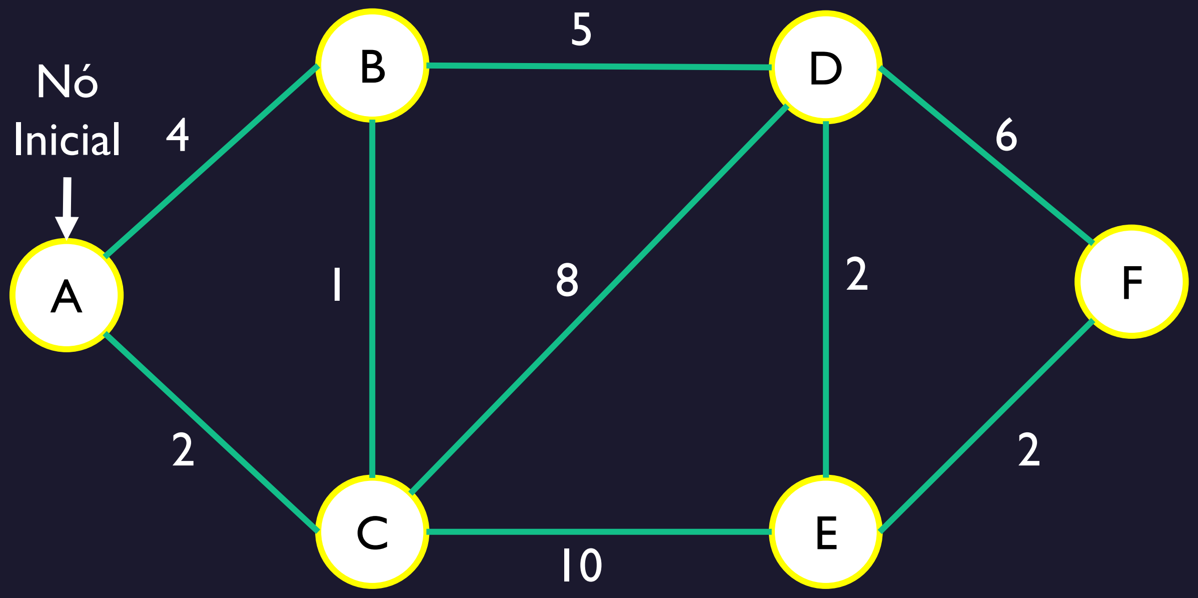
Distâncias

A	0
B	3
C	2
D	8
E	10
F	12

3.1 – Relaxa F
Pred[F] $\leftarrow$ E

Predecessores

A	-1
B	C
C	A
D	B
E	D
F	E



Iteração

Algoritmo de Dijkstra

1. u = F (nó com a menor distância no conjunto Q)

2. Q = {} (removeu do conjunto)

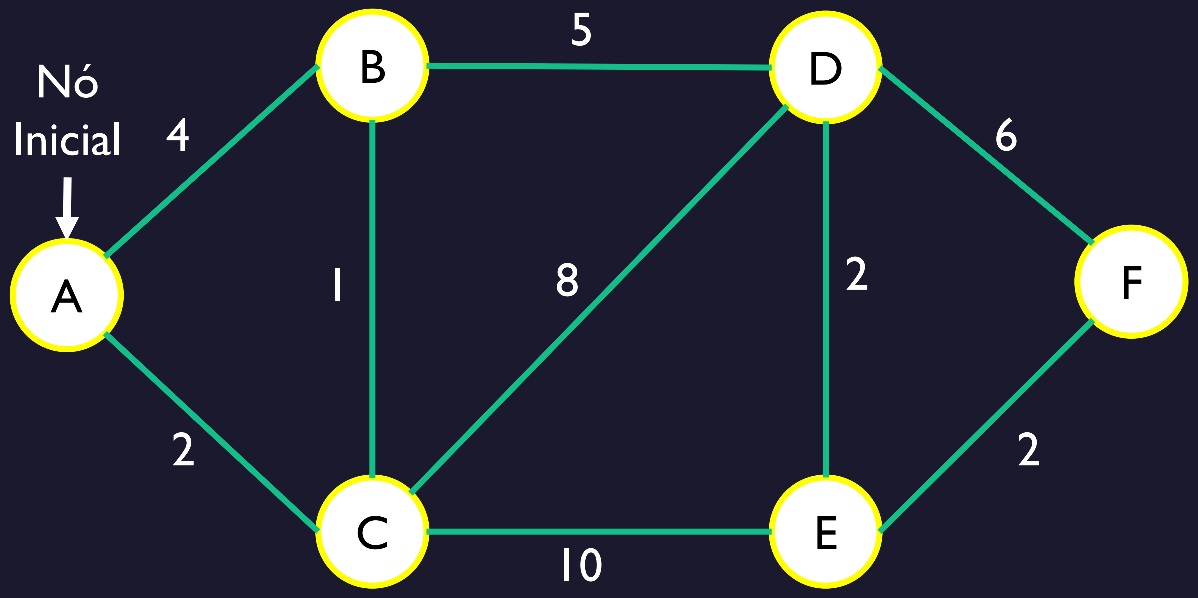
3. Adjacentes do u {}

Distâncias

A	0
B	3
C	2
D	8
E	10
F	12

Predecessores

A	-1
B	C
C	A
D	B
E	D
F	E



FIM

Algoritmo de Dijkstra

- O vetor de distâncias guarda agora o valor dos menores caminhos saindo de A para os demais vértices do grafo.
- Para traçar esse caminho, usamos o vetor de predecessores

Algoritmo de Dijkstra

Distâncias

A	0
B	3
C	2
D	8
E	10
F	12

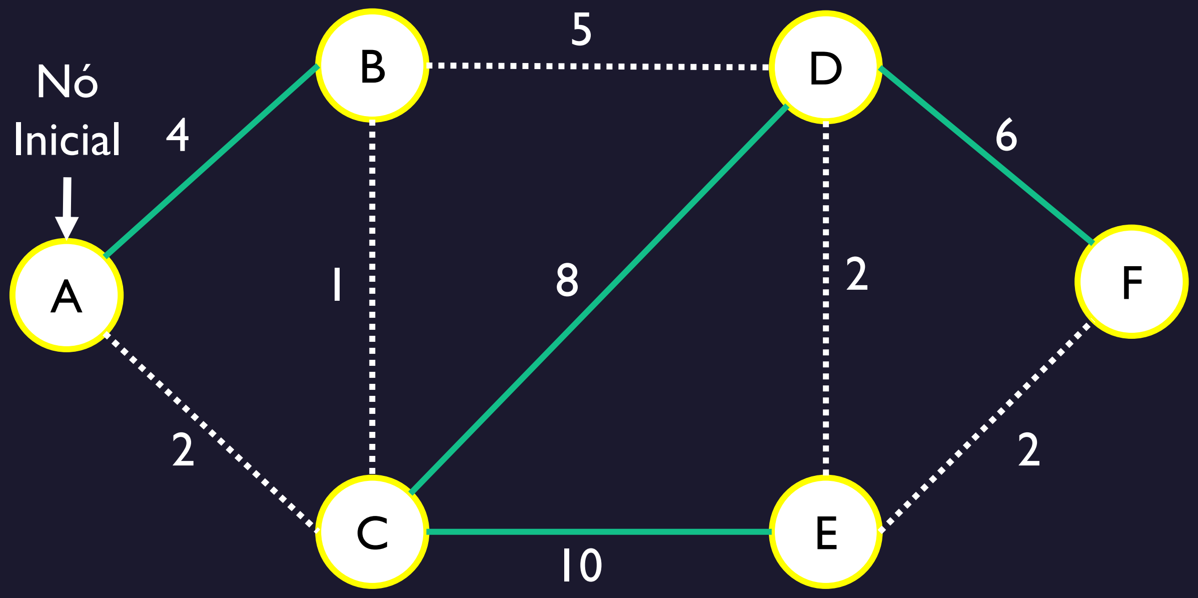
Por exemplo,
O menor caminho de A até F tem valor 12

Predecessores

A	-1
B	C
C	A
D	B
E	D
F	E

Observamos o vetor de predecessores a partir do destino até o vértice inicial:

- Para chegar em F devo passar pelo E
- Para chegar em E devo passar pelo D
- Para chegar em D devo passar pelo B
- Para chegar em B devo passar pelo C
- Para chegar em C devo passar pelo A



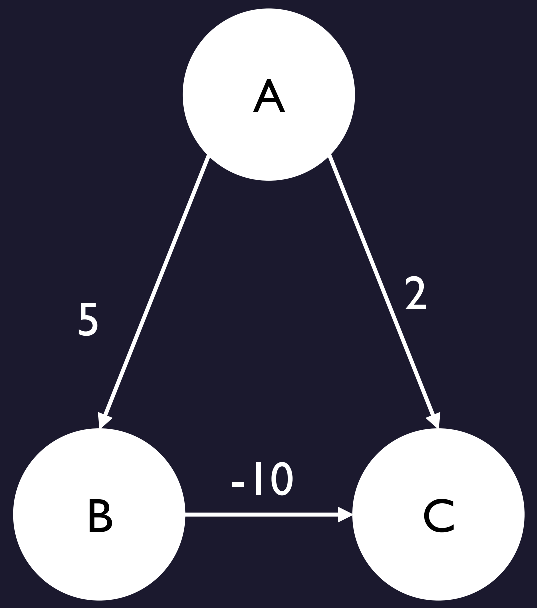
O caminho é: $A \to C \to B \to D \to E \to F$

Limitação do Algoritmo de Dijkstra

- Note que o algoritmo de Dijkstra utiliza uma abordagem gulosa
  - A cada iteração, esse algoritmo inclui a melhor solução local (inclui o vértice que implicará em menor incremento no custo do caminho) – é o que ele faz quando executa "extrair_min"
- **Nem sempre** a abordagem gulosa produz resultados corretos
- No caso do algoritmo de Dijkstra, a abordagem gulosa funciona bem quanto os pesos são todos não-negativos

Limitação do Algoritmo de Dijkstra

- Veja esse exemplo, para calcular o menor caminho a partir de A
  - Na primeira iteração retiramos o A
  - Na segunda iteração teremos que o caminho até C tem peso 2
  - Na terceira iteração teremos que o caminho até B tem peso 5



- Isso é incorreto, pois o menor caminho até C tem peso -5

Limitação do Algoritmo de Dijkstra

- Mas... Em que contexto poderemos mapear uma situação real para um peso negativo?
  - Se o grafo for usado para calcular o custo de atravessar determinado caminho (em um jogo, por exemplo), uma passagem vantajosa teria custo associado negativo.
  - Algumas reações químicas podem ser endotérmicas (que consomem energia) ou exotérmicas, que liberam energia. Essa representação seria com valores positivos e negativos, respectivamente

Algoritmo de Bellman-Ford

- O algoritmo de Bellman-Ford resolve o problema de Caminho Mínimo quando o grafo apresenta arestas de peso negativo
- Algumas ponderações:
  - Em um grafo qualquer, o caminho mínimo não pode possuir mais do que |V| -1 arestas
  - Se o número de arestas for maior que |V|-1, teríamos um ciclo.
    - Se esse ciclo tiver peso total = 0, então ele não precisa fazer parte do caminho mínimo
    - Se o ciclo tiver peso total > 0, ele não poderia fazer parte do caminho mais curto – nosso cálculo estaria errado
    - Se o ciclo tiver peso <0, então não é possível estabelecer um caminho mais curto nesse grafo (cada volta no ciclo geraria um caminho mais curto ainda)

Algoritmo de Bellman-Ford

A ideia desse algoritmo é usar uma técnica de Programação Dinâmica (Dynamic Programming)

Basicamente, calculamos os caminhos mais curtos de maneira ascendente.

Primeiro, calculamos os caminhos mais curtos que têm no máximo **uma aresta**

Em seguida, calculamos os caminhos mais curtos com no máximo 2 arestas, e assim por diante.

Após a i-ésima iteração, os caminhos mais curtos com no máximo i arestas são calculados.

Uma vez que sabemos que existem no máximo de |V| - 1 arestas em qualquer caminho simples, iteramos |V| - 1 vezes.

Algoritmo de Bellman-Ford

```
Bellman-Ford(Grafo, W, Origem)          //W guarda todos os pesos das arestas do grafo
    Initialize-Single-Source(G, S)      //Inicializa o peso do vértice origem como 0
    para i = 1  |G.V| - 1
        Para cada  aresta(u, v) ∈ G.E
            Relaxar(u, v, W)

    Para cada aresta (u, v) ∈ G.E
        se distância[v] > distância[u] + peso(u, v)
            return False
    return True
```

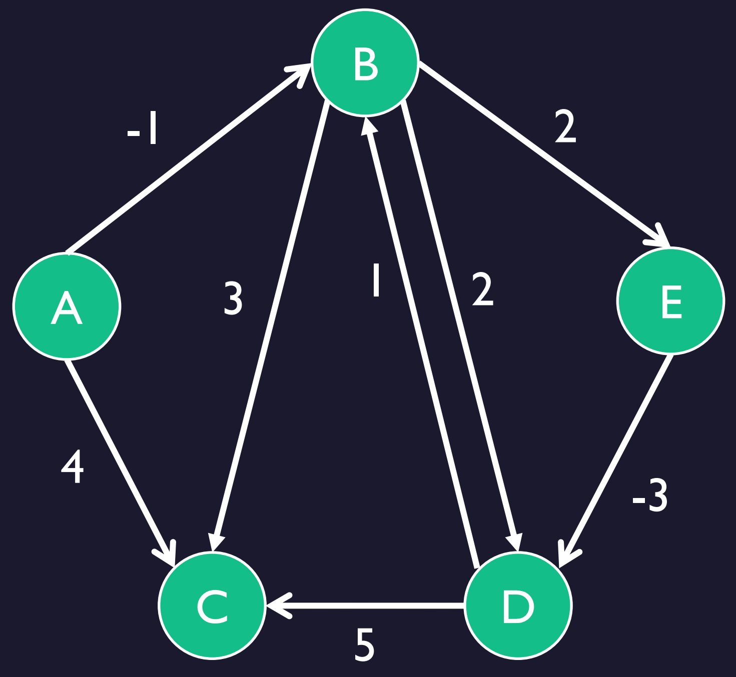
Algoritmo de Bellman-Ford

- Observe que, diferente do algoritmo de Dijkstra, o algoritmo de Bellman Ford executa a operação de 'relaxar' em todas as arestas do grafo

```
Relaxar (u,v, W):
    se Distancia[v] > Distancia[u]+ W(u, v):
        Distancia[v] = Distancia[u] + W(u, v)
        Predecessor[v] = u
```

- Ao iterar V-1 vezes, garantimos que marcamos as V-1 arestas que estão no caminho mais curto.
- Se após essas V-1 passagens ainda for possível relaxar alguma aresta, então significa que existe 1 ciclo negativo no grafo (linha 6-8)

Exemplo Bellman-Ford



Obter o Caminho Mínimo a partir do vértice A

Distâncias

A	B	C	D	E
0	∞	∞	∞	∞

Predecessores

A	B	C	D	E
-1	-1	-1	-1	-1

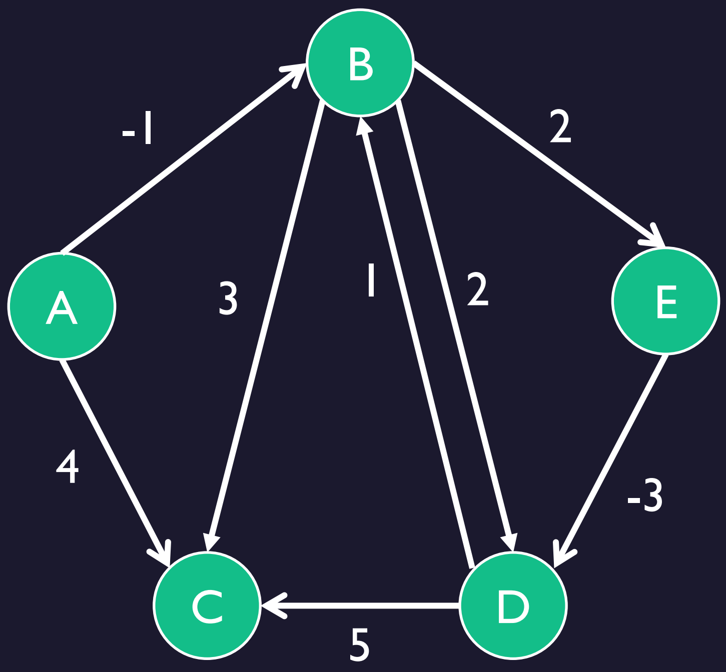
Nesse exemplo, as arestas serão avaliadas nessa ordem:

1. (B,E)
2. (D,B)
3. (B,D)
4. (A,B)
5. (A,C)
6. (D,C)
7. (B,C)
8. (E,D)

Inicialização

Exemplo Bellman-Ford

Obter o Caminho Mínimo a partir do vértice A



Distâncias

A	B	C	D	E
0	∞	∞	∞	∞

Predecessores

A	B	C	D	E
-1	-1	-1	-1	-1

Nesse exemplo, as arestas serão avaliadas nessa ordem:

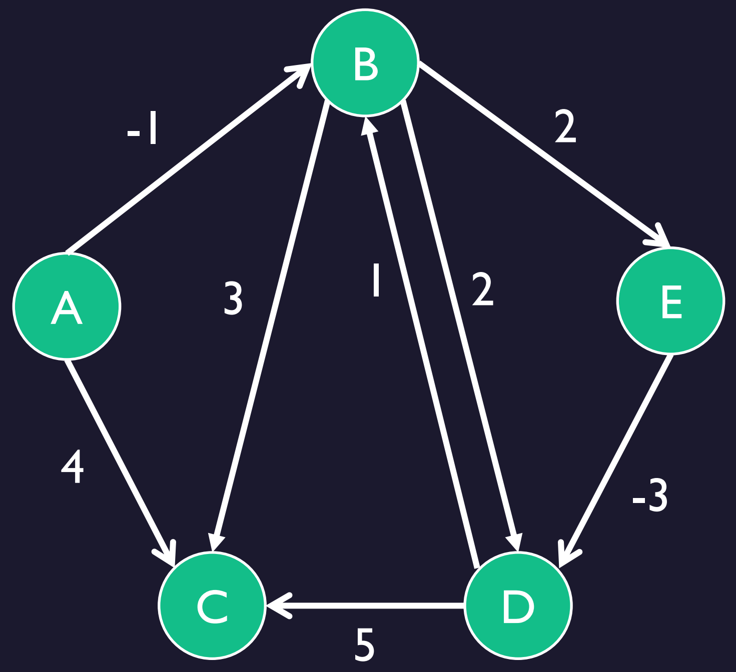
1. (B,E)
2. (D,B)
3. (B,D)
4. (A,B)
5. (A,C)
6. (D,C)
7. (B,C)
8. (E,D)

```
Relaxar (B,E, W):
    se Distancia[E] > Distancia[B]+ W(B, E):
        Distancia[E] = Distancia[B] + W(B, E)
        Predecessor[E] = B
```

Nada ocorre

1ª Iteração - Processando (B, E)

Exemplo Bellman-Ford



Obter o Caminho Mínimo a partir do vértice A

Distâncias

A	B	C	D	E
0	∞	∞	∞	∞

Predecessores

A	B	C	D	E
-1	-1	-1	-1	-1

Nesse exemplo, as arestas serão avaliadas nessa ordem:

1. (B,E)
2. (D,B)
3. (B,D)
4. (A,B)
5. (A,C)
6. (D,C)
7. (B,C)
8. (E,D)

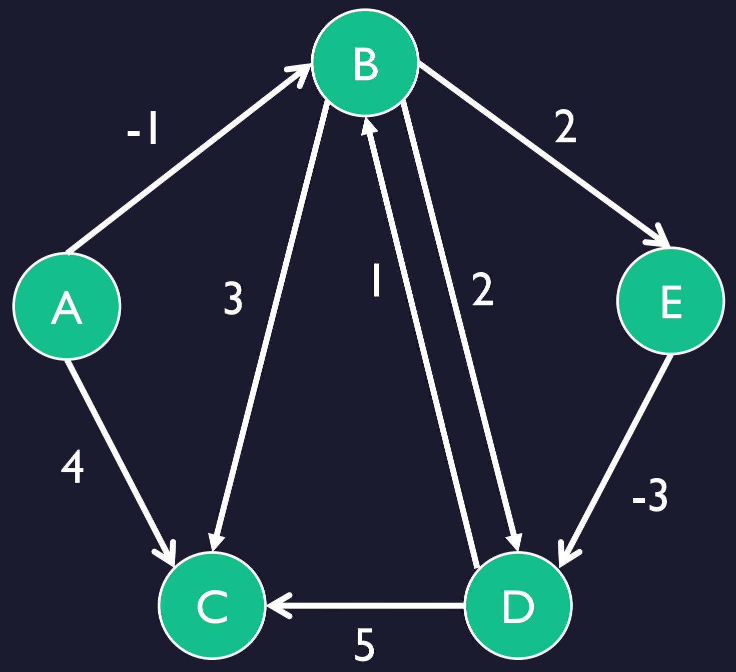
```
Relaxar (D,B, W):
    se Distancia[B] > Distancia[D]+ W(D, B):
        Distancia[B] = Distancia[D] + W(D, B)
        Predecessor[B] = D
```

Nada ocorre

1ª Iteração - Processando (D, B)

Exemplo Bellman-Ford

Obter o Caminho Mínimo a partir do vértice A



Distâncias

A	B	C	D	E
0	∞	∞	∞	∞

Predecessores

A	B	C	D	E
-1	-1	-1	-1	-1

Nesse exemplo, as arestas serão avaliadas nessa ordem:

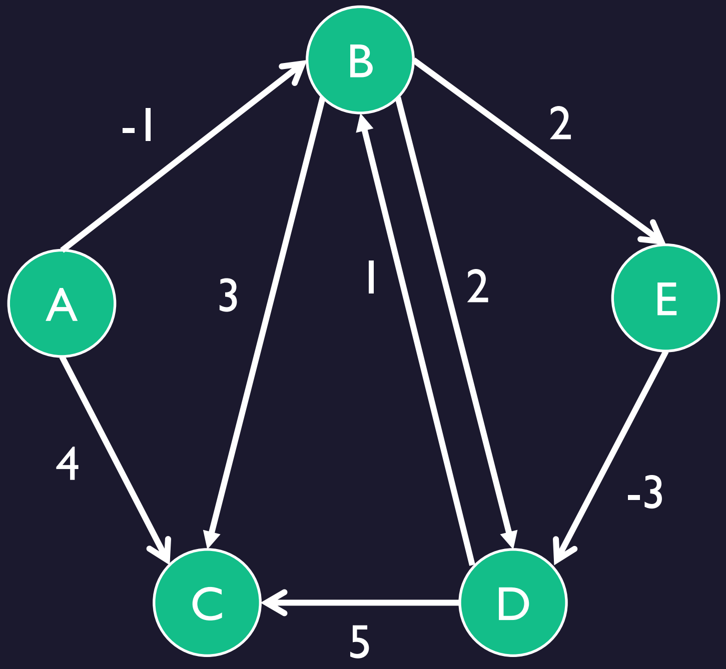
1. (B,E)
2. (D,B)
3. (B,D)
4. (A,B)
5. (A,C)
6. (D,C)
7. (B,C)
8. (E,D)

```
Relaxar (B,D, W):
    se Distancia[D] > Distancia[B]+ W(B, D):
        Distancia[D] = Distancia[B] + W(B, D)
        Predecessor[D] = B
```

Nada ocorre

1ª Iteração - Processando (B, D)

Exemplo Bellman-Ford



Obter o Caminho Mínimo a partir do vértice A

Distâncias

A	B	C	D	E
0	-1	∞	∞	∞

Predecessores

A	B	C	D	E
-1	A	-1	-1	-1

Nesse exemplo, as arestas serão avaliadas nessa ordem:

1. (B,E)
2. (D,B)
3. (B,D)
4. (A,B)
5. (A,C)
6. (D,C)
7. (B,C)
8. (E,D)

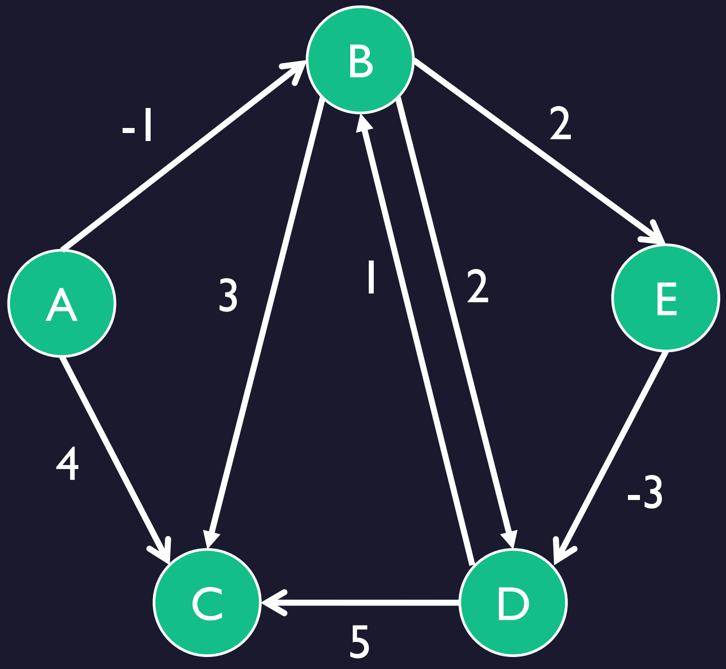
```
Relaxar (A,B, W):
    se Distancia[B] > Distancia[A]+ W(A, B):
        Distancia[B] = Distancia[A] + W(A, B)
        Predecessor[B] = A
```

Atualiza distâncias!
Atualiza Predecessores!

1ª Iteração - Processando (A, B)

Exemplo Bellman-Ford

Obter o Caminho Mínimo a partir do vértice A



Distâncias

A	B	C	D	E
0	-1	4	∞	∞

Predecessores

A	B	C	D	E
-1	A	A	-1	-1

Nesse exemplo, as arestas serão avaliadas nessa ordem:

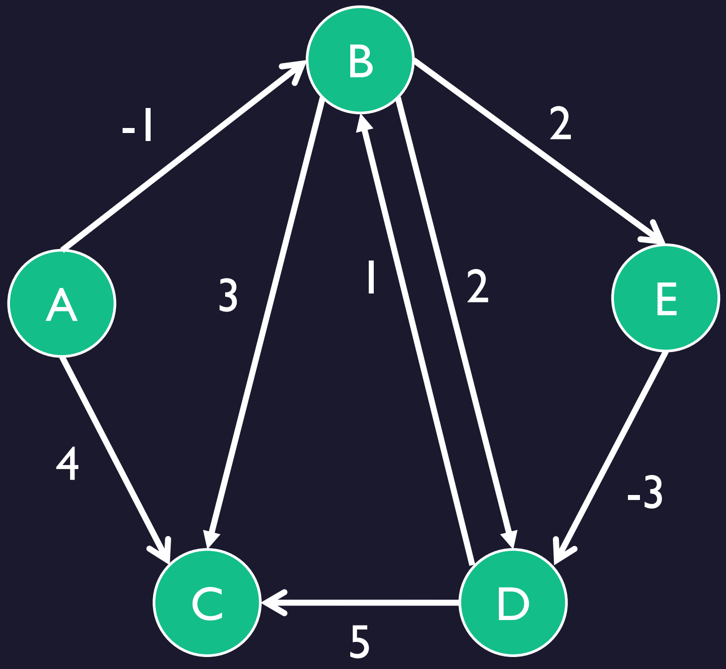
1. (B,E)
2. (D,B)
3. (B,D)
4. (A,B)
5. (A,C)
6. (D,C)
7. (B,C)
8. (E,D)

```
Relaxar (A,C, W):
    se Distancia[C] > Distancia[A]+ W(A, C):
        Distancia[C] = Distancia[A] + W(A, C)
        Predecessor[C] = A
```

Atualiza distâncias!
Atualiza Predecessores!

1ª Iteração - Processando (A, C)

Exemplo Bellman-Ford



Obter o Caminho Mínimo a partir do vértice A

Distâncias

A	B	C	D	E
0	-1	4	∞	∞

Predecessores

A	B	C	D	E
-1	A	A	-1	-1

Nesse exemplo, as arestas serão avaliadas nessa ordem:

1. (B,E)
2. (D,B)
3. (B,D)
4. (A,B)
5. (A,C)
6. (D,C)
7. (B,C)
8. (E,D)

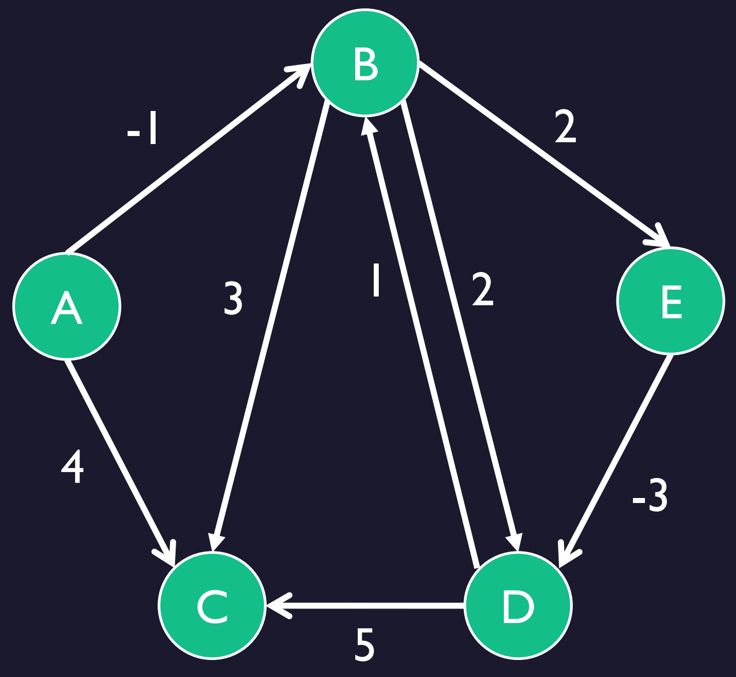
```
Relaxar (D,C, W):
    se Distancia[C] > Distancia[D]+ W(D, C):
        Distancia[C] = Distancia[D] + W(D, C)
        Predecessor[C] = D
```

Nada ocorre

1ª Iteração - Processando (D, C)

Exemplo Bellman-Ford

Obter o Caminho Mínimo a partir do vértice A



Distâncias

A	B	C	D	E
0	-1	2	∞	∞

Predecessores

A	B	C	D	E
-1	A	B	-1	-1

Nesse exemplo, as arestas serão avaliadas nessa ordem:

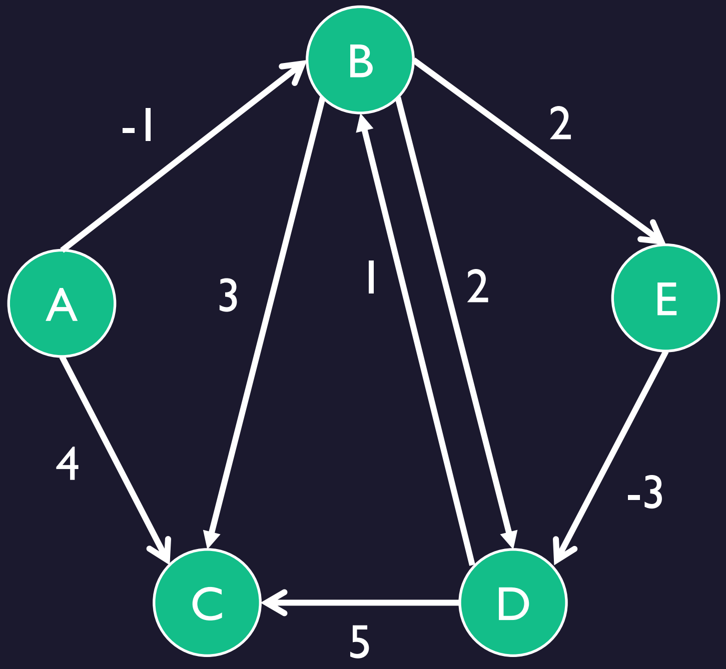
1. (B,E)
2. (D,B)
3. (B,D)
4. (A,B)
5. (A,C)
6. (D,C)
7. (B,C)
8. (E,D)

```
Relaxar (B,C, W):
    se Distancia[C] > Distancia[B]+ W(B, C):
        Distancia[C] = Distancia[B] + W(B, C)
        Predecessor[C] = B
```

Atualiza distâncias!
Atualiza Predecessores!

1ª Iteração - Processando (B, C)

Exemplo Bellman-Ford



Obter o Caminho Mínimo a partir do vértice A

Distâncias

A	B	C	D	E
0	-1	2	∞	∞

Predecessores

A	B	C	D	E
-1	A	B	-1	-1

Nesse exemplo, as arestas serão avaliadas nessa ordem:

1. (B,E)
2. (D,B)
3. (B,D)
4. (A,B)
5. (A,C)
6. (D,C)
7. (B,C)
8. (E,D)

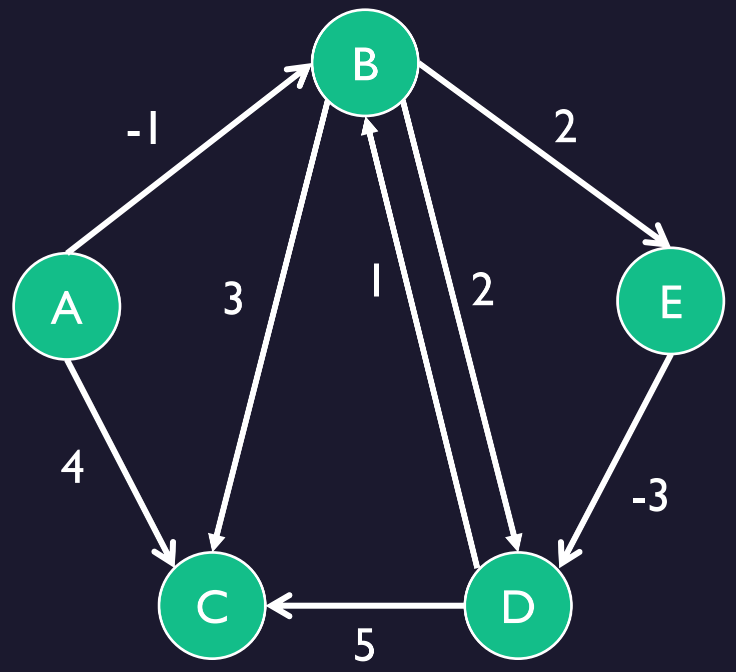
```
Relaxar (E,D, W):
    se Distancia[D] > Distancia[E]+ W(E, D):
        Distancia[D] = Distancia[E] + W(E, D)
        Predecessor[D] = E
```

Nada ocorre

1ª Iteração - Processando (E, D)

Exemplo Bellman-Ford

Obter o Caminho Mínimo a partir do vértice A



Distâncias

A	B	C	D	E
0	-1	2	∞	1

Predecessores

A	B	C	D	E
-1	A	B	-1	B

Nesse exemplo, as arestas serão avaliadas nessa ordem:

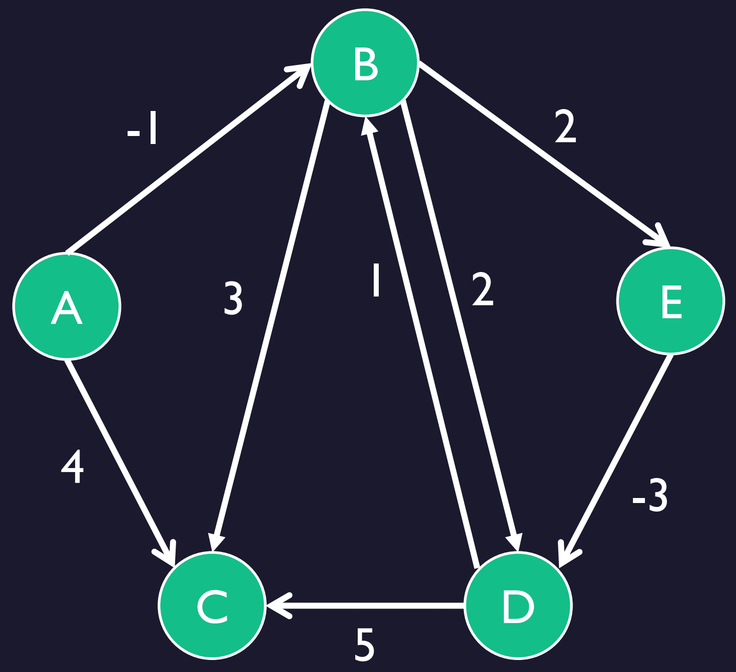
1. (B,E)
2. (D,B)
3. (B,D)
4. (A,B)
5. (A,C)
6. (D,C)
7. (B,C)
8. (E,D)

```
Relaxar (B,E, W):
    se Distancia[E] > Distancia[B]+ W(B, E):
        Distancia[E] = Distancia[B] + W(B, E)
        Predecessor[E] = B
```

Atualiza distâncias!
Atualiza Predecessores!

2ª Iteração - Processando (B, E)

Exemplo Bellman-Ford



Obter o Caminho Mínimo a partir do vértice A

Distâncias

A	B	C	D	E
0	-1	2	∞	1

Predecessores

A	B	C	D	E
-1	A	B	-1	B

Nesse exemplo, as arestas serão avaliadas nessa ordem:

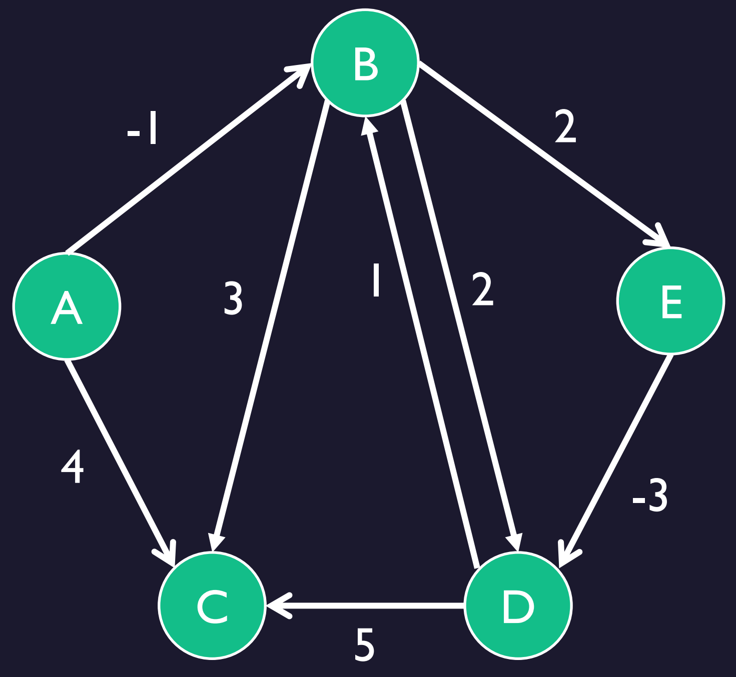
1. (B,E)
2. (D,B)
3. (B,D)
4. (A,B)
5. (A,C)
6. (D,C)
7. (B,C)
8. (E,D)

```
Relaxar (D,B, W):
    se Distancia[B] > Distancia[D]+ W(D, B):
        Distancia[B] = Distancia[D] + W(D, B)
        Predecessor[B] = D
```

Nada ocorre

2ª Iteração - Processando (D, B)

Exemplo Bellman-Ford



Obter o Caminho Mínimo a partir do vértice A

Distâncias

A	B	C	D	E
0	-1	2	1	1

Predecessores

A	B	C	D	E
-1	A	B	B	B

Nesse exemplo, as arestas serão avaliadas nessa ordem:

1. (B,E)
2. (D,B)
3. (B,D)
4. (A,B)
5. (A,C)
6. (D,C)
7. (B,C)
8. (E,D)

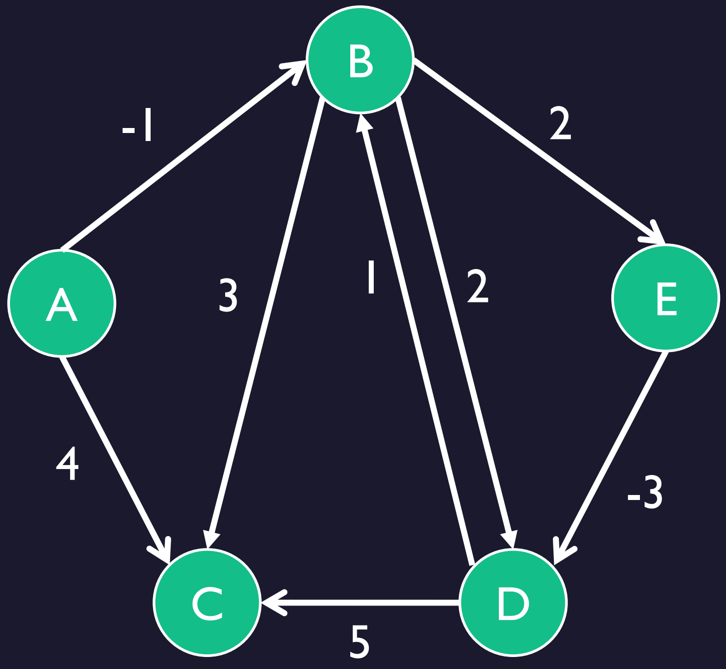
```
Relaxar (B,D, W):
    se Distancia[D] > Distancia[B]+ W(B, D):
        Distancia[D] = Distancia[B] + W(B, D)
        Predecessor[D] = B
```

Atualiza distâncias!
Atualiza Predecessores!

2ª Iteração - Processando (B, D)

Exemplo Bellman-Ford

Obter o Caminho Mínimo a partir do vértice A



Distâncias

A	B	C	D	E
0	-1	2	1	1

Predecessores

A	B	C	D	E
-1	A	B	B	B

Nesse exemplo, as arestas serão avaliadas nessa ordem:

1. (B,E)
2. (D,B)
3. (B,D)
4. (A,B)
5. (A,C)
6. (D,C)
7. (B,C)
8. (E,D)

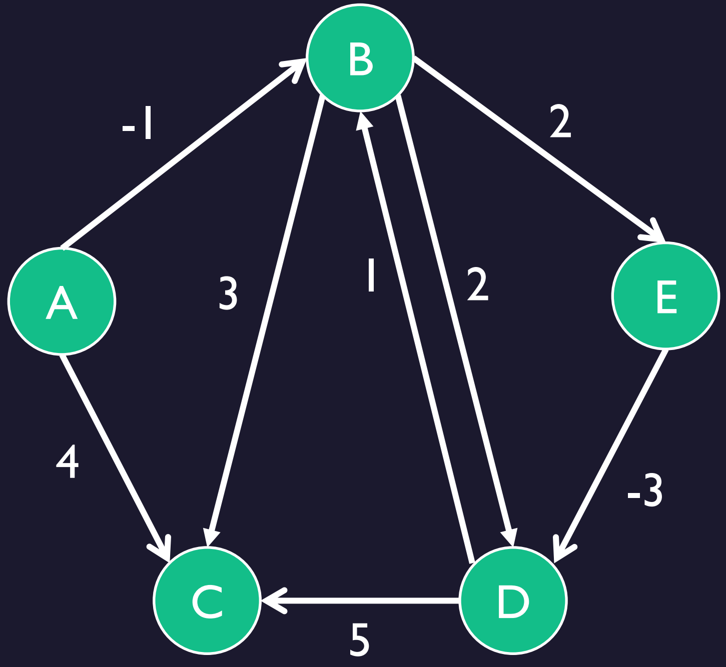
```
Relaxar (A,B, W):
    se Distancia[B] > Distancia[A]+ W(A, B):
        Distancia[B] = Distancia[A] + W(A, B)
        Predecessor[B] = A
```

Nada ocorre

2ª Iteração - Processando (A, B)

Exemplo Bellman-Ford

Obter o Caminho Mínimo a partir do vértice A



Distâncias

A	B	C	D	E
0	-1	2	1	1

Predecessores

A	B	C	D	E
-1	A	B	B	B

Nesse exemplo, as arestas serão avaliadas nessa ordem:

1. (B,E)
2. (D,B)
3. (B,D)
4. (A,B)
5. (A,C)
6. (D,C)
7. (B,C)
8. (E,D)

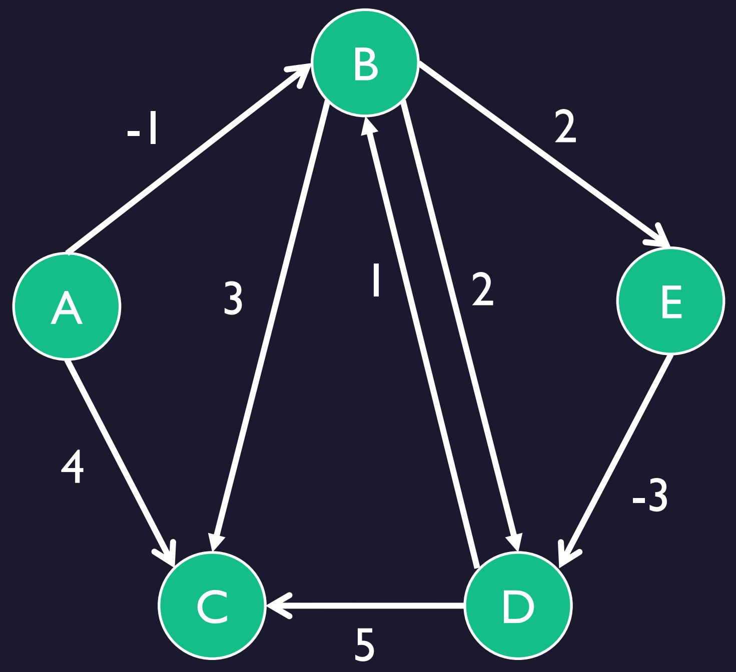
```
Relaxar (A,C, W):
    se Distancia[C] > Distancia[A]+ W(A, C):
        Distancia[C] = Distancia[A] + W(A, C)
        Predecessor[C] = A
```

Nada ocorre

2ª Iteração - Processando (A, C)

Exemplo Bellman-Ford

Obter o Caminho Mínimo a partir do vértice A



Distâncias

A	B	C	D	E
0	-1	2	1	1

Predecessores

A	B	C	D	E
-1	A	B	B	B

Nesse exemplo, as arestas serão avaliadas nessa ordem:

1. (B,E)
2. (D,B)
3. (B,D)
4. (A,B)
5. (A,C)
6. (D,C)
7. (B,C)
8. (E,D)

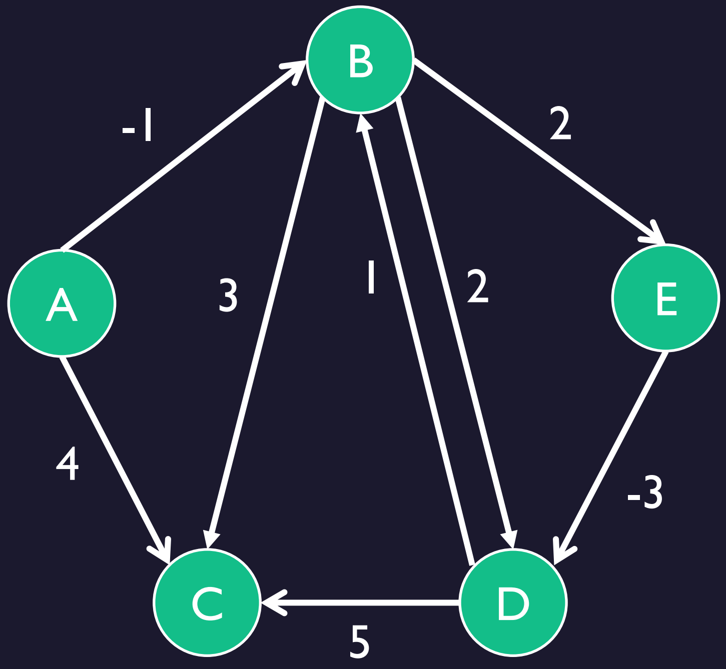
```
Relaxar (D,C, W):
    se Distancia[C] > Distancia[D]+ W(D, C):
        Distancia[C] = Distancia[D] + W(D, C)
        Predecessor[C] = D
```

Nada ocorre

2ª Iteração - Processando (D, C)

Exemplo Bellman-Ford

Obter o Caminho Mínimo a partir do vértice A



Distâncias

A	B	C	D	E
0	-1	2	1	1

Predecessores

A	B	C	D	E
-1	A	B	B	B

Nesse exemplo, as arestas serão avaliadas nessa ordem:

1. (B,E)
2. (D,B)
3. (B,D)
4. (A,B)
5. (A,C)
6. (D,C)
7. (B,C)
8. (E,D)

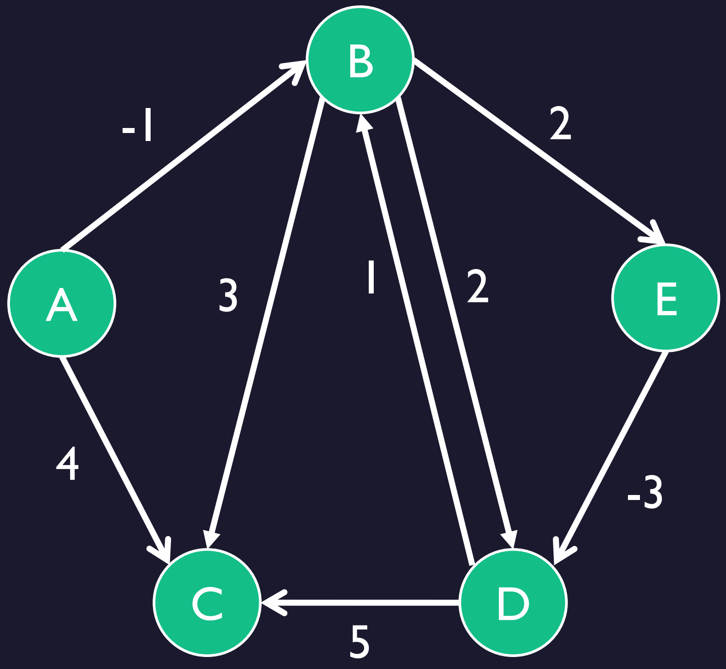
```
Relaxar (B,C, W):
    se Distancia[C] > Distancia[B]+ W(B, C):
        Distancia[C] = Distancia[B] + W(B, C)
        Predecessor[C] = B
```

Nada ocorre

2ª Iteração - Processando (B, C)

Exemplo Bellman-Ford

Obter o Caminho Mínimo a partir do vértice A



Distâncias

A	B	C	D	E
0	-1	2	-2	1

Predecessores

A	B	C	D	E
-1	A	B	E	B

Nesse exemplo, as arestas serão avaliadas nessa ordem:

1. (B,E)
2. (D,B)
3. (B,D)
4. (A,B)
5. (A,C)
6. (D,C)
7. (B,C)
8. (E,D)

```
Relaxar (E,D, W):
    se Distancia[D] > Distancia[E]+ W(E, D):
        Distancia[D] = Distancia[E] + W(E, D)
        Predecessor[D] = E
```

Atualiza distâncias!
Atualiza Predecessores!

2ª Iteração - Processando (E, D)

Exemplo Bellman-Ford

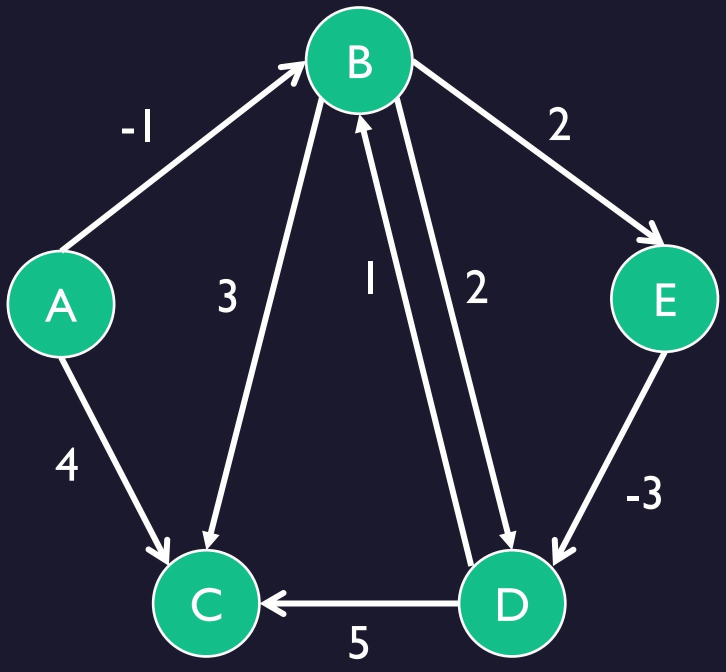
- Nas iterações 3 e 4 não ocorrem mudanças. As distâncias já estão minimizadas
- Após as (V-1) iterações, temos as distâncias mínimas e a sequência de predecessores no caminho mínimo a partir de A

Distâncias

A	B	C	D	E
0	-1	2	-2	1

Predecessores

A	B	C	D	E
-1	A	B	E	B



Comparação entre os Algoritmos

DIJKSTRA

- Complexidade – depende do método para escolher o menor valor – pode chegar a O (E Log V)
- Abordagem Gulosa
- Não funciona em grafos com arestas de peso negativo
- Simulador: https://www-m9.ma.tum.de/graph-algorithms/spp-dijkstra/index_en.html

- BELLMAN-FORD

- Complexidade O (V.E)
- Programação Dinâmica
- Funciona em grafos com arestas de peso negativo
- Simulador: https://www-m9.ma.tum.de/graph-algorithms/spp-bellman-ford/index_en.html

Questão para Pensar:

- E se quisermos encontrar o caminho mínimo para um **único destino?**
- Em um exemplo, haverá uma reunião familiar em uma determinada cidade, com familiares vindo de diversos lugares, utilizando diversos meios (avião, carro, ônibus).
- Desejamos calcular, para todos, a rota mais barata possível.

Solução

- Esse problema pode ser facilmente convertido em um problema de Caminho Mínimo de Fonte Única.

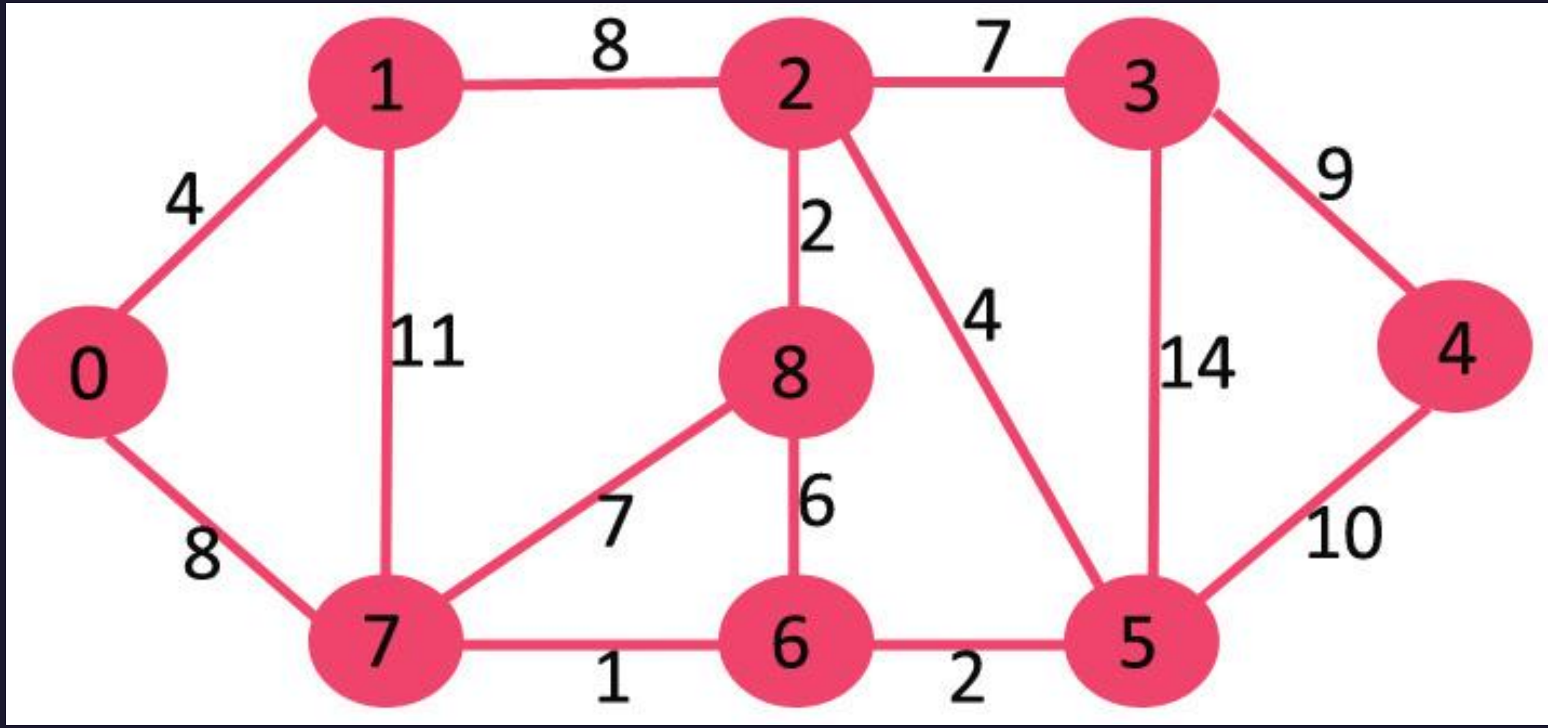
- Se o grafo for direcionado:

  Inverta o sentido de todas as arestas e calcule o caminho mínimo a partir do vértice desejado

- Se o grafo não for direcionado:
  - Basta calcular o caminho mínimoa partir do vértice desejado

Exercício

Calcule o menor caminho a partir do vértice 0 para todos os demais vértices no grafo da figura

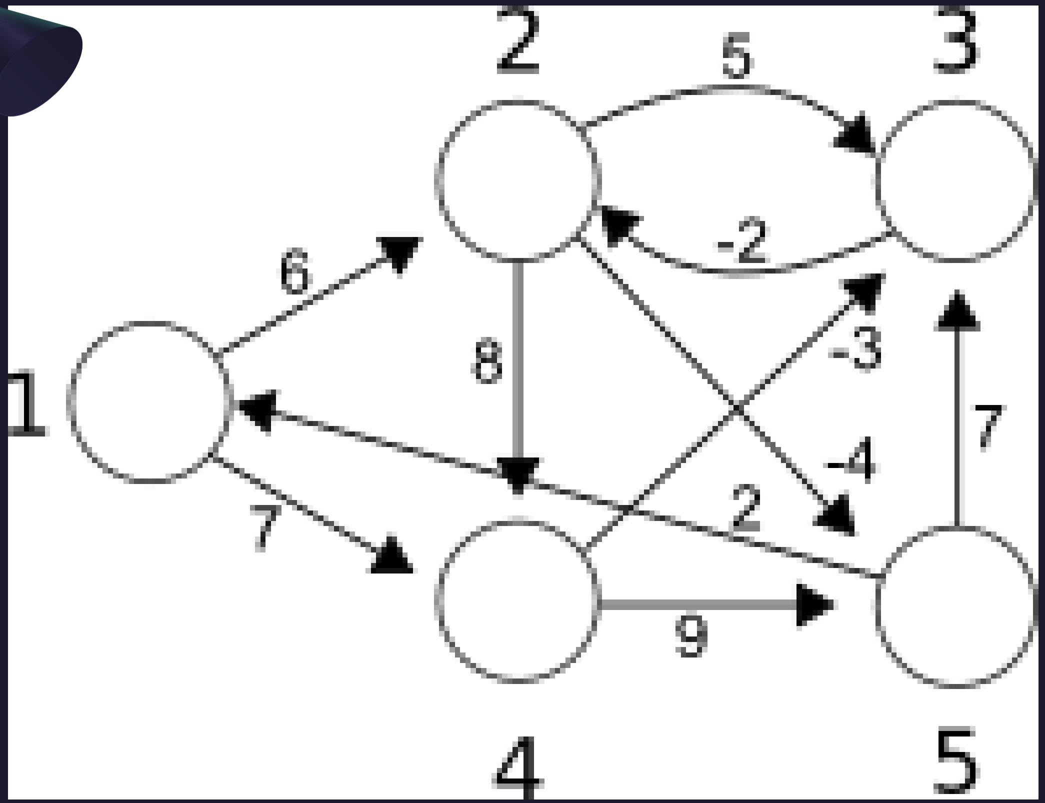


Exercício

- Considere um tabuleiro com 12 casas, cada uma com um número.
- Objetivo é partir do canto superior esquerdo (casa com um 0 – zero) e chegar ao canto inferior direito por meio de movimentos para a esquerda ou para baixo, de forma que o valor da soma das casas percorridas seja o mínimo possível
  - Formule uma solução como um grafo – caminho mínimo
  - Resolva esse problema (do tabuleiro da figura)
  - Proponha uma solução genérica (para qualquer tabuleiro)

0	4	3	9
7	8	9	-1
2	-3	1	8

Exercício para Aprendizagem



- Qual é o valor do menor caminho entre 1 e 2 no grafo da Figura?
- É possível interromper a execução do algoritmo assim que atingimos o vértice destino desejado?
- E se o grafo tivesse apenas pesos não-negativos, seria possível interromper a execução do algoritmo assim que atingimos o vértice desejado?

Referências usadas nesse material

- Dijsktra's algorithm. Disponível em: <https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/>. Acesso em: 2 jul. 2020
- Bellman–Ford Algorithm. Disponível em: <https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23//>. Acesso em: 2 jul. 2020
- CORMEN, Thomas. **Desmistificando algoritmos**. Elsevier Brasil, 2017.