

## Sumários

- Caminhos mais curtos de origem única
  - Propriedades
  - Algoritmo de Dijkstra
  - Algoritmo de Bellman-Ford
  - Caminhos mais curtos em DAGs

Aula 7

## Definição [Grafo Pesoado]

- Um grafo pesoado  $G = (V, E, w)$  é um tripla constituída por:

- um conjunto de vértices  $V$
  - um conjunto de arestas  $E \subseteq V \times V$
  - uma função de peso  $w: E \rightarrow \mathbb{R}$

- Seja  $G = (V, E, w)$  um grafo pesoado, o peso do caminho  $p = \langle v_0, \dots, v_n \rangle$  é dado por:

$$w(p) = \sum_{i=0}^{n-1} w(v_i, v_{i+1})$$

- Seja  $\delta: V \times V \rightarrow \mathbb{R}$  a função que mede cada par de vértices no peso do caminho mais curto que os liga.

$$\delta(u, v) = \min \left\{ w(p) \mid u \xrightarrow{p} v \right\}$$

↳  $v$  é atingível a partir de  $u$  pelo caminho  $p$

## Definição [Problemas de Caminhos Mais Curtos]

- Caminhos mais curtos de origem única

Dado um vértice  $s$  determinar para todo o vértice  $r \in V$   
o caminho  $p$  tal que:  $s \xrightarrow{p} r$  e  $\delta(s, r) = w(p)$ .

- Caminhos mais curtos de origem única e fonte única

Dados dois vértices  $s$  e  $u$ , determinar o caminho  $p$   
tal que:  $s \xrightarrow{p} u$  e  $\delta(s, u) = w(p)$ .

- Caminhos mais curtos entre todos os pares

Para todos os vértices  $u, v \in V$ , determinar o  
caminho  $p$  tal que:  $u \xrightarrow{p} v$  e  $\delta(u, v) = w(p)$ .

### Lema [Caminhos Mais Curtos - Sub-estrutura Óptima]

Seja  $G = (V, E, w)$  um grafo pesado e  $p = \langle v_0, \dots, v_n \rangle$  um caminho mais curto em  $G$ , temos que:

$$\forall 0 \leq i \leq j \leq n. \quad w(\langle v_i, \dots, v_j \rangle) = \delta(v_i, v_j)$$

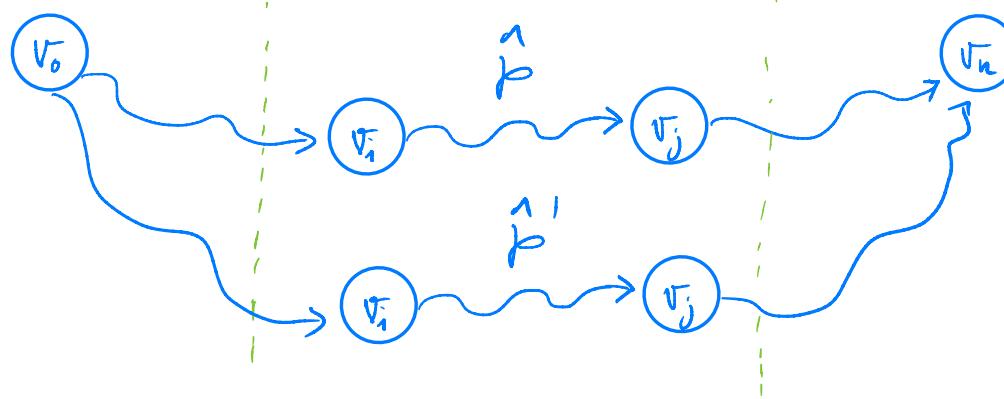
### Lema [Desigualdade Triangular]

$$(u, v) \in E \Rightarrow \delta(s, v) \leq \delta(s, u) + w(u, v)$$

### Lema [Caminhos Mais Curtos - Sub-estrutura Óptima]

Seja  $G = (V, E, w)$  um grafo pesado e  $p = \langle v_0, \dots, v_n \rangle$  um caminho mais curto em  $G$ , temos que:

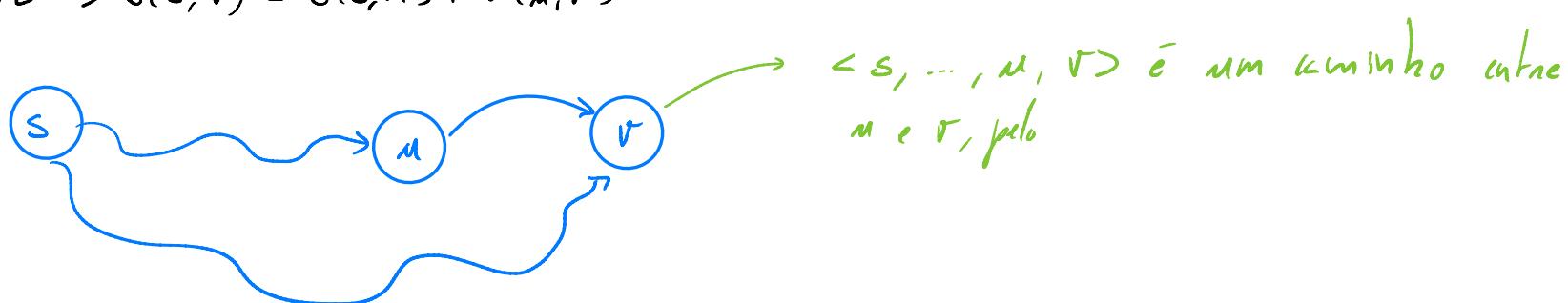
$$\forall 0 \leq i \leq j \leq n. \quad w(\langle v_i, \dots, v_j \rangle) = \delta(v_i, v_j)$$



- Se  $\hat{p}$  fosse mais curto que  $p$ , então o caminho de baixo seria mais curto que o caminho de cima  
contradição

### Lema [Desigualdade de Triangulares]

$$(u, v) \in E \Rightarrow \delta(s, v) \leq \delta(s, u) + w(u, v)$$



## Lema [Caminhos Mais Curtos - Subestrutura Óptima]

Seja  $G = (V, E, w)$  um grafo pesado e  $p = \langle v_0, \dots, v_n \rangle$  um caminho mais curto em  $G$ , temos que:

$$\forall 0 \leq i \leq j \leq n. \quad w(\langle v_i, \dots, v_j \rangle) = \delta(v_i, v_j)$$

### Prova

- Suponhamos, por contradição, que existem índices  $i, j$  tais que

$$w(\langle v_i, \dots, v_j \rangle) > \delta(v_i, v_j)$$

$$- p_L = \langle v_0, \dots, v_{i-1} \rangle$$

$$- p_m = \langle v_i, \dots, v_j \rangle$$

$$- p_R = \langle v_{j+1}, \dots, v_n \rangle$$

$$p = p_L \cdot p_m \cdot p_R$$

$$\begin{aligned} w(p) = & w(p_L) + w(v_{i-1}, v_i) \\ & + w(p_m) + w(v_{j+1}, v_j) \\ & + w(p_R) \end{aligned}$$

- Segue que existe uma sequência de vértices  $p'_m = \langle v_i, \dots, v_j \rangle$  tal que  $w(p'_m) = \delta(v_i, v_j) < w(\langle v_i, \dots, v_j \rangle)$

- Considerese o caminho:  $p' = p_L \cdot p'_m \cdot p_R$

$$w(p') = w(p_L) + w(v_{i-1}, v_i) + w(p'_m) + w(v_{j+1}, v_j) + w(p_R) < w(p)$$

$\left. \begin{array}{l} \text{ } \\ \text{ } \end{array} \right\} \begin{array}{l} \text{ } \\ \text{ } \end{array}$ 
 } is not a  
 shortest path  
 from  $v_0$  and  $v_n$

## O operação de Relaxação

- Os algoritmos de caminhos mais curtos estudados na disciplina funcionam através do cálculo de estimativas de distância.

Associam cada vértice  $u$  a uma distância  $u.d$ .

Estimativa de distância entre  $s$  e  $u$ .

- As distâncias são actualizadas pela operação de relaxamento:

$\text{Relax}(w, u, v)$   
 $\text{if } (v.d > u.d + w(u, v))$

Initialize SingleSource( $G, s$ )  
for  $v \in G.V$

- Diferentes algoritmos usam estratégias diferentes para escolher a ordem pela qual devem efectuar as operações de relaxamento.

## O operação de Relaxação

- Os algoritmos de caminhos mais curtos estudados na disciplina funcionam através do cálculo de estimativas de distância.

Associam cada vértice  $u$  a uma distância  $u.d$ .

Estimativa de distância entre  $s$  e  $u$ .

- As distâncias são actualizadas pela operação de relaxamento:

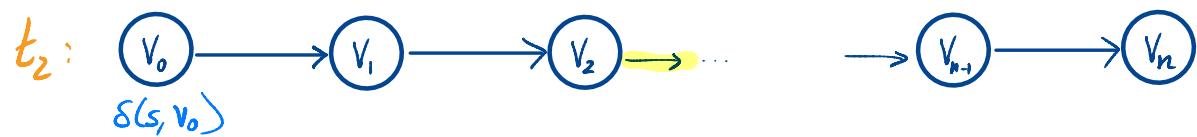
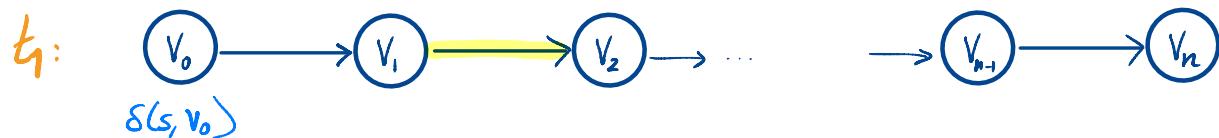
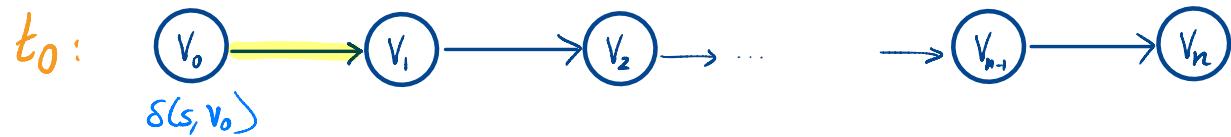
```
Relax (w, u, v)
if (v.d > u.d + w(u, v))
    v.d = u.d + w(u, v)
    v.PI = u
```

```
InitializeSingleSource (G, s)
for  $v \in G.V$ 
     $v.d := \infty$ ;  $v.PI := \text{nil}$ 
     $s.d := 0$ 
```

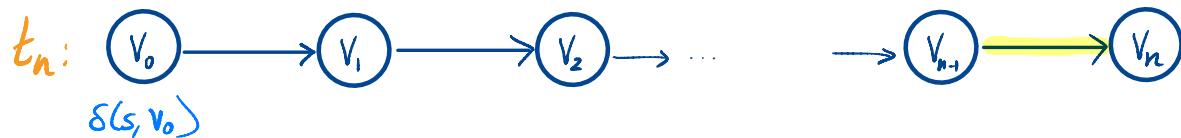
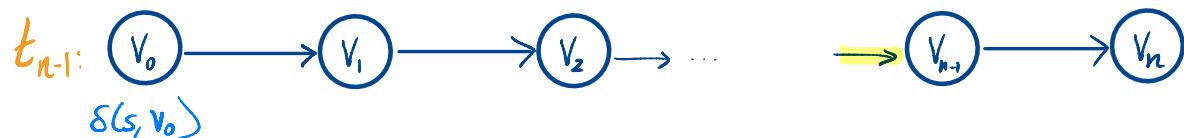
- Diferentes algoritmos usam estratégias diferentes para escolher a ordem pela qual devem efectuar as operações de relaxamento.

## Relaxação de Caminho

- $\langle v_0, \dots, v_n \rangle$  é um caminho mais curto em  $G$ :

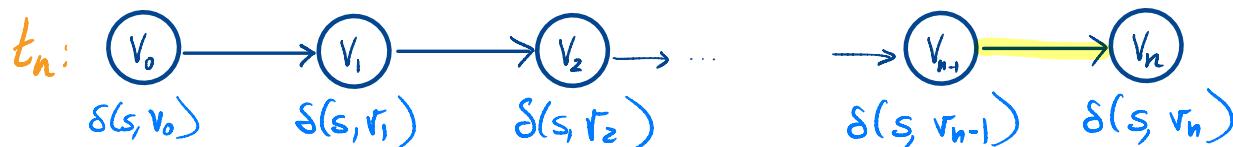
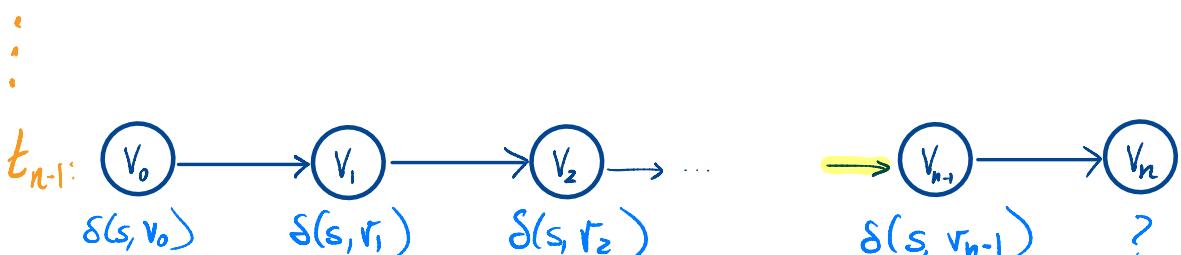
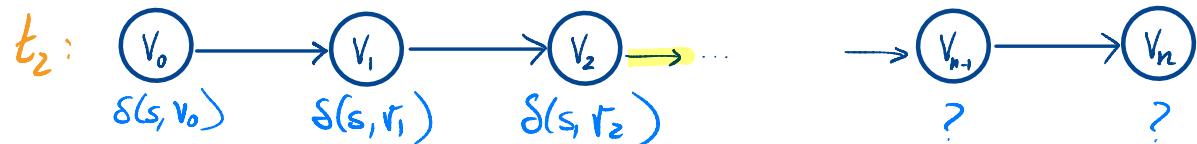
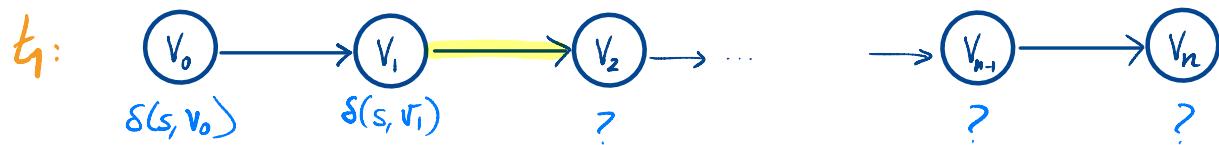
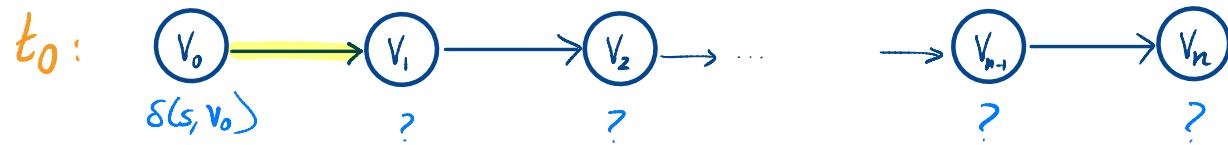


⋮



## Relaxação de Caminho

- $\langle v_0, \dots, v_n \rangle$  é um caminho mais curto em  $G$ :



## Relaxação de Caminho

### Lema [Relaxação de Caminho]

Seja  $G = (V, E, w)$  um grafo pesado e  $p = \langle v_0, \dots, v_n \rangle$  um caminho mais curto entre  $v_0$  e  $v_n$ . Se os arcos  $(v_0, v_1), \dots, (v_{n-1}, v_n)$  forem relaxados por ordem. Então, depois das  $n$  operações de relaxação, concluímos que  $v_i.d = S(s, v_i)$  para todo  $0 \leq i \leq n$ .

Prova:

- Provamos o resultado por indução em  $\underline{n}$ .
- Base  $n=0$      $p = \langle v_0 \rangle$  e     $v_0.d = S(s, v_0) \rightarrow \checkmark$
- Passo  $\underline{n+1}$ 
  - $p' = \langle v_0, \dots, v_n, v_{n+1} \rangle$  é um caminho mais curto entre  $v_0$  e  $v_{n+1}$
  - $p = \langle v_0, \dots, v_n \rangle$  é um caminho mais curto entre  $v_0$  e  $v_n$  (sub-estrutura óptima)
  - Aplicando a hipótese de indução, concluímos que depois das  $n$  primeiras operações de relaxação  $v_i.d = S(s, v_i)$  para  $0 \leq i \leq n$ . Resta provar que depois da última operação de relaxação  $v_{n+1}.d = S(s, v_{n+1})$ .

Observamos que, depois da última operação de relaxação  $v_{n+1}.d \leq v_n.d + w(v_n, v_{n+1})$

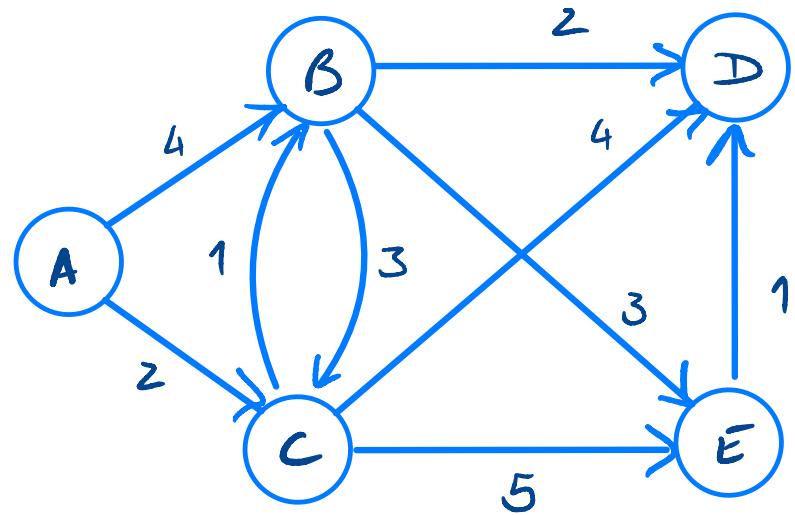
$$v_{n+1}.d \leq S(s, v_n) + w(v_n, v_{n+1})$$

$$v_{n+1}.d \leq S(s, v_{n+1})$$

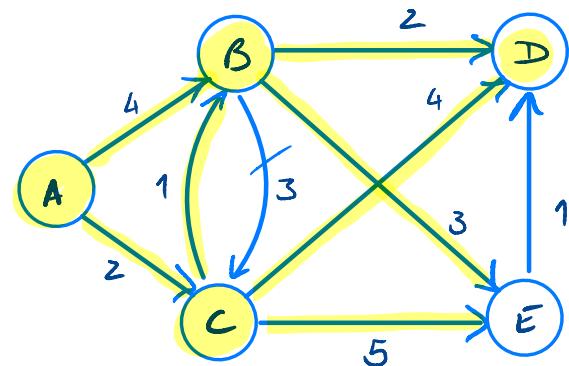
De onde segue que:

$$v_{n+1}.d = S(s, v_{n+1})$$

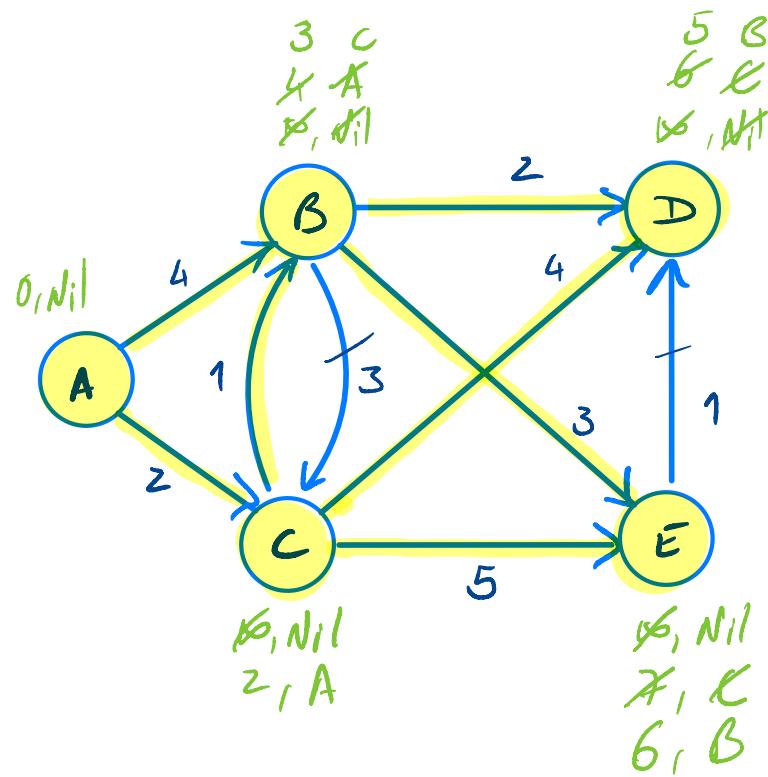
## Algoritmo de Dijkstra



## Algoritmo de Dijkstra



A	0				
B	∞	4	3		
C	∞	2			
D	∞	∞	6	5	
E	∞	∞	7	6	6



## Algoritmo de Dijkstra

Dijkstra( $G, s$ )

- ① Initialize Single Source( $G, s$ )
- ②  $Q := \text{new MinQueue}(G.V)$
- ③  $S = \{\}$
- ④ while ( $\exists v \in Q$ ) do
- $v = \text{extractMin}(Q)$
- for each  $u \in G[v]$  do
- $d_u = d_v + w(v, u)$
- if  $d_u < \infty$  and  $u \notin S$  then
- $\text{insert}(Q, u, d_u)$
- end for
- end while

fila de prioridade mínima  $Q$  com conteúdo  $G.V$

As chaves são as distâncias!

## Algoritmo de Dijkstra

Dijkstra( $G, s$ )

- ① Initialize Single Source( $G, s$ )
- ②  $Q := \text{new MinQueue}(G.V)$
- ③  $S = \{\}$
- ④ while ( $\neg Q.\text{empty}()$ ) {  
     $u := Q.\text{extractMin}();$   
     $S := S \cup \{u\};$   
    for each  $v \in G.\text{Adj}[u]$   
        Relax( $b.w, u, v$ )  
}

fila de prioridade mínima  $Q$  com todo  $G.V$

As chaves são as distâncias!

## Algoritmo de Dijkstra

Dijkstra( $G, s$ )

- ① InitializeSingleSource( $G, s$ )
- ②  $Q := \text{new MinQueue}(G.V)$
- ③  $S = \{\}$   
while ( $\neg Q.\text{empty}()$ ) {  
     $u := Q.\text{extractMin}();$   
     $S := S \cup \{u\};$   
    for each  $v \in G.\text{Adj}[u]$   
        Relax( $G, w, u, v$ )  
}
- ④

## Análise de Complexidade

- Análise agregada

①

②

③

④



---

# Algoritmo de Dijkstra

Dijkstra( $G, s$ )

- ① Initialize Single Source( $G, s$ )
- ②  $Q := \text{new MinQueue}(G.V)$
- ③  $S = \{\}$
- ④ while ( $\neg Q.\text{empty}()$ ) {  
     $u := Q.\text{extractMin}();$   
     $S := S \cup \{u\};$   
    for each  $v \in G.\text{Adj}[u]$   
        Relax( $t, w, u, v$ )  
}

## Análise de Complexidade

- Análise agregada

- ①  $O(V)$  build MinHeap
- ②  $O(V)$  min heapif
- ③  $O(V)$  iterações  
 $\hookrightarrow$  cada iteração custo  $O(\lg V)$   $\left. O(V \cdot \lg V) \right\}$
- ④  $O(E)$  iterações  
 $\hookrightarrow$  cada iteração custo  $O(\lg V)$   $\left. O(E \cdot \lg V) \right\}$   
 $\hookrightarrow$  decrease key

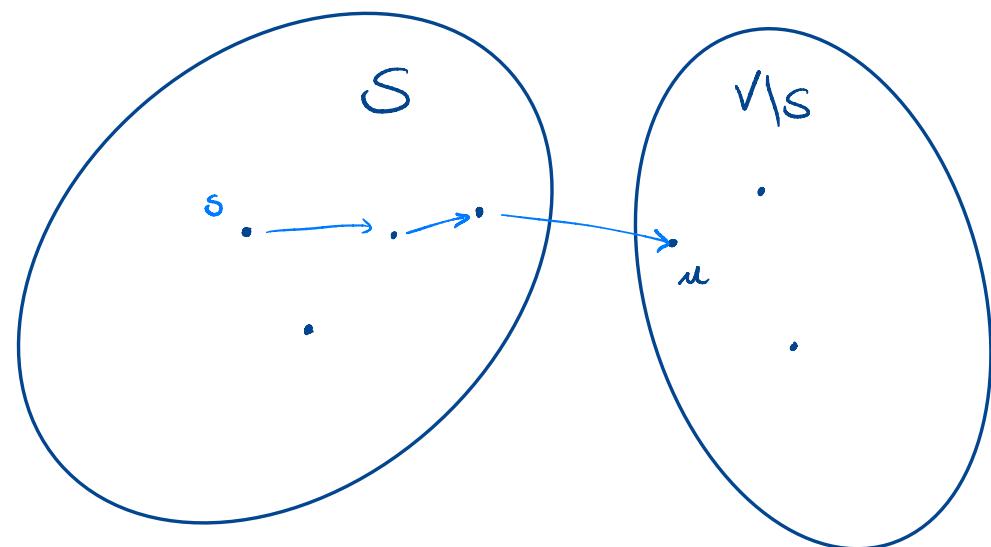


---

$$O((E + V) \lg V)$$

## Algoritmo de Dijkstra - Invariante

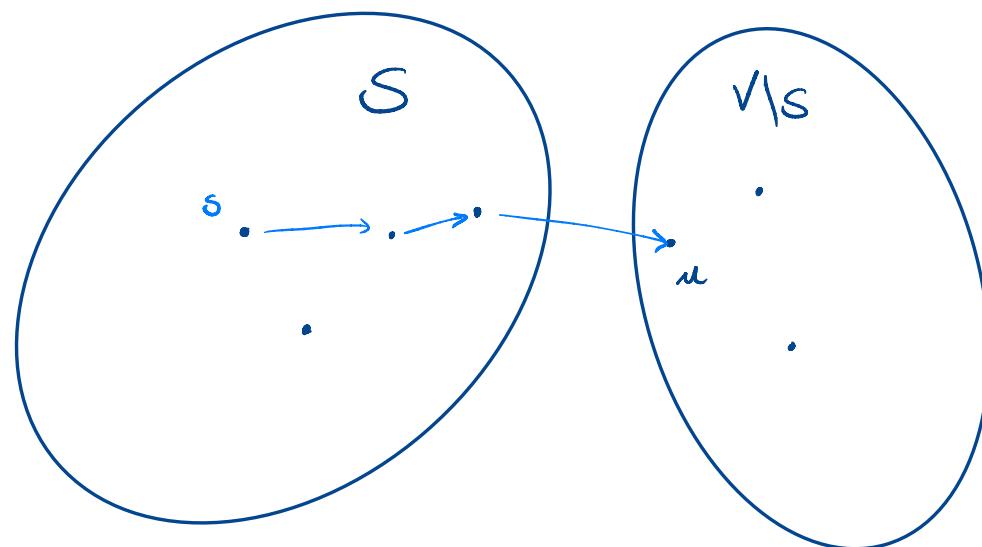
$\forall v \in S$ .



## Algoritmo de Dijkstra - Invariante

$$\forall r \in S. \quad \delta(s, r) = r.d$$

$$\wedge \quad S = V \setminus Q$$



- $u \in Q$  que:

$$u.d = \min \{ u.d \mid u \in V \setminus Q \}$$

- Há que provar que:

$$u.d = \delta(s, u)$$

- Suponhamos que  $u.d \neq \delta(s, u)$

- Existe um caminho  $\bar{p}$  que liga  $s$  a  $u$  tal que  $s \xrightarrow{\bar{p}} u \wedge w(\bar{p}) = u.d$

- Seja  $p$  o predecessor de  $u$  no caminho mais curto entre  $s$  e  $u$ .

## Algoritmo de Dijkstra - Invariante

$$\forall r \in S. \quad d(s, r) = r.d \\ \wedge \quad S = V \setminus Q$$

- $m \in \text{tal que}$ :  
 $m.d = \min \{ u.d \mid u \in V \setminus Q \}$

- Há que provar que:  
 $m.d = d(s, m)$

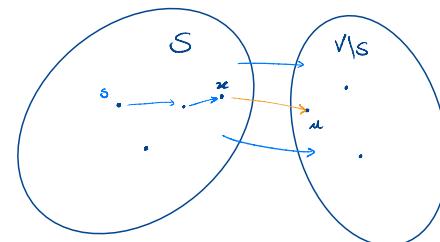
- Suponhamos que  $m.d \neq d(s, m)$

- Existe um caminho  $p \neq q$  liga  $s$  a  $m$  tal que  $s \xrightarrow{p} m \wedge w(p) = m.d$

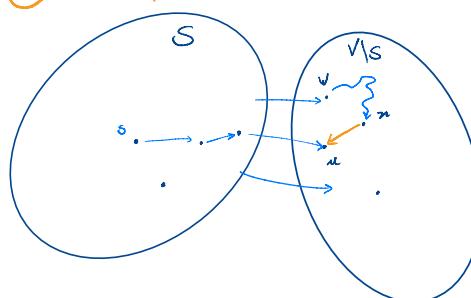
- Seja  $x$  o predecessor de  $m$  no caminho mais curto entre  $s$  e  $m$ .

- 2 casos a considerar:

①  $x \in S$



②  $x \notin S$



## Algoritmo de Dijkstra - Invariante

$$\forall r \in S. \quad \delta(s, r) = r.d$$

$$\wedge \quad S = V \setminus Q$$

• M é tal que:

$$m.d = \min \{ u.d \mid u \in V \setminus Q \}$$

• Há que provar que:

$$m.d = \delta(s, m)$$

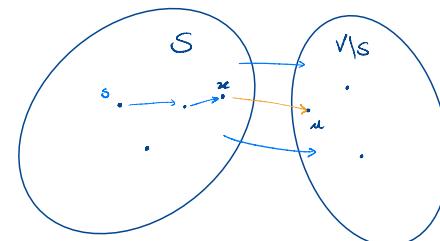
• Suponhamos que  $m.d \neq \delta(s, m)$

- Existe um caminho  $p \neq j$  liga s a m tal que  $s \xrightarrow{p} m \wedge w(p) = m.d$

- Seja  $x$  o predecessor de  $m$  no caminho mais curto entre  $s$  e  $m$ .

- 2 casos a considerar:

I)  $x \in S$



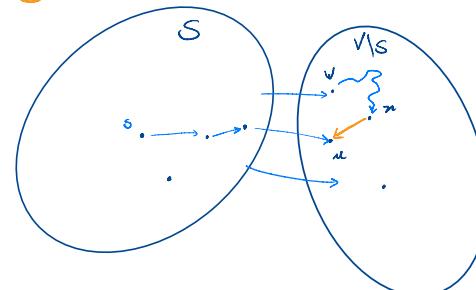
$$x.d = \delta(s, x) \quad (\text{invariante})$$

• O arco  $(x, m)$  já foi relaxado

$$m.d = x.d + w(x, m)$$

$$= \delta(s, m)$$

II)  $x \notin S$



$$\delta(s, m) = \delta(s, x) + w(x, m)$$

• Seja  $w$  o 1º vizinho de  $p$  em  $V \setminus S$   
 $w.d = \delta(s, w)$

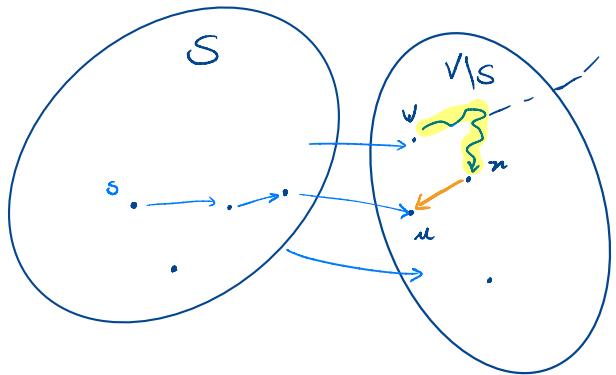
$$w.d = \delta(s, w) \leq \delta(s, m) < m.d$$

$$w.d < m.d$$

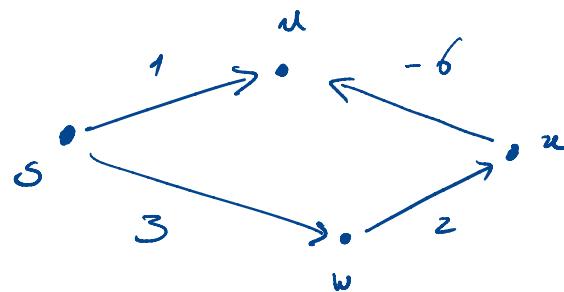
Mas m é o vértice com distância mais baixa

## Algoritmo de Dijkstra - Peso Negativo

II)  $n \notin S$

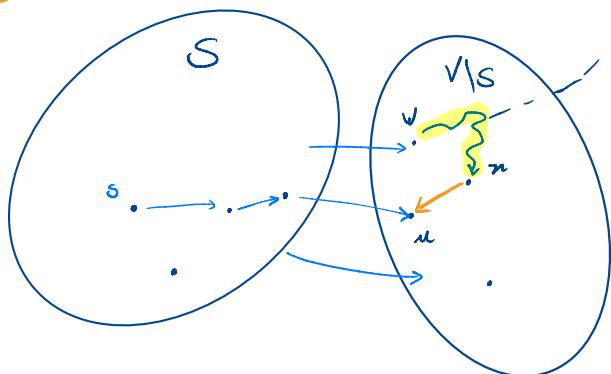


→ É se o caminho entre  $w$  e  $u$  tiver peso negativo?

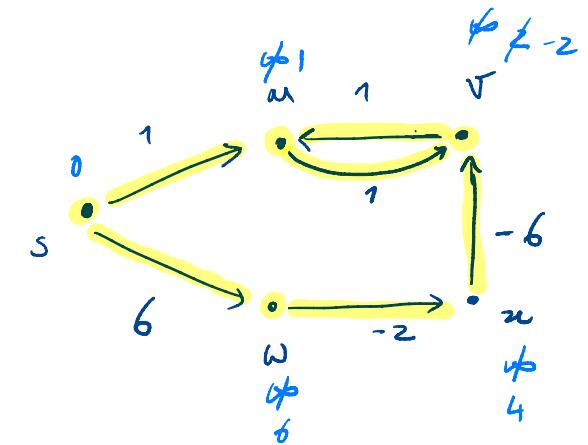
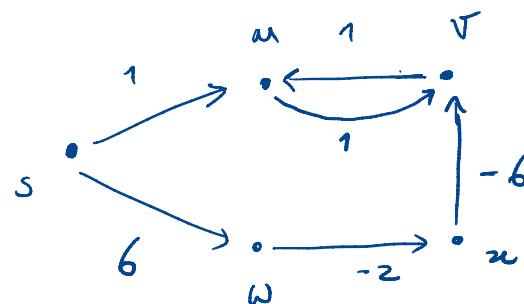
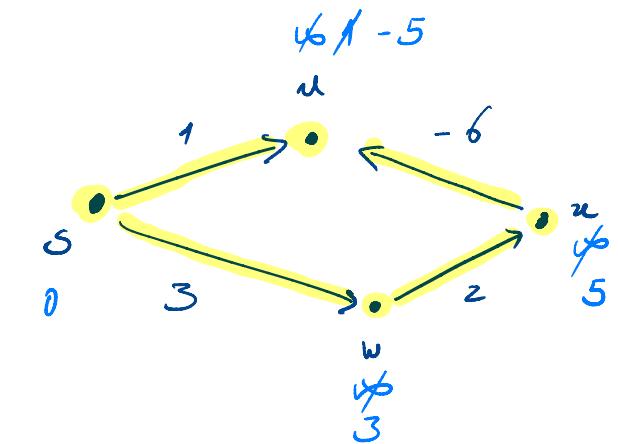
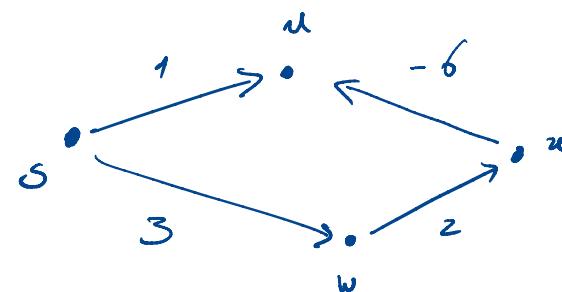


## Algoritmo de Dijkstra - Peso Negativo

II)  $n \notin S$



→ Existe caminho entre w e u com peso negativo?



## Algoritmo de Bellman-Ford

Bellman-Ford ( $G, s$ )

InitializeSingleSource ( $G, s$ ) {

for  $i := 1$  to  $|G.V| - 1$   
    for each  $(u, v) \in G.E$   
        Relax ( $G, u, v$ ) }

for each  $(u, v) \in E$   
    if  $v.d > u.v + G.w(u, v)$   
        return false  
    return true }

Análise de Complexidade

Total:

## Algoritmo de Bellman-Ford

Bellman-Ford ( $G, s$ )

InitializeSingleSource ( $G, s$ ) {  $O(V)$

for  $i := 1$  to  $|G.V| - 1$   
for each  $(u, v) \in G.E$       }  $O(V \cdot E)$   
    Relax ( $G, u, v$ )

Análise de Complexidade

Total:  $O(V \cdot E)$

for each  $(u, v) \in E$   
if  $v.d > u.v + G.w(u, v)$       }  $O(E)$   
: return false  
return true

## Algoritmo de Bellman-Ford - Conexão

- ① Se o grafo não tem ciclos negativos, o algoritmo de Bellman-Ford calcula as distâncias conectas:

$$\forall v \in V. \quad v.d = \delta(s, v)$$

Prova:

- Considere-se um vértice qualquer  $v \in V$ . Seja  $p$  o caminho mais curto que liga  $s$  a  $v$  ( $s \xrightarrow{p} v$ ).
- Temos de provar q no fim do algoritmo de BF,  $v.d = \delta(s, v)$ .
- Sabemos que o caminho  $p$  tem nó máximo  $|V|-1$  arcos.
- Depois de  $|V|-1$  etapas de relaxação nas quais relaxamos todos os arcos do grafo, existe uma subsequência de relaxações correspondentes ao caminho mais curto entre  $s$  e  $v$ .
- O resultado segue pelo lema da Relaxação de Caminho.

## Algoritmo de Bellman-Ford - Conexão

(2) O algoritmo de Bellman-Ford retorna false se o grafo contém um ciclo negativo

(2.1)  $BF(G, s) = \text{false} \Rightarrow G \text{ contém um ciclo negativo}$

## Algoritmo de Bellman-Ford - Conexão

(2) O algoritmo de Bellman-Ford retorna false se o grafo contém um ciclo negativo

(2.1)  $BF(G, s) = \text{false} \Rightarrow G \text{ contém um ciclo negativo}$

$\Leftrightarrow G \text{ não contém um ciclo negativo} \Rightarrow BF(G, s) = \text{true}$



$$\forall (u, v) \in E. \quad v.d \leq u.d + w(u, v)$$

- $G$  não contém um ciclo negativo (hyp)
- $\forall v \in V. \quad v.d = \delta(s, v)$
- $\forall (u, v) \in E. \quad \delta(s, v) \leq \delta(s, u) + w(u, v)$  (desigualdade triangular)
- $\forall (u, v) \in E. \quad v.d \leq u.d + w(u, v)$  (conexão: (1))

## Algoritmo de Bellman-Ford - Conexão

- (2) O algoritmo de Bellman-Ford retorna false se o grafo contém um ciclo negativo
- (2.1)  $G$  contém um ciclo negativo  $\Rightarrow BF(t, s) = \text{false}$

## Algoritmo de Bellman-Ford - Conexão

(2) O algoritmo de Bellman-Ford retorna false se o grafo contém um ciclo negativo

(2.1)  $G$  contém um ciclo negativo  $\Rightarrow BF(G, s) = \text{false}$

- Suponhamos, por contradição, que  $G$  contém um ciclo negativo e  $BF(G, s) = \text{true}$

- Seja  $p = \langle v_0, v_1, \dots, v_{n-1}, v_n \rangle$ , com  $v_n = v_0$ , o ciclo negativo.

$$\forall 0 \leq i \leq n-1. (v_i, v_{i+1}) \in E$$

$$\Rightarrow \forall 0 \leq i \leq n-1. v_{i+1}.d \leq v_i.d + w(v_i, v_{i+1})$$

$$\Rightarrow \sum_{i=0}^{n-1} v_{i+1}.d \leq \sum_{i=0}^{n-1} v_i.d + w(v_i, v_{i+1})$$

$$\Rightarrow \sum_{i=0}^{n-1} v_{i+1}.d \leq \underbrace{\sum_{i=0}^{n-1} v_i.d}_{=} + \underbrace{\sum_{i=0}^{n-1} w(v_i, v_{i+1})}_{\text{peso do caminho}}$$

$$w(p) \geq 0$$

$\therefore$

(contradição)

$\Rightarrow p$  não é ciclo negativo !!

# Caminhos Mais Curtos em Grafos Acíclicos

## Lema [Caminho Mais Curto em DAG]

Seja  $G = (V, E)$  um grafo acíclico e  $\langle v_0, \dots, v_n \rangle$  uma ordenação topológica de  $G$ ; então qualquer caminho em  $G$  é uma subsequência de  $\langle v_0, \dots, v_n \rangle$ .

### Prova:

- Suponhamos que existe um caminho em  $G$ ,  $\langle u_1, \dots, u_k \rangle$  tal que  $\langle u_1, \dots, u_k \rangle$  não é uma subsequência de  $\langle v_0, \dots, v_n \rangle$ .

Concluímos que existe um índice  $i \in \{1, \dots, k\}$  que  $u_i$  ocorre depois de  $u_{i+1}$  na ordenação topológica.

- $(u_i, u_{i+1}) \in E \Rightarrow f(u_i) > f(u_{i+1})$



### Observação Clave:

- Todos os caminhos mais curtos são subsequências da ordenação topológica

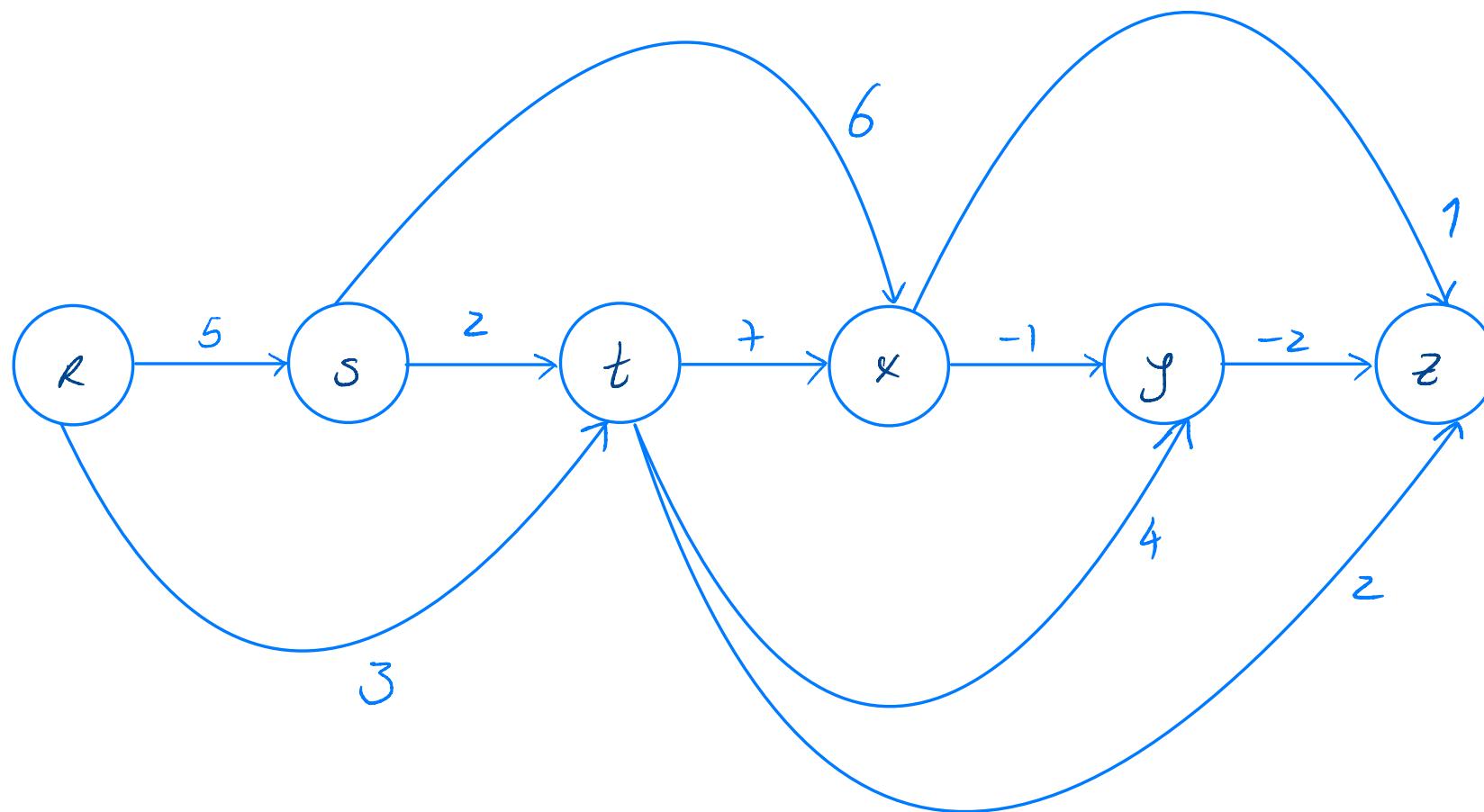
+

- Lema da Relaxação de Caminho

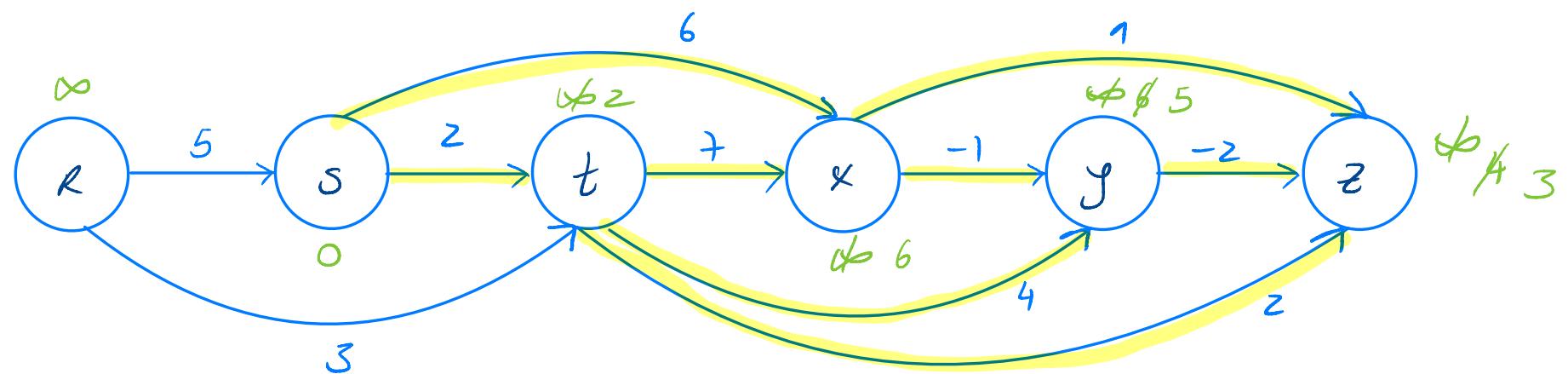


- Em DAGs podemos calcular caminhos mais curtos, relaxando as arestas do grafo por ordem topológica!

Caminhos Mais Curtos em Grafos Acíclicos



## Caminhos Mais Curtos em Grafos Acíclicos



## Caminhos Mais Curtos em Grafos Acíclicos

DAG - Shortest-Paths ( $G, s$ )

① Initialize-Single-Source ( $G, s$ )

②  for each  $u \in G.V$  in topological order  
    for each  $v \in G.Adj[u]$   
        Relax( $G, w, u, v$ )

③

④

Análise de Complexidade

①  $O(V)$

②  $O(V)$

③  $O(E)$

$$\frac{(4) O(V+E)}{O(V+E)} =$$

