AULA 4

Problema do caminho mais curto de uma única origem em grafos

Karina Valdivia Delgado

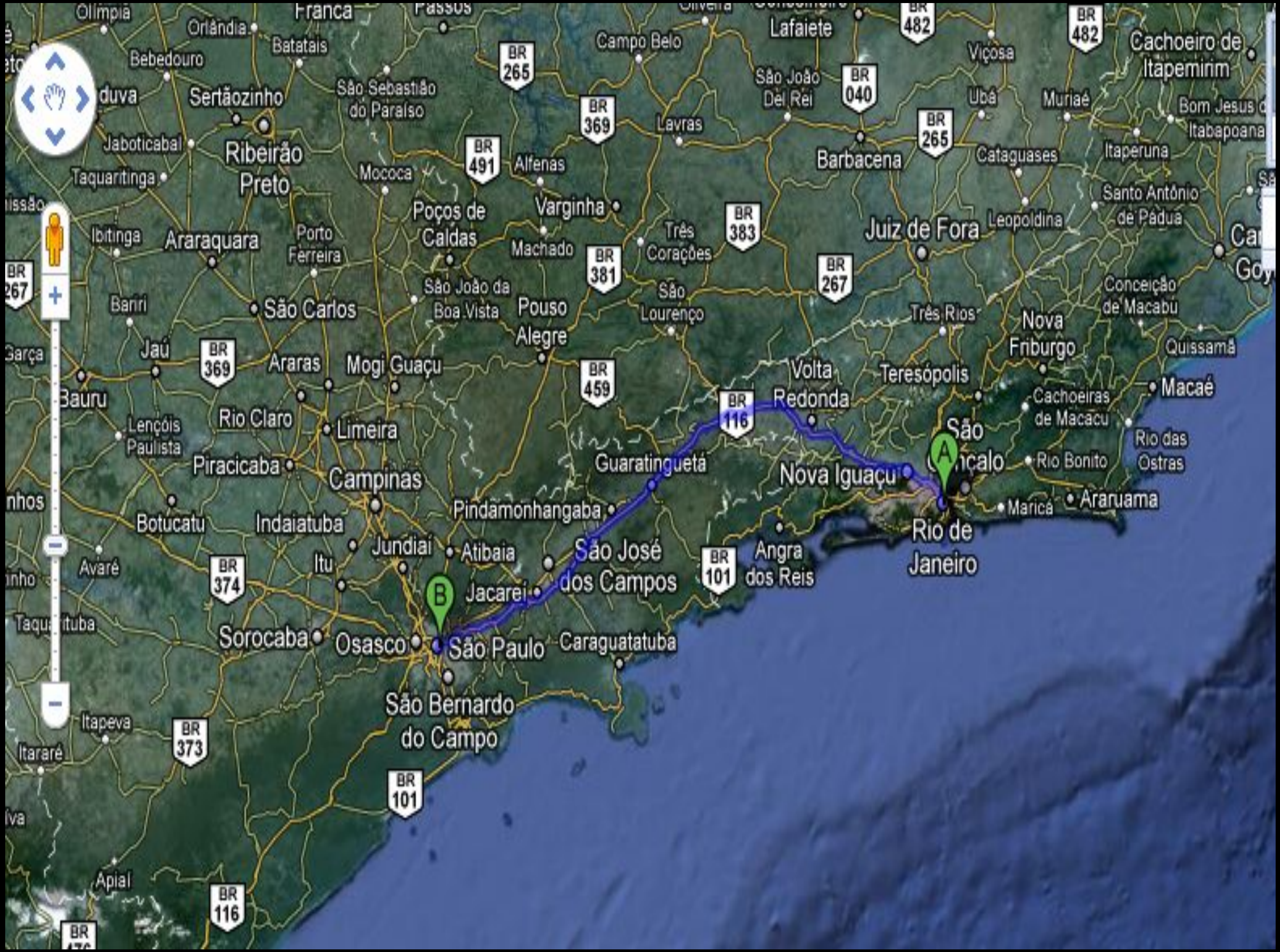
Roteiro

Motivação
Relaxamento
Algoritmo de Bellman-Ford
Algoritmo de Dijkstra

Motivação

Suponha que você deseja encontrar o caminho mais curto possível do Rio de Janeiro a São Paulo.

Como determinar a rota mais curta?



Caminho mais curto de origem única.

Temos um grafo orientado ponderado $G=(V,A)$

Uma função peso $w: A \rightarrow \Re$

O peso do caminho $p=<v_0,v_1,...,v_k>$ é:

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

Definimos o peso do caminho mais curto desde u até v por:

$$\delta(u,v)=\begin{cases}\min\{w(p) : u \overset{p}{\rightsquigarrow} v\}, & \text{se existe caminho de u até v}\\ \infty, & \text{caso contrário}\end{cases}$$

Sub-estrutura ótima

Seja $G=(V,A)$ um grafo orientado ponderado, com função peso $w:A\rightarrow\mathfrak{R}$:

Seja o caminho $p=<v_1,v_2,\ldots,v_k>$ um caminho mais curto de v_1 até v_k.

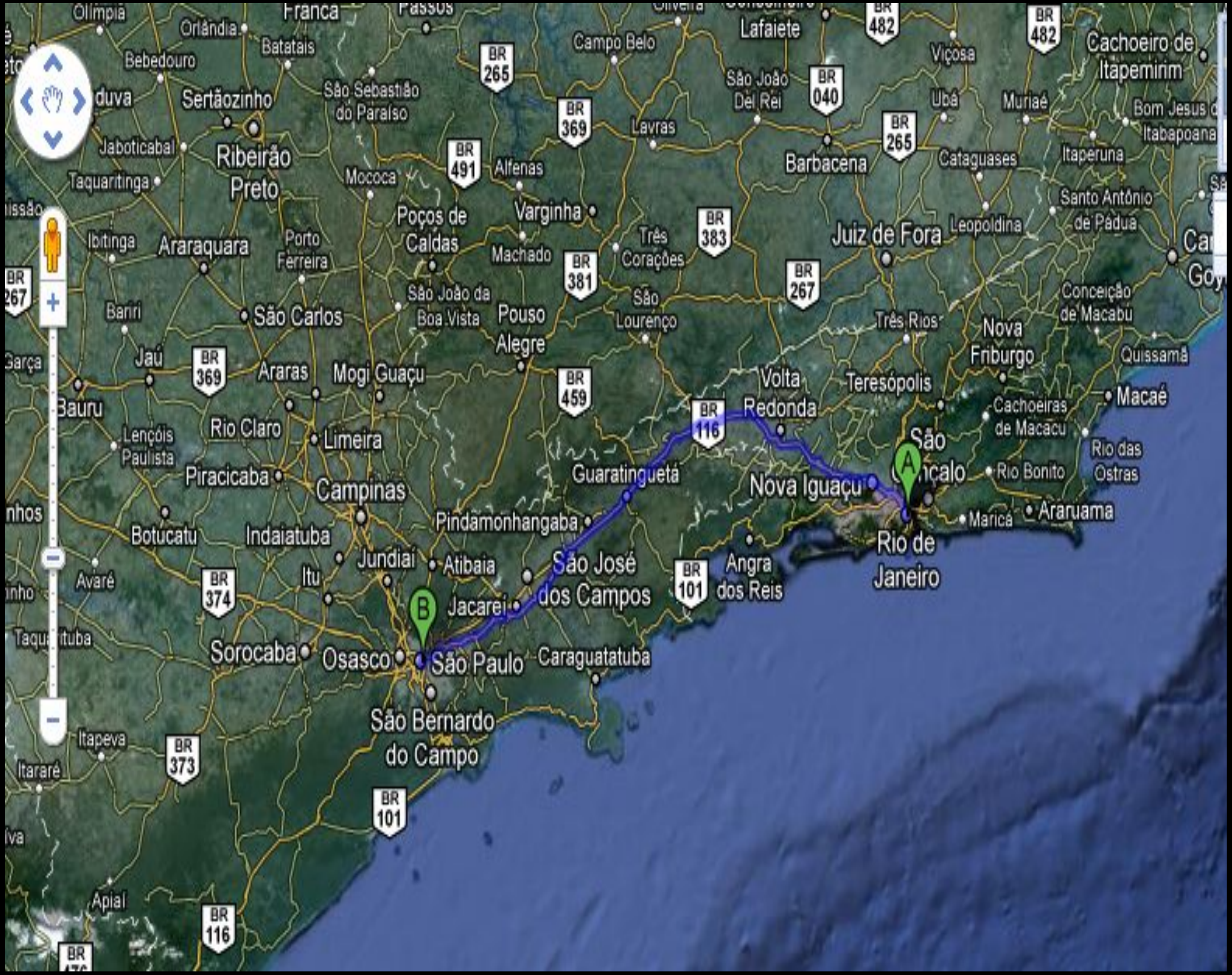
Seja $p_{ij}=<v_i,v_{i+1},,\ldots,v_j>$ o sub-caminho de p desde o vértice v_i até o vértice v_j, para quaisquer i e j tais que $1<=i<=j<=k$.

Então p_{ij} é um caminho mais curto de v_i até v_j.

Sub-estrutura ótima

Suponha que o caminho mais curto de Rio de Janeiro a São Paulo é o mostrado no mapa.

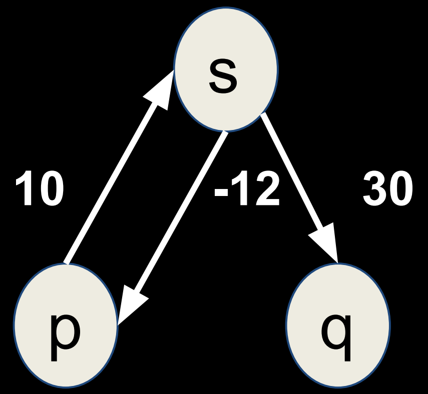
Então o sub-caminho de Guaratinguetá a São Paulo também é um sub-caminho mais curto entre elas.



Arestas de peso negativo

Podem existir arestas cujos pesos são negativos.

Se existe um **ciclo de peso negativo** acessível a partir de s, os pesos de caminhos mais curtos não são bem definidos pois sempre será possível encontrar um caminho de peso menor que o já encontrado.



Arestas de peso negativo

Se existe um ciclo de peso negativo em algum caminho entre s até v, definimos:

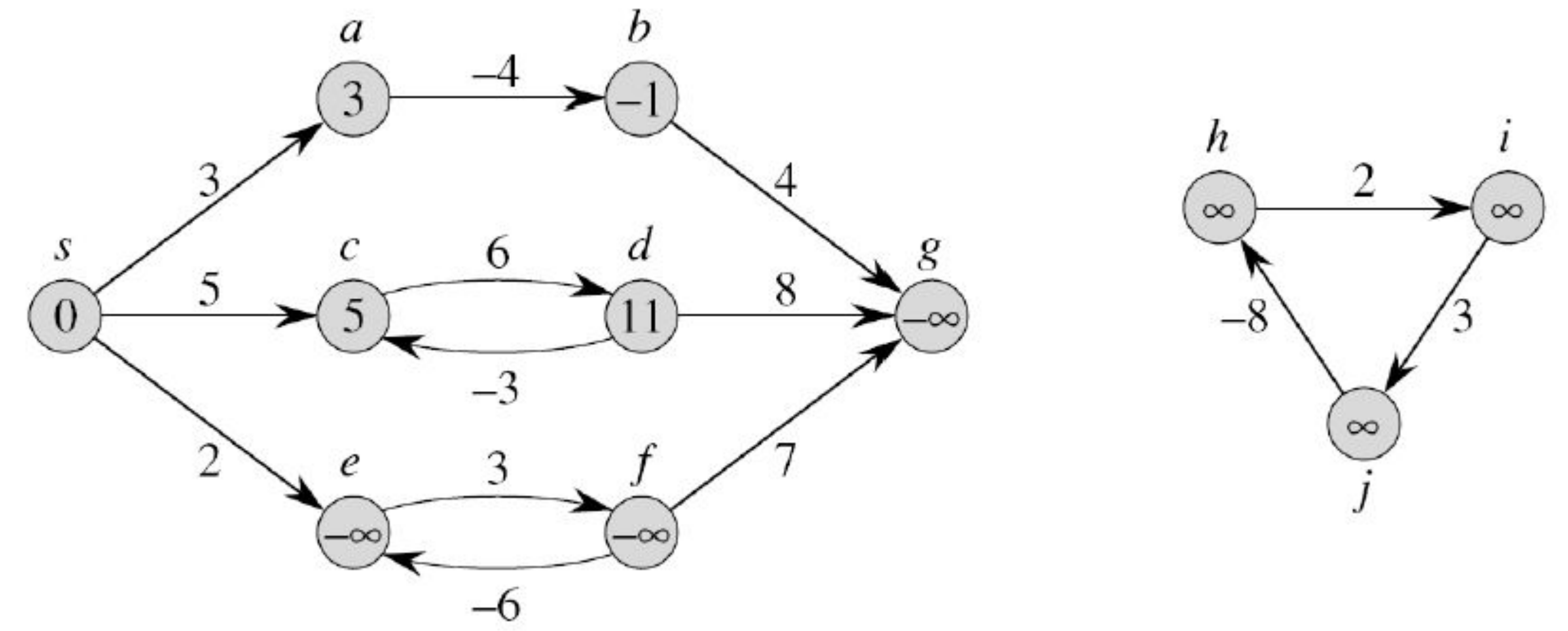
$$\delta(s,v)=-\infty$$

Arestas de peso negativo

- Lembrando:

$$\delta(u,v) = \begin{cases} \min\{ w(p) : u \overset{p}{\leadsto} v \}, & \text{se existe caminho de u até v} \\ \infty, & \text{caso contrário} \end{cases}$$

- E se existe um ciclo de peso negativo accesível a partir de s então δ(s,v)= -∞

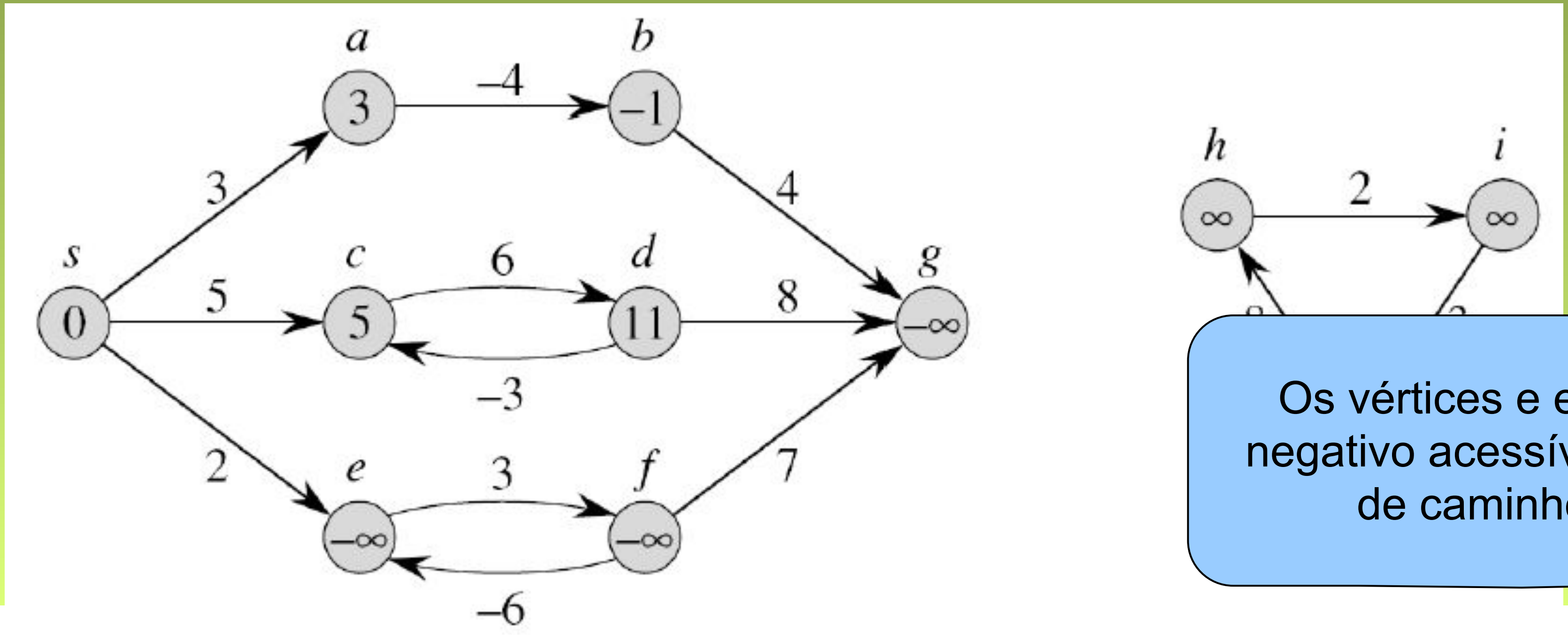


Arestas de peso negativo

- Lembrando:

$$\delta(u,v) = \begin{cases} \min\{w(p) : u \overset{p}{\leadsto} v\}, & \text{se existe caminho de u até v} \\ \infty, & \text{caso contrário} \end{cases}$$

- E se existe um ciclo de peso negativo accesível a partir de s então δ(s,v)= -∞



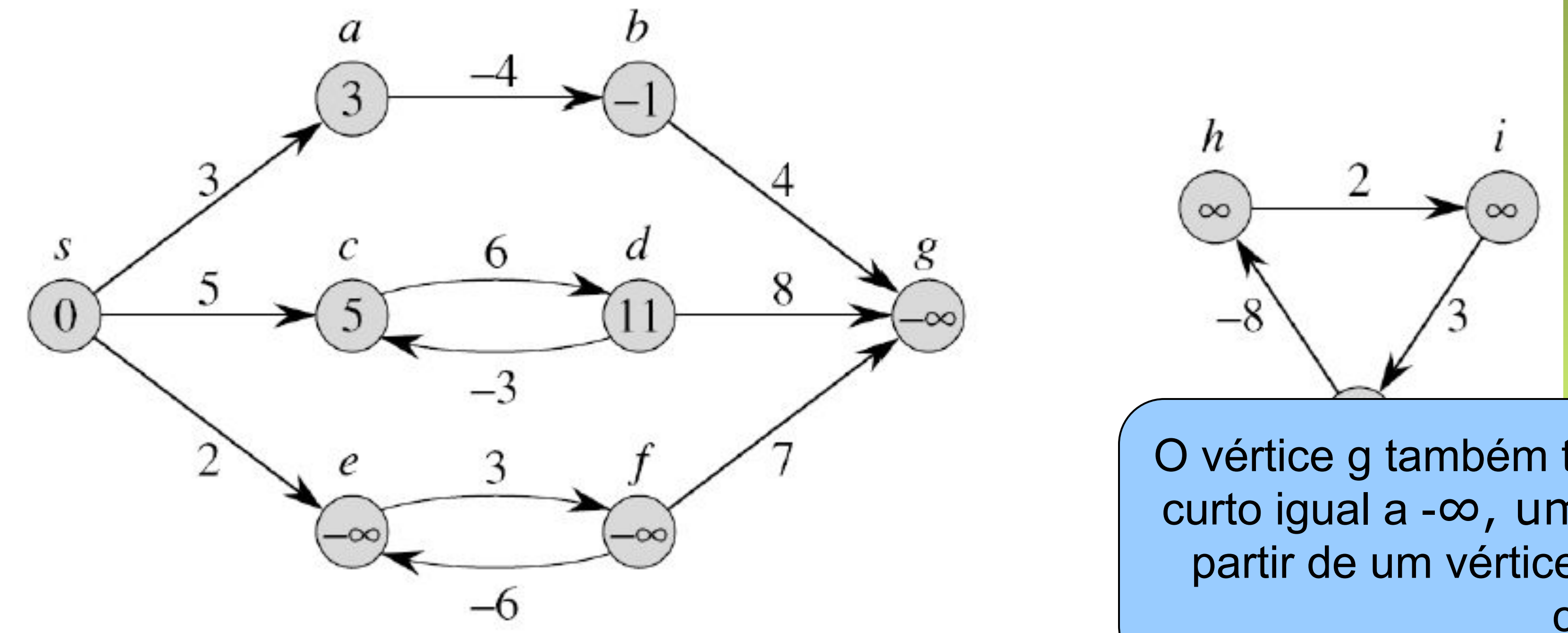
Os vértices e e f formam um ciclo de peso negativo acessível a partir de s, então o peso de caminho mais curto deles é -∞

Arestas de peso negativo

- Lembrando:

$$\delta(u,v) = \begin{cases} \min\{w(p) : u \overset{p}{\leadsto} v\}, & \text{se existe caminho de u até v} \\ \infty, & \text{caso contrário} \end{cases}$$

- E se existe um ciclo de peso negativo accesível a partir de s então δ(s,v)= -∞



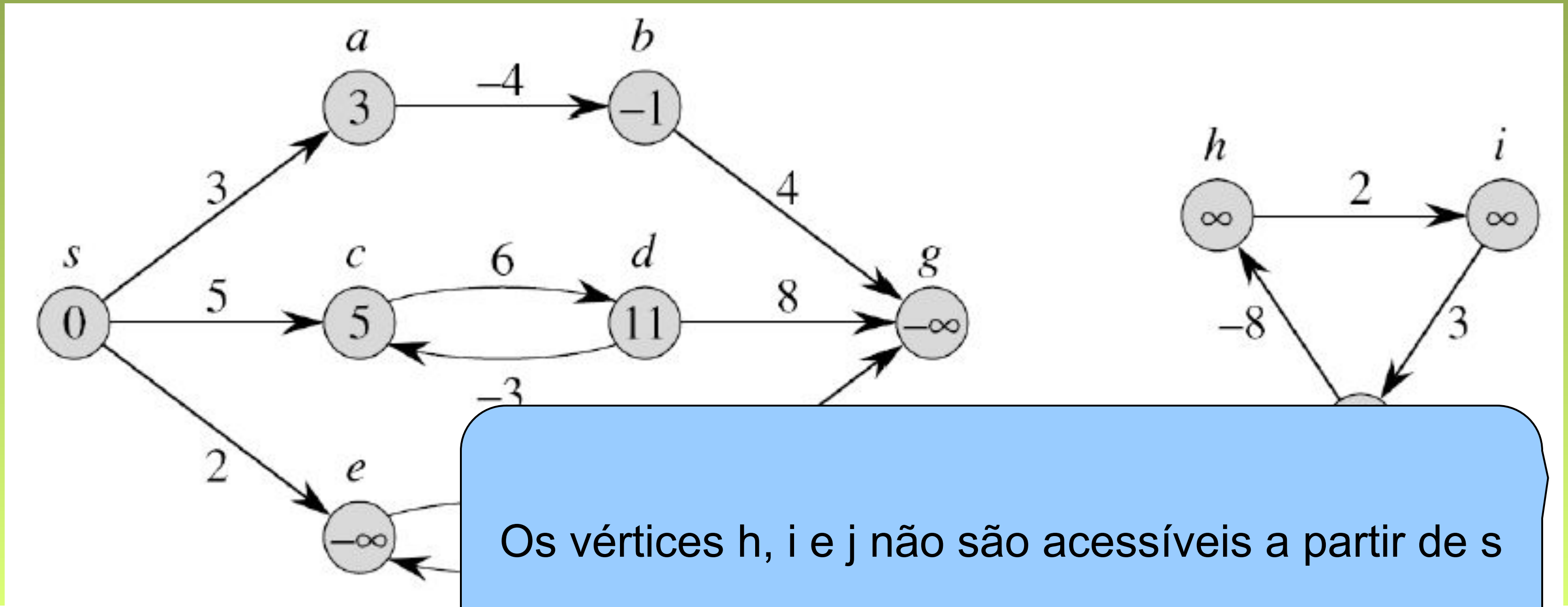
O vértice g também tem um peso de caminho mais curto igual a -∞, uma vez que ele é acessível a partir de um vértice cujo peso de caminho mais curto é -∞.

Arestas de peso negativo

- Lembrando:

$$\delta(u,v) = \begin{cases} \min\{ w(p) : u \overset{p}{\leadsto} v \}, & \text{se existe caminho de u até v} \\ \infty, & \text{caso contrário} \end{cases}$$

- E se existe um ciclo de peso negativo accesível a partir de s então δ(s,v)= -∞



Os vértices h, i e j não são acessíveis a partir de s

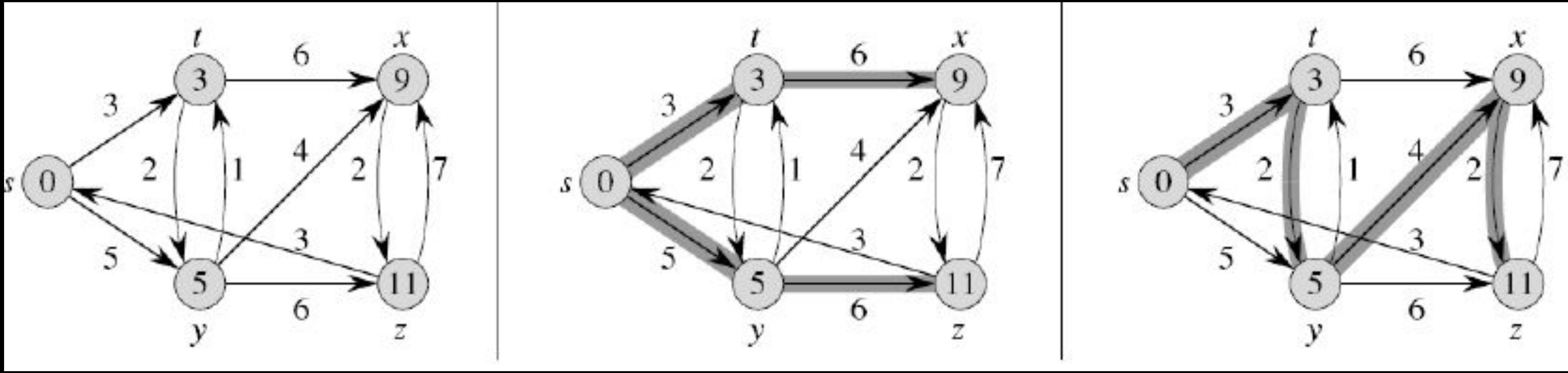
Algoritmos para caminho mais curto de origem única

Algoritmo de Dijkstra: supõe que todos os pesos das arestas no grafo de entrada são não negativos. Ex: mapa rodoviário.

Algoritmo de Bellman-Ford: permite arestas de peso negativo no grafo de entrada e produz uma resposta correta detectando a existência de ciclos.

Árvore do caminho mais curto

É uma árvore enraizada que contém um caminho mais curto desde a origem s até todo vértice acessível a partir de s.



Relaxamento

Os algoritmos de Dijkstra e Bellman-Ford usam a técnica de relaxamento.

É calculada uma estimativa do caminho mais curto:

d[v]: limite superior sobre o peso de um caminho mais curto desde a origem s até v

Relaxamento

Inicialização: são inicializadas as estimativas de caminhos mais curtos e os predecessores de cada vértice.

```
INITIALIZE-SINGLE-SOURCE(V,A,s)

1. for cada vértice v ∈ V
2.     d[v] ←∞
3.     π[v]  ← NIL
4. d[s] ← 0
```

Relaxamento

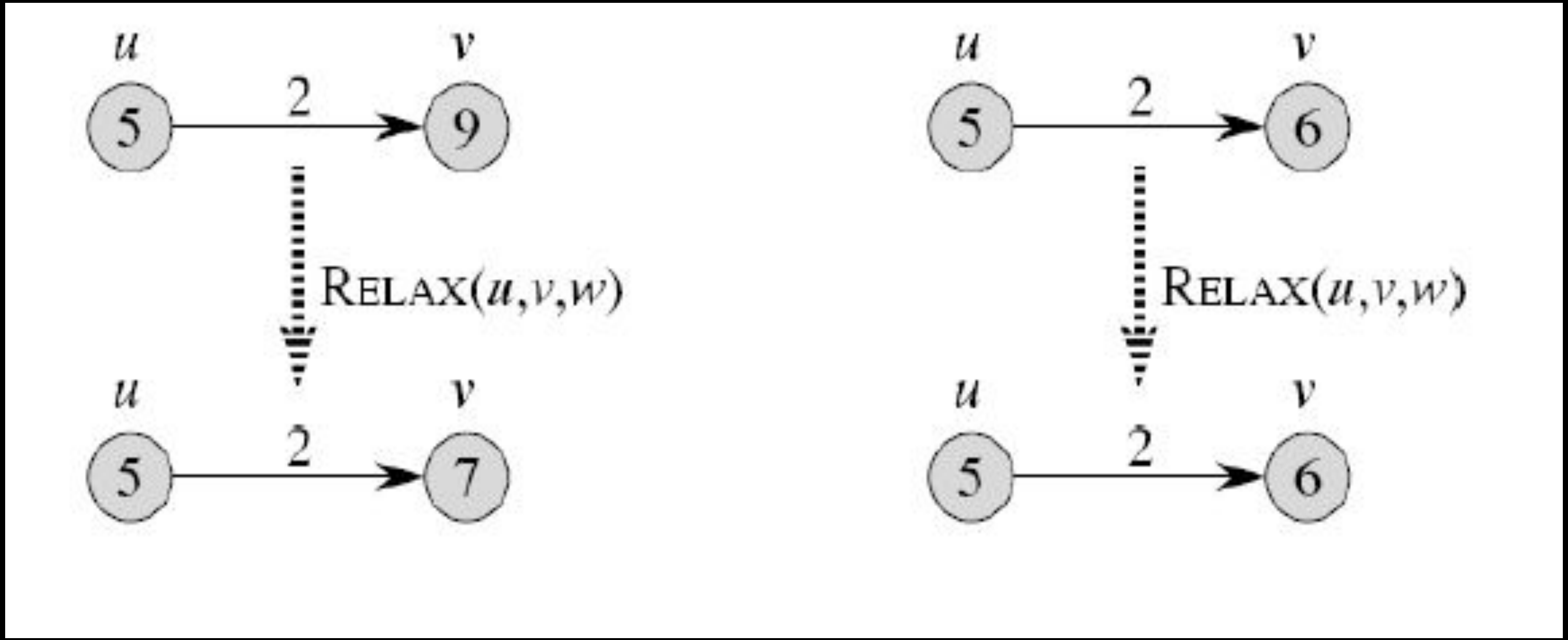
O processo de relaxar uma aresta (u,v) consiste em testar se podemos melhorar o caminho mais curto para v encontrado até agora pela passagem através de u e, neste caso, atualizar d[v] e π[v].

```
RELAX(u,v,w)

   1. if d[v]>d[u]+w(u,v)
   2.     then d[v] ← d[u]+w(u,v)
   3.              π[v]  ← u
```

Relaxamento

O processo de relaxar uma aresta (u,v) consiste em testar se podemos melhorar o caminho mais curto para v encontrado até agora pela passagem através de u e, neste caso, atualizar d[v] e π[v].



O algoritmo de Bellman-Ford

- Resolve o problema de caminhos mais curtos de única origem no caso mais geral
- Os pesos das arestas podem ser negativos
- Devolve verdadeiro se existe um ciclo de peso negativo acessível a partir da origem.
- Se não existe tal ciclo, o algoritmo encontra os caminhos mais curtos

Algoritmo de Bellman-Ford

```
BELLMAN-FORD (V,A, w, s)

1. INITIALIZE-SINGLE-SOURCE (V, A, s)
2. for i = 1 to |V| - 1 do
3.      for each edge (u, v) in A do
4.           RELAX (u, v, w)
5. for each edge (u, v) in A do
6.      if d[u] + w(u, v) < d[v] then
7.           return FALSE
8. return TRUE
```

Algoritmo de Bellman-Ford

```
BELLMAN-FORD (V,A, w, s)

  1. INITIALIZE-SINGLE-SOURCE (V, A, s)
  2. for i = 1 to |V| - 1 do
  3.     for each edge (u, v) in A do
  4.         RELAX (u, v, w)
  5. for each edge (u, v) in A do
  6.     if d[u] + w(u, v) < d[v] then
  7.         return FALSE
  8. return TRUE
```

Usa o processo de relaxamento diminuindo a estimativa d[v]: peso de uma caminho mais curto desde a origem s até v

Algoritmo de Bellman-Ford

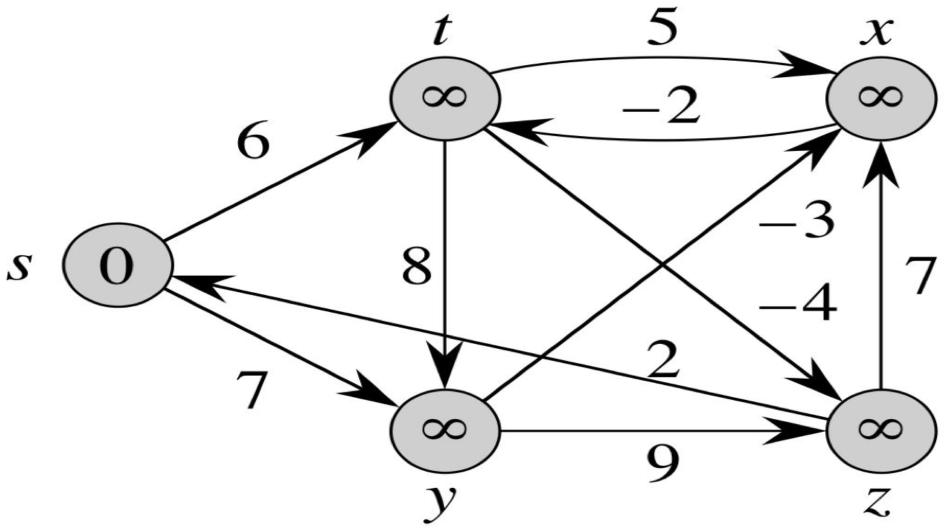
```
BELLMAN-FORD (V,A, w, s)

  1. INITIALIZE-SINGLE-SOURCE (V, A, s)
  2. for i = 1 to |V| - 1 do
  3.     for each edge (u, v) in A do
  4.         RELAX (u, v, w)
  5. for each edge (u, v) in A do
  6.     if d[u] + w(u, v) < d[v] then
  7.         return FALSE
  8. return TRUE
```

Procuramos por um ciclo de peso negativo

Algoritmo de Bellman-Ford

BELLMAN-FORD (V,A, w, s)
1. INITIALIZE-SINGLE-SOURCE (V, A, s)
2. for i = 1 to |V| - 1 do
3.   for each edge (u, v) in A do
4.   RELAX (u, v, w)
5. for each edge (u, v) in A do
6.   if d[u] + w(u, v) < d[v] then
7.   return FALSE
8. return TRUE

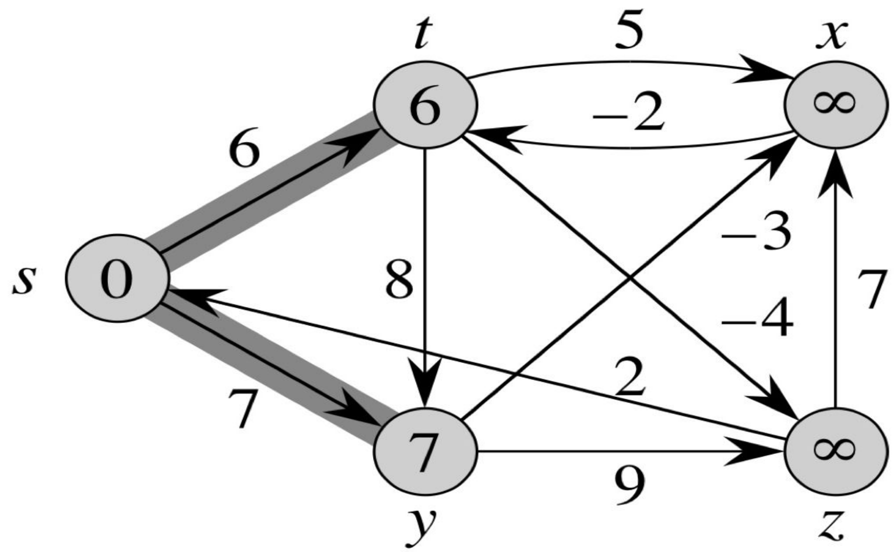


(t,x), (t,y), (t,z), (x,t), (y,x), (y,z), (z,x), (z,s), (s,t), (s,y)

vértice	s	t	x	y	z
d	0	∞	∞	∞	∞
π	NIL	NIL	NIL	NIL	NIL

Algoritmo de Bellman-Ford

BELLMAN-FORD (V,A, w, s)
1. INITIALIZE-SINGLE-SOURCE (V, A, s)
2. for i = 1 to |V| - 1 do
3. for each edge (u, v) in A do
4. RELAX (u, v, w)
5. for each edge (u, v) in A do
6. if d[u] + w(u, v) < d[v] then
7. return FALSE
8. return TRUE

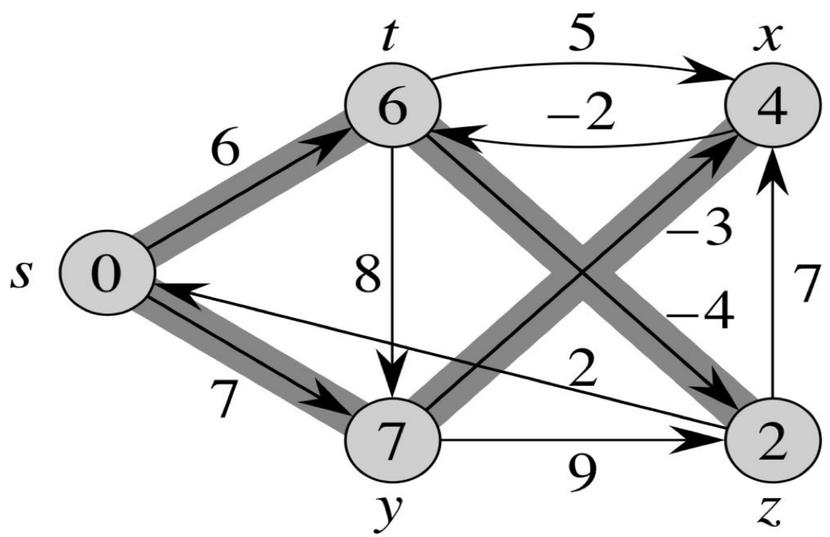


(t,x), (t,y), (t,z), (x,t), (y,x), (y,z), (z,x), (z,s), (s,t), (s,y)

vértice	s	t	x	y	z
d	0	6	∞	7	∞
π	NIL	s	NIL	s	NIL

Algoritmo de Bellman-Ford

BELLMAN-FORD (V,A, w, s)
1. INITIALIZE-SINGLE-SOURCE (V, A, s)
2. for i = 1 to |V| - 1 do
3.   for each edge (u, v) in A do
4.   RELAX (u, v, w)
5. for each edge (u, v) in A do
6.   if d[u] + w(u, v) < d[v] then
7.   return FALSE
8. return TRUE

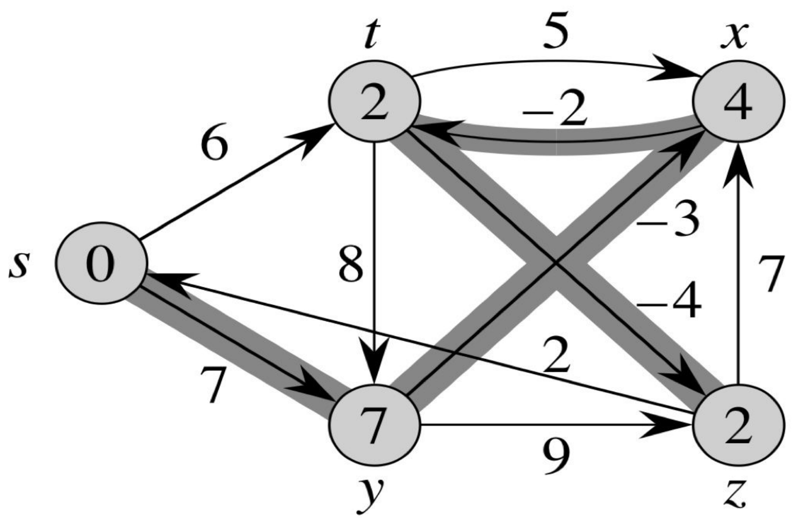


(t,x), (t,y), (t,z), (x,t), (y,x), (y,z), (z,x), (z,s), (s,t), (s,y)

vértice	s	t	x	y	z
d	0	6	4	7	2
π	NIL	s	y	s	t

Algoritmo de Bellman-Ford

BELLMAN-FORD (V,A, w, s)
1. INITIALIZE-SINGLE-SOURCE (V, A, s)
2. for i = 1 to |V| - 1 do
3.   for each edge (u, v) in A do
4.   RELAX (u, v, w)
5. for each edge (u, v) in A do
6.   if d[u] + w(u, v) < d[v] then
7.   return FALSE
8. return TRUE

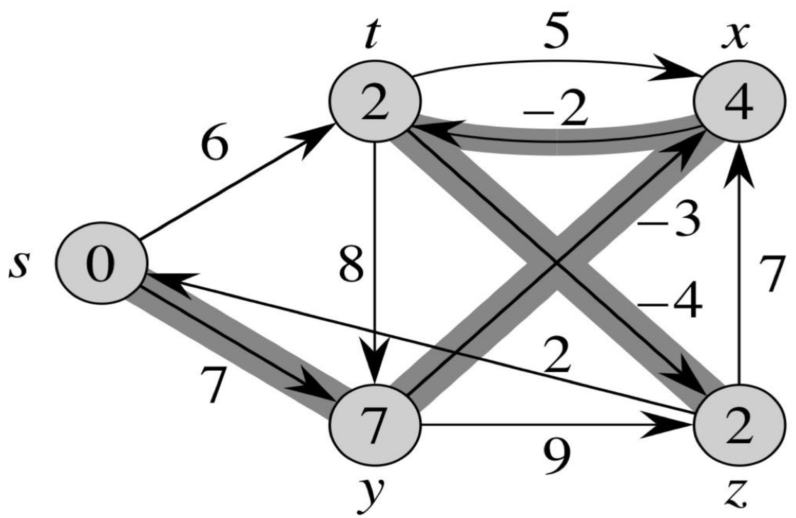


(t,x), (t,y), (t,z), (x,t), (y,x), (y,z), (z,x), (z,s), (s,t), (s,y)

vértice	s	t	x	y	z
d	0	2	4	7	2
π	NIL	x	y	s	t

Algoritmo de Bellman-Ford

BELLMAN-FORD (V,A, w, s)
1. INITIALIZE-SINGLE-SOURCE (V, A, s)
2. for i = 1 to |V| - 1 do
3.   for each edge (u, v) in A do
4.   RELAX (u, v, w)
5. for each edge (u, v) in A do
6.   if d[u] + w(u, v) < d[v] then
7.   return FALSE
8. return TRUE

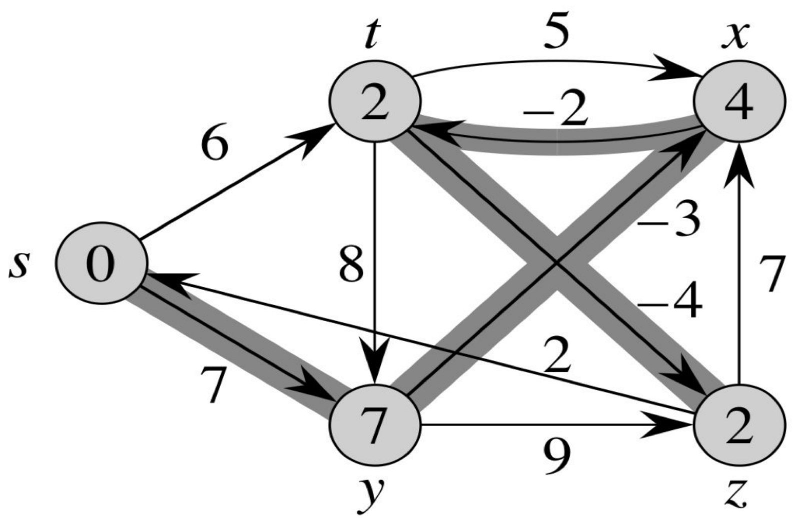


(t,x), (t,y), (t,z), (x,t), (y,x), (y,z), (z,x), (z,s), (s,t), (s,y)

vértice	s	t	x	y	z
d	0	2	4	7	2
π	NIL	x	y	s	t

Algoritmo de Bellman-Ford

```
BELLMAN-FORD (V,A, w, s)
1. INITIALIZE-SINGLE-SOURCE (V, A, s)
2. for  i = 1 to |V| - 1 do
3.      for each edge (u, v) in A do
4.           RELAX (u, v, w)
5. for each edge (u, v) in A do
6.      if d[u] + w(u, v) < d[v] then
7.           return FALSE
8. return TRUE
```



(t,x), (t,y), (t,z), (x,t), (y,x), (y,z), (z,x), (z,s), (s,t), (s,y)

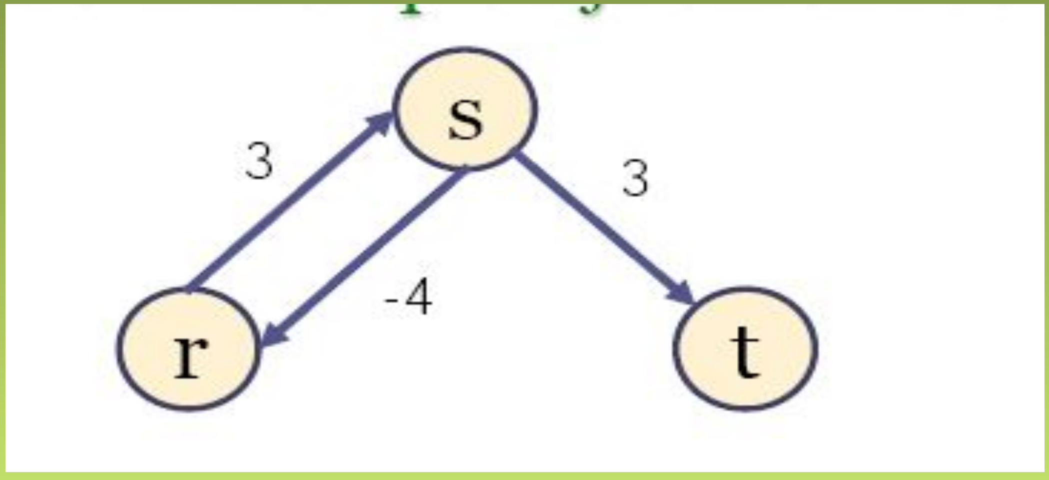
Se conseguir relaxar as arestas depois das V-1 iterações, é porque o grafo possui um ciclo de peso negativo!

vértice	s	t	x	y	z
d	0	2	4	7	2
π	NIL	x	y	s	t

O algoritmo retorna TRUE

O algoritmo de Bellman-Ford

- Aplicar o algoritmo de Bellman-Ford para o grafo a seguir, considerar a origem s.



- Podemos modificar o algoritmo para devolver quais vértices podem ser alcançados com custo -∞?

O algoritmo de Dijkstra

Algoritmo guloso que resolve o problema de caminhos mais curtos de única origem.

Os pesos das arestas são não negativos. Conseqüentemente não possui ciclos de peso negativo.

O tempo de execução é inferior ao algoritmo de Bellman-Ford.

O algoritmo de Dijkstra

Trabalha com dois conjuntos de vértices:

S: vértices cuja menor distância para a raiz já é conhecida (definitiva).

V-S: vértices em que a distância conhecida ainda é uma estimativa (provisória) .

Para isso, o algoritmo utiliza:

S: um conjunto de vértices cuja distância já é definitiva.

Q: uma fila de prioridade mínima de vértices com distância provisória

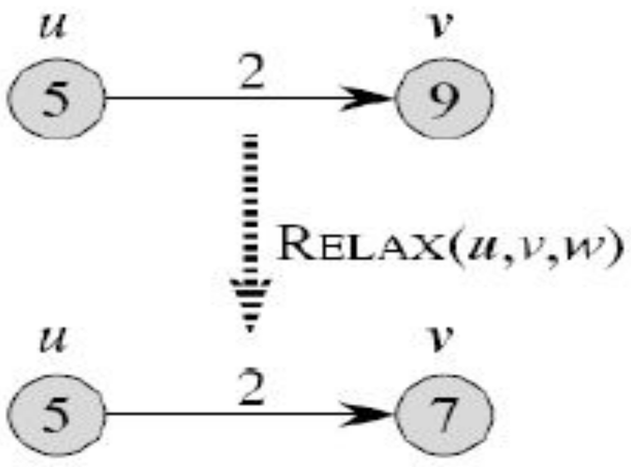
Algoritmo de Dijkstra: métodos usados

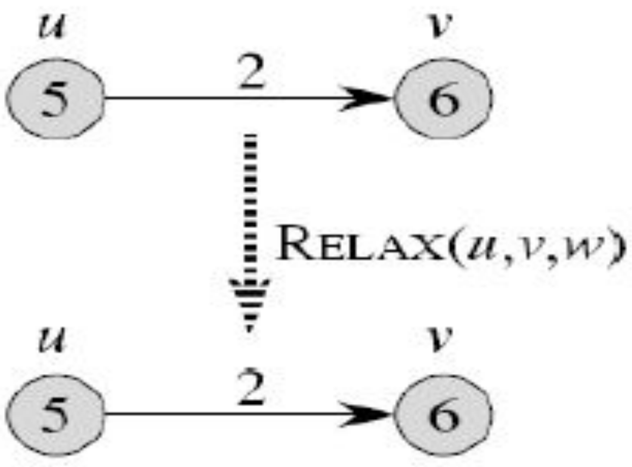
INITIALIZE-SINGLE-SOURCE(V,A,s)

for cada vértice $v \in V$
    $d[v] \leftarrow \infty$
    $\pi[v] \leftarrow$ NIL
$d[s] \leftarrow 0$

RELAX(u,v,w)

if $d[v] > d[u] + w(u,v)$
    then $d[v] \leftarrow d[u] + w(u,v)$
        $\pi[v] \leftarrow u$



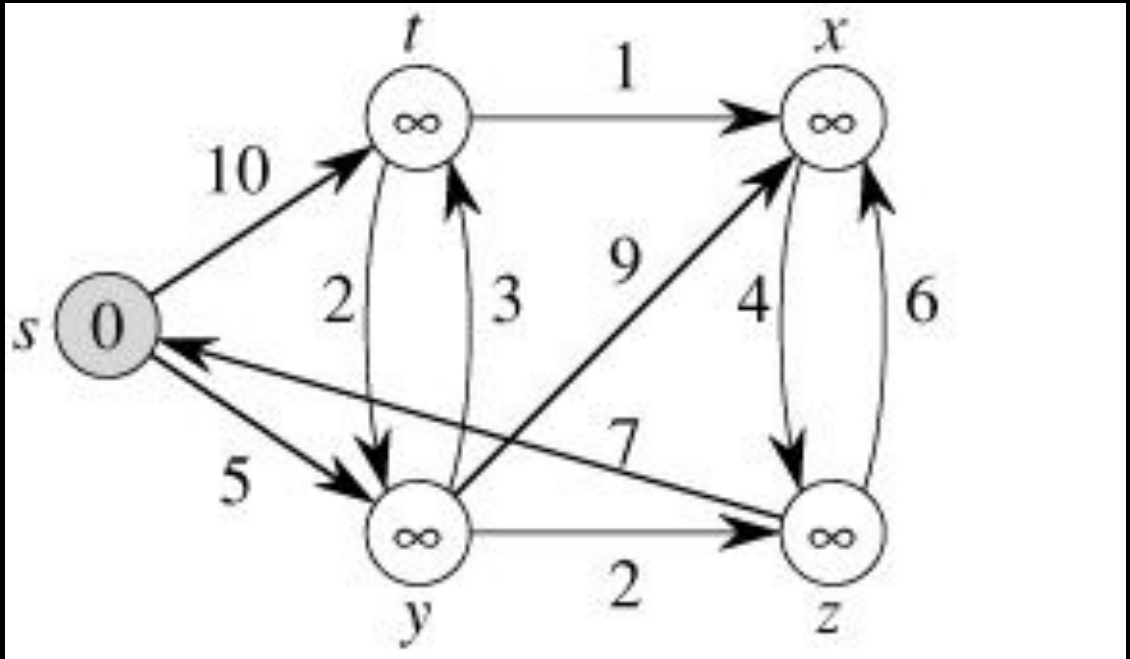


O algoritmo de Dijkstra

```
DIJKSTRA (V, A, w, s)
1. INITIALIZE SINGLE-SOURCE (V,A, s)
2. S ← { }
3. Q  ←  V
4. while Q  is not empty do
5.        u  ←  EXTRACT_MIN(Q)
6.        S  ←  S ∪ {u}
           // Relaxar cada vértice adjacente a u
7.         for each vertex v in Adj[u] do
8.                   RELAX (u, v, w)
```

O algoritmo de Dijkstra

```
DIJKSTRA (V, A, w, s)
1. INITIALIZE SINGLE-SOURCE (V,A, s)
2. S ← { }
3. Q  ←  V
4. while Q  is not empty do
5.          u  ←  EXTRACT_MIN(Q)
6.          S  ←  S  U  {u}
             // Relaxar cada vértice adjacente a u
7.           for each vertex v in Adj[u] do
8.                     RELAX (u, v, w)
```

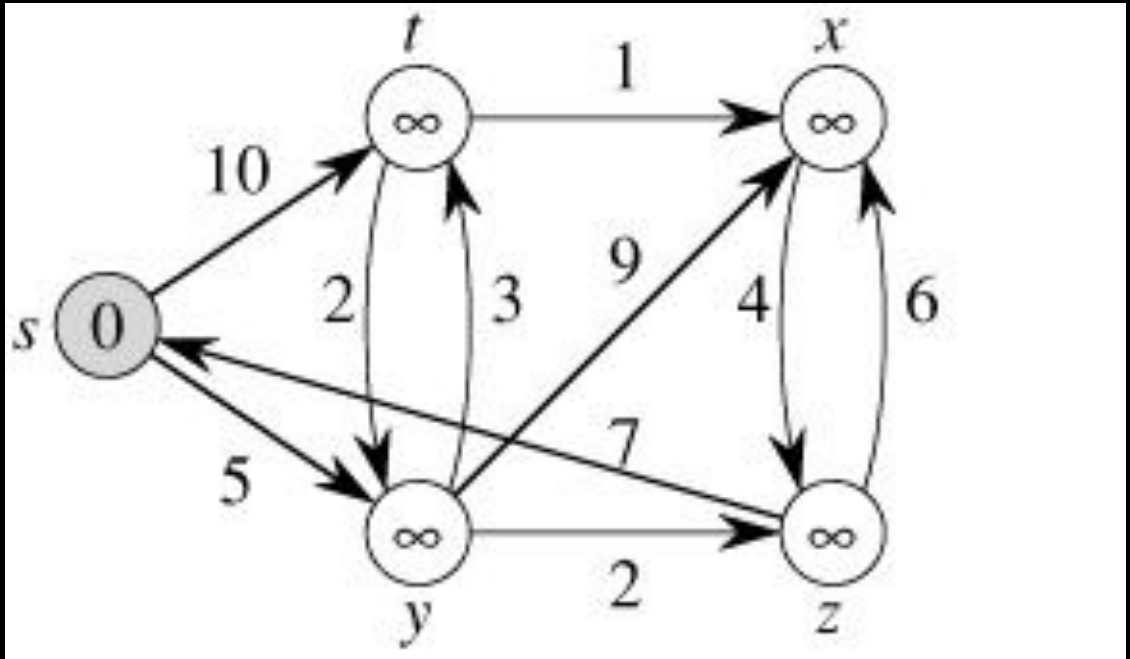


Fonte: Wikimedia Commons

VÉRTICE	s	t	x	y	z
d					
π					
Q					
S					

O algoritmo de Dijkstra

DIJKSTRA (V, A, w, s)
1. INITIALIZE SINGLE-SOURCE (V,A, s)
2. S ← { }
3. Q ← V
4. while Q is not empty do
5.   u ← EXTRACT_MIN(Q)
6.   S ← S ∪ {u}
   // Relaxar cada vértice adjacente a u
7.   for each vertex v in Adj[u] do
8.   RELAX (u, v, w)

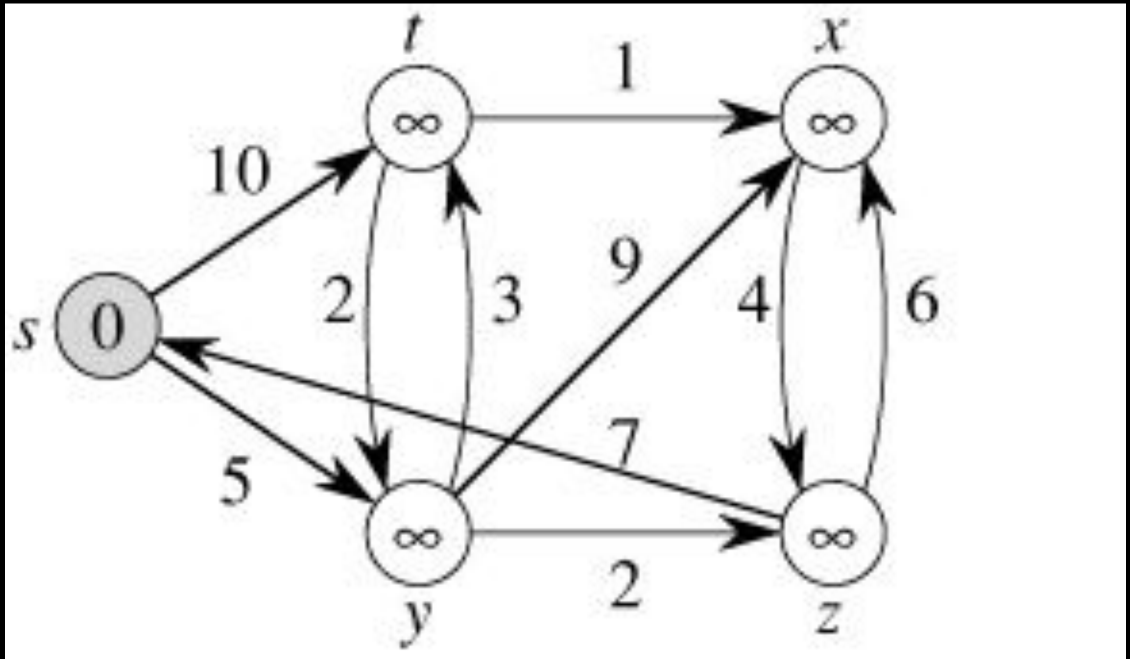


Fonte: Wikimedia Commons

VÉRTICE	s	t	x	y	z
d	0	∞	∞	∞	∞
π	NIL	NIL	NIL	NIL	NIL
Q					
S					

O algoritmo de Dijkstra

DIJKSTRA (V, A, w, s)
1. INITIALIZE SINGLE-SOURCE (V,A, s)
2. S ← { }
3. Q ← V
4. while Q is not empty do
5. u ← EXTRACT_MIN(Q)
6. S ← S U {u}
// Relaxar cada vértice adjacente a u
7. for each vertex v in Adj[u] do
8. RELAX (u, v, w)

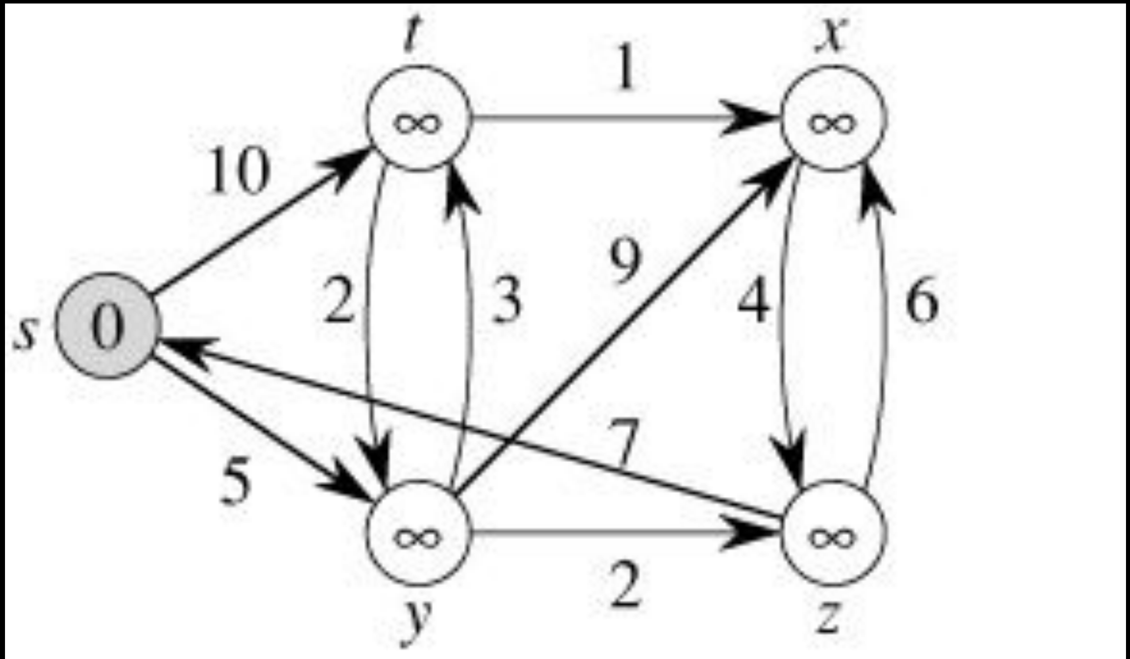


Fonte: Wikimedia Commons

VÉRTICE	s	t	x	y	z
d	0	∞	∞	∞	∞
π	NIL	NIL	NIL	NIL	NIL
Q	✔	✔	✔	✔	✔
S					

O algoritmo de Dijkstra

DIJKSTRA (V, A, w, s)
1. INITIALIZE SINGLE-SOURCE (V,A, s)
2. S ← { }
3. Q ← V
4. while Q is not empty do
5. u ← EXTRACT_MIN(Q)
6. S ← S U {u}
// Relaxar cada vértice adjacente a u
7. for each vertex v in Adj[u] do
8. RELAX (u, v, w)

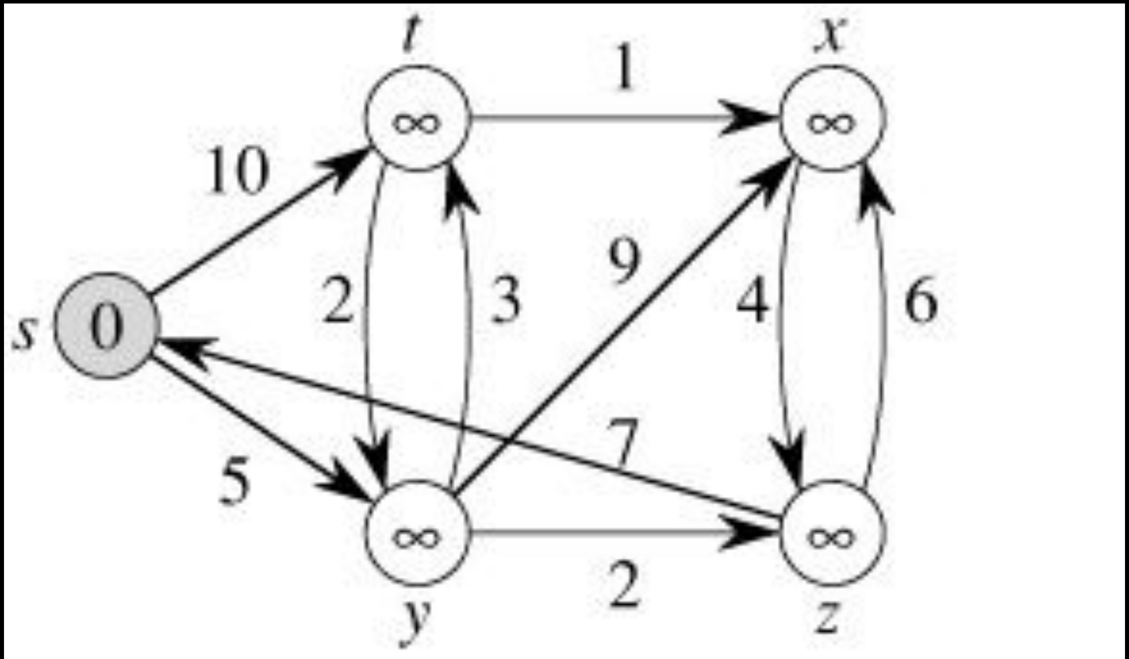


Fonte: Wikimedia Commons

VÉRTICE	s	t	x	y	z
d	0	∞	∞	∞	∞
π	NIL	NIL	NIL	NIL	NIL
Q		✔	✔	✔	✔
S	✔				

O algoritmo de Dijkstra

DIJKSTRA (V, A, w, s)
1. INITIALIZE SINGLE-SOURCE (V,A, s)
2. S ← { }
3. Q ← V
4. while Q is not empty do
5. u ← EXTRACT_MIN(Q)
6. S ← S U {u}
// Relaxar cada vértice adjacente a u
7. for each vertex v in Adj[u] do
8. RELAX (u, v, w)

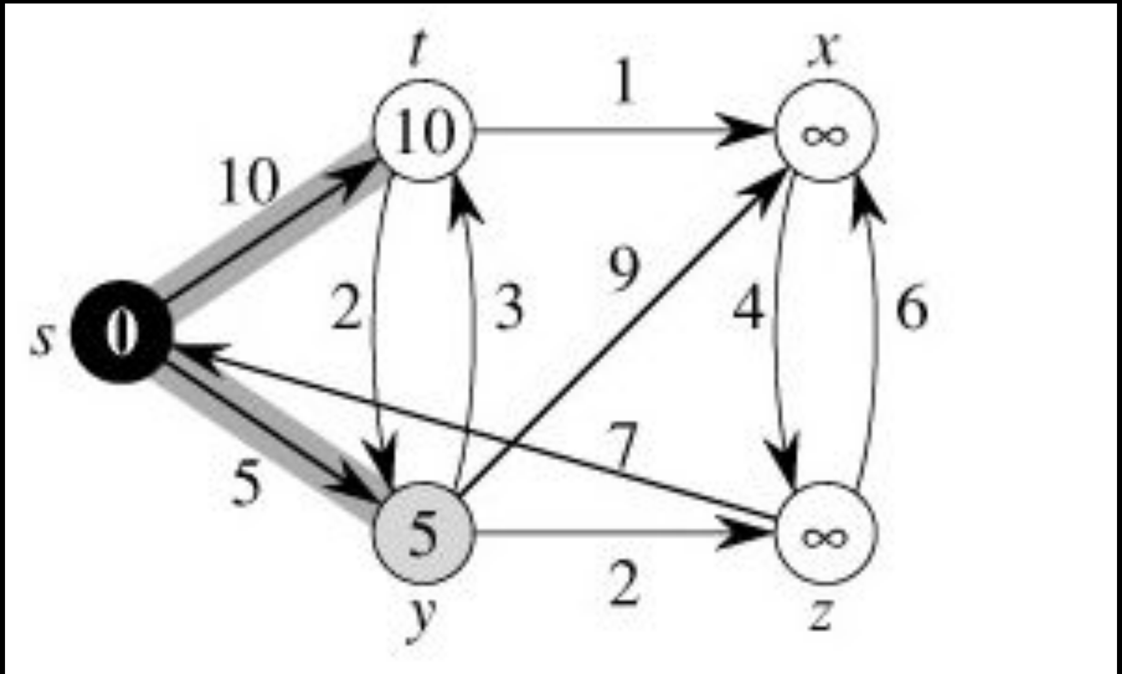


VÉRTICE	s	t
d	0	∞
π	NIL	NIL
Q		✔
S	✔	

Será que podemos **melhorar o caminho mais curto para t** encontrado até agora pela passagem através de s?
Será que podemos **melhorar o caminho mais curto para y** encontrado até agora pela passagem através de s?

O algoritmo de Dijkstra

DIJKSTRA (V, A, w, s)
1. INITIALIZE SINGLE-SOURCE (V,A, s)
2. S ← { }
3. Q ← V
4. while Q is not empty do
5. u ← EXTRACT_MIN(Q)
6. S ← S ∪ {u}
// Relaxar cada vértice adjacente a u
7. for each vertex v in Adj[u] do
8. RELAX (u, v, w)

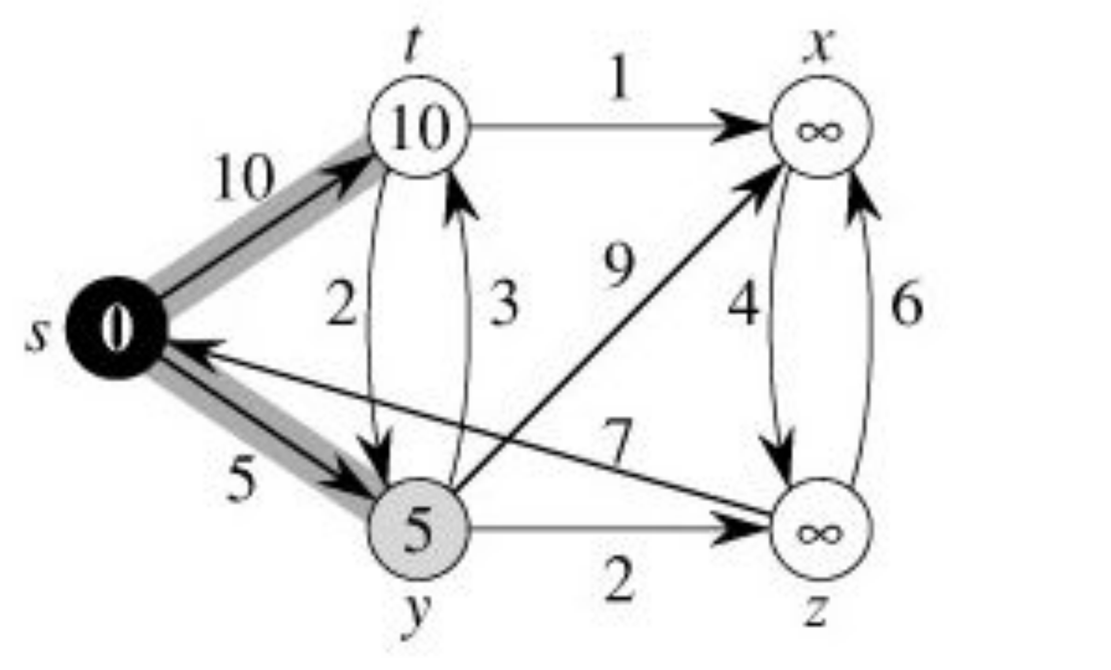


Fonte: Wikimedia Commons

VÉRTICE	s	t	x	y	z
d	0	10	∞	5	∞
π	NIL	s	NIL	s	NIL
Q		✔	✔	✔	✔
S	✔				

O algoritmo de Dijkstra

DIJKSTRA (V, A, w, s)
1. INITIALIZE SINGLE-SOURCE (V,A, s)
2. S ← { }
3. Q ← V
4. while Q is not empty do
5. u ← EXTRACT_MIN(Q)
6. S ← S U {u}
// Relaxar cada vértice adjacente a u
7. for each vertex v in Adj[u] do
8. RELAX (u, v, w)

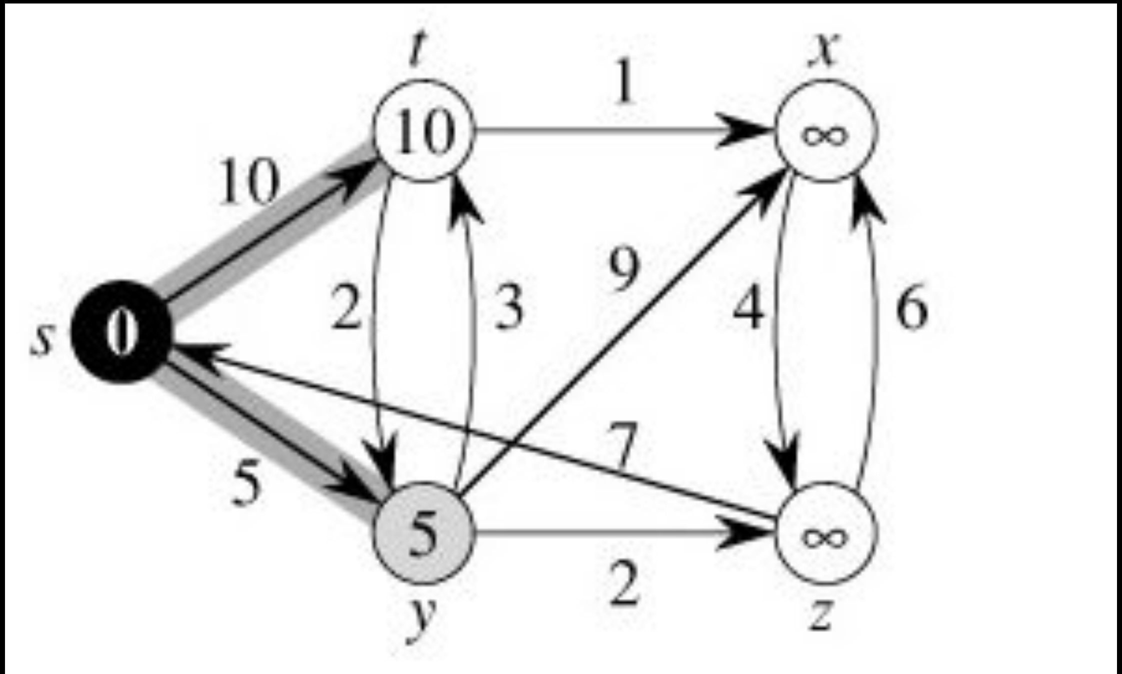


Fonte: Wikimedia Commons

VÉRTICE	s	t	x	y	z
d	0	10	∞	5	∞
π	NIL	s	NIL	s	NIL
Q		✔	✔		✔
S	✔			✔	

O algoritmo de Dijkstra

DIJKSTRA (V, A, w, s)
1. INITIALIZE SINGLE-SOURCE (V,A, s)
2. S ← { }
3. Q ← V
4. while Q is not empty do
5. u ← EXTRACT_MIN(Q)
6. S ← S U {u}
// Relaxar cada vértice adjacente a u
7. for each vertex v in Adj[u] do
8. RELAX (u, v, w)

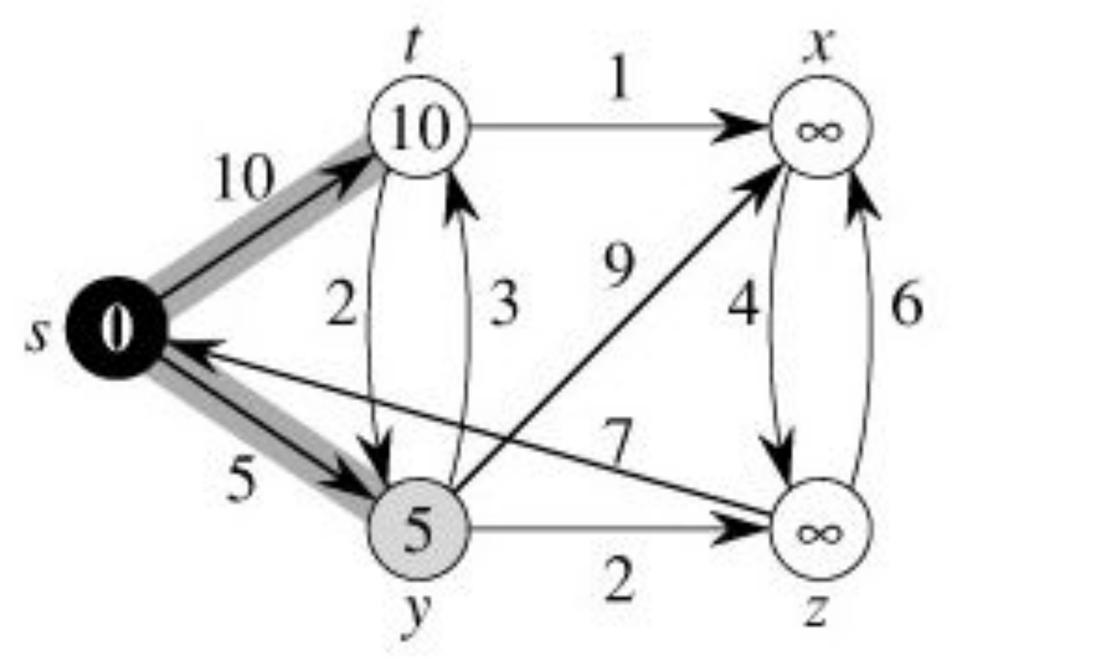


A distância definitiva (mínima) até o vértice y é 5.

VÉRTICE	s	t	x	y	z
d	0	10	∞	5	∞
π	NIL	s	NIL	s	NIL
Q		✔	✔		✔
S	✔			✔	

O algoritmo de Dijkstra

DIJKSTRA (V, A, w, s)
1. INITIALIZE SINGLE-SOURCE (V,A, s)
2. S ← { }
3. Q ← V
4. while Q is not empty do
5. u ← EXTRACT_MIN(Q)
6. S ← S U {u}
// Relaxar cada vértice adjacente a u
7. for each vertex v in Adj[u] do
8. RELAX (u, v, w)

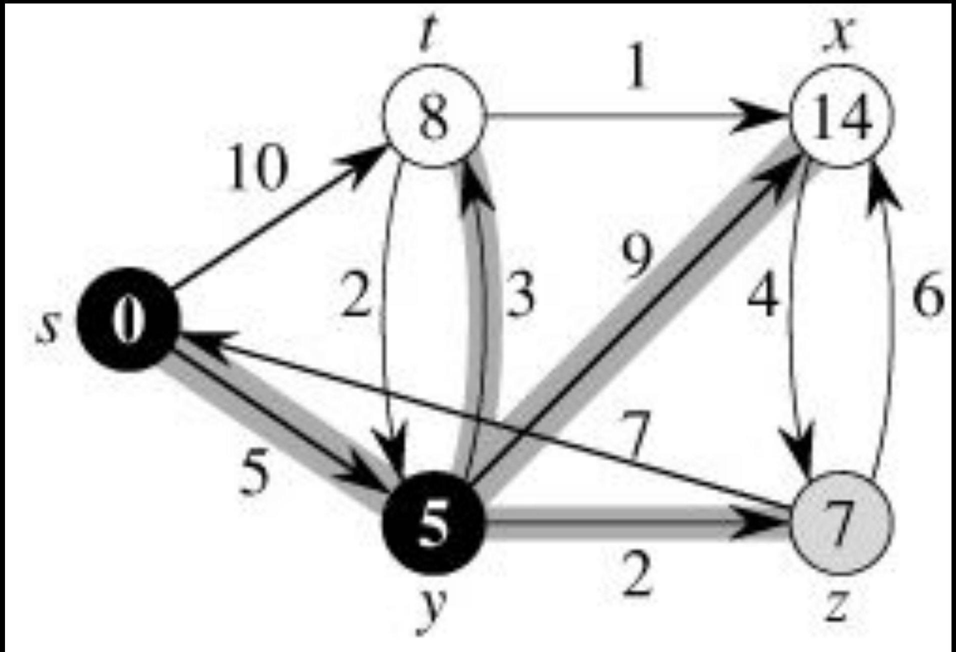


VÉRTICE	s	t
d	0	10
π	NIL	s
Q		✔
S	✔	

Será que podemos **melhorar o caminho mais curto para t** encontrado até agora pela passagem através de y?
Será que podemos **melhorar o caminho mais curto para x** encontrado até agora pela passagem através de y?
Será que podemos **melhorar o caminho mais curto para z** encontrado até agora pela passagem através de y?

O algoritmo de Dijkstra

DIJKSTRA (V, A, w, s)
1. INITIALIZE SINGLE-SOURCE (V,A, s)
2. S ← { }
3. Q ← V
4. while Q is not empty do
5. u ← EXTRACT_MIN(Q)
6. S ← S U {u}
// Relaxar cada vértice adjacente a u
7. for each vertex v in Adj[u] do
8. RELAX (u, v, w)

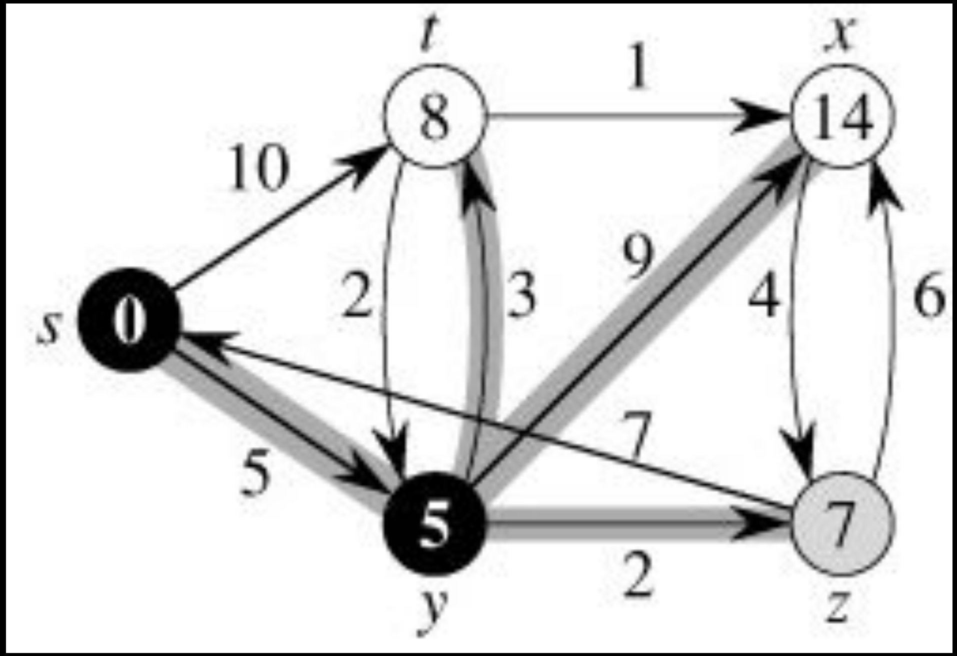


Fonte: Wikimedia Commons

VÉRTICE	s	t	x	y	z
d	0	8	14	5	7
π	NIL	y	y	s	y
Q		✔	✔		✔
S	✔			✔	

O algoritmo de Dijkstra

DIJKSTRA (V, A, w, s)
1. INITIALIZE SINGLE-SOURCE (V,A, s)
2. S ← { }
3. Q ← V
4. while Q is not empty do
5. u ← EXTRACT_MIN(Q)
6. S ← S ∪ {u}
   // Relaxar cada vértice adjacente a u
7. for each vertex v in Adj[u] do
8. RELAX (u, v, w)

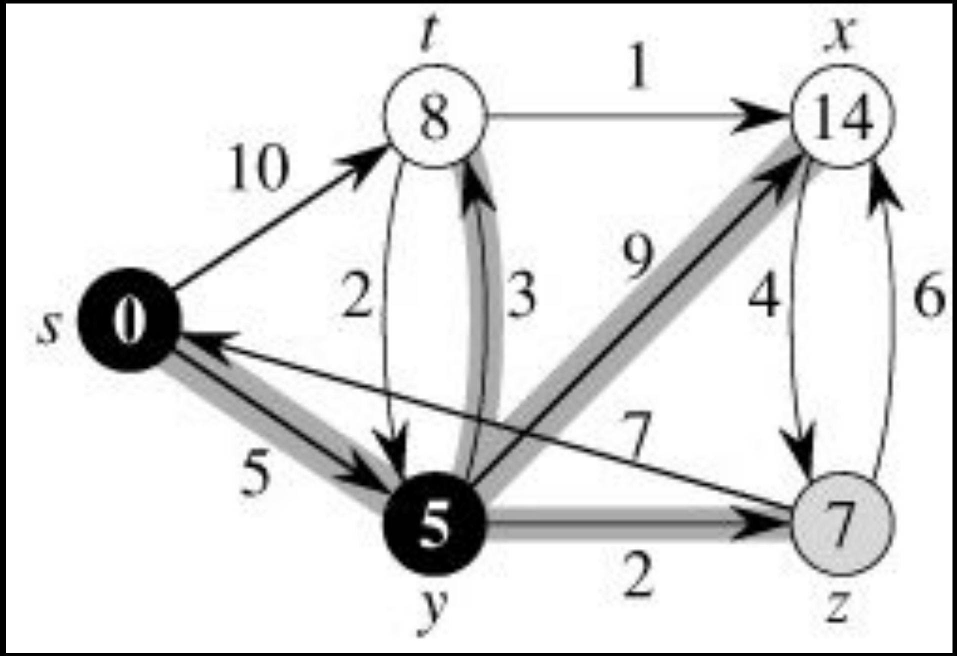


Fonte: Wikimedia Commons

VÉRTICE	s	t	x	y	z
d	0	8	14	5	7
π	NIL	y	y	s	y
Q		✔	✔		
S	✔			✔	✔

O algoritmo de Dijkstra

DIJKSTRA (V, A, w, s)
1. INITIALIZE SINGLE-SOURCE (V,A, s)
2. S ← { }
3. Q ← V
4. while Q is not empty do
5. u ← EXTRACT_MIN(Q)
6. S ← S ∪ {u}
// Relaxar cada vértice adjacente a u
7. for each vertex v in Adj[u] do
8. RELAX (u, v, w)

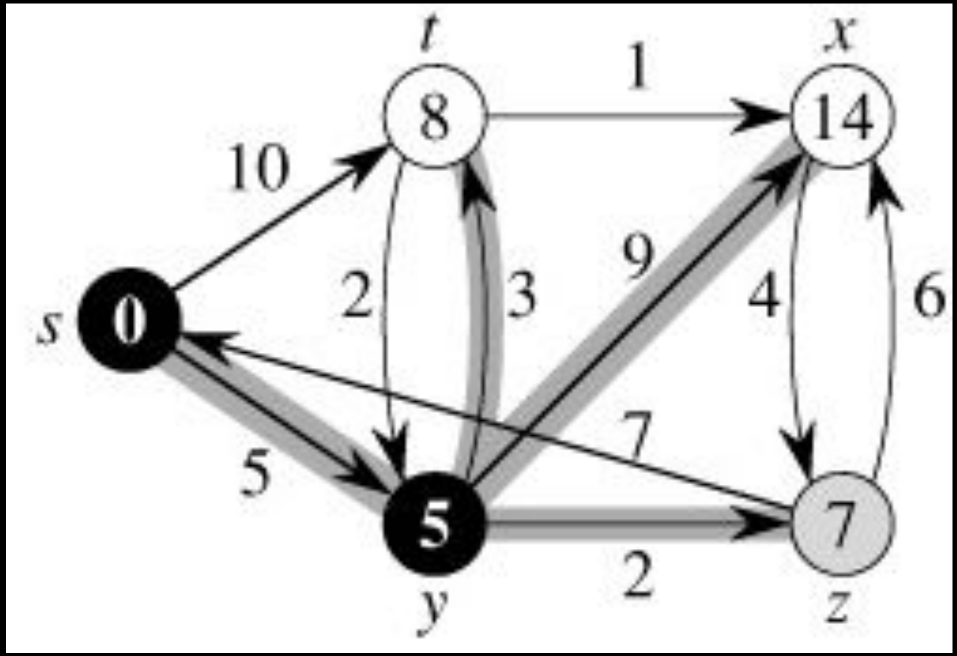


A distância definitiva (mínima) até o vértice z é 7.

VÉRTICE	s	t	x	y	z
d	0	8	14	5	7
π	NIL	y	y	s	y
Q		✔	✔		
S	✔			✔	✔

O algoritmo de Dijkstra

DIJKSTRA (V, A, w, s)
1. INITIALIZE SINGLE-SOURCE (V,A, s)
2. S ← { }
3. Q ← V
4. while Q is not empty do
5. u ← EXTRACT_MIN(Q)
6. S ← S ∪ {u}
   // Relaxar cada vértice adjacente a u
7. for each vertex v in Adj[u] do
8. RELAX (u, v, w)

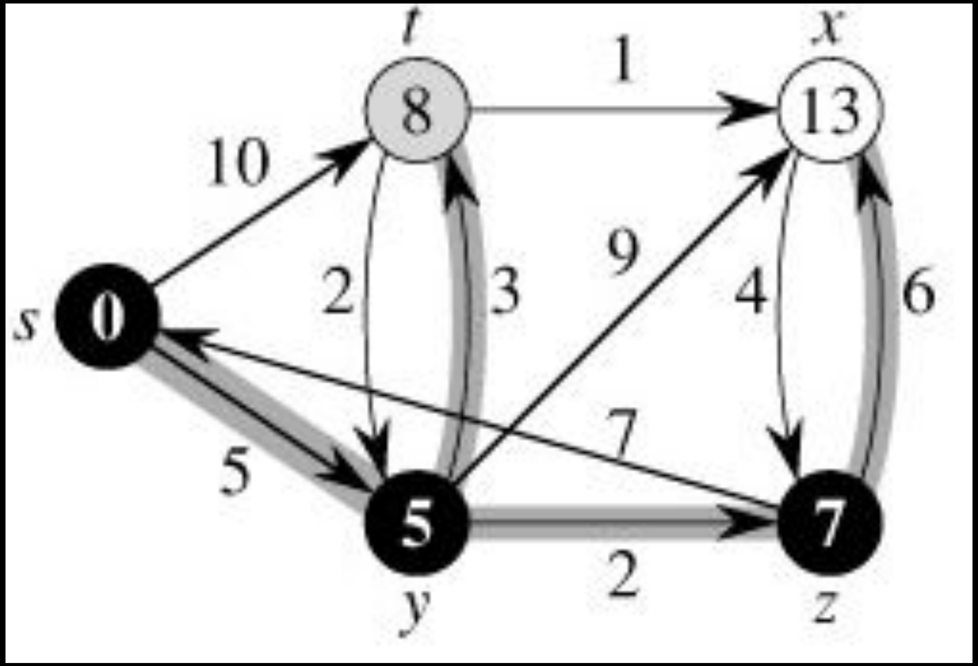


Será que podemos **melhorar o caminho mais curto para x** encontrado até agora pela passagem através de z?

VÉRTICE	s	t			
d	0	8			
π	NIL	y	y	s	y
Q		✔	✔		
S	✔			✔	✔

O algoritmo de Dijkstra

DIJKSTRA (V, A, w, s)
1. INITIALIZE SINGLE-SOURCE (V,A, s)
2. S ← { }
3. Q ← V
4. while Q is not empty do
5. u ← EXTRACT_MIN(Q)
6. S ← S U {u}
// Relaxar cada vértice adjacente a u
7. for each vertex v in Adj[u] do
8. RELAX (u, v, w)

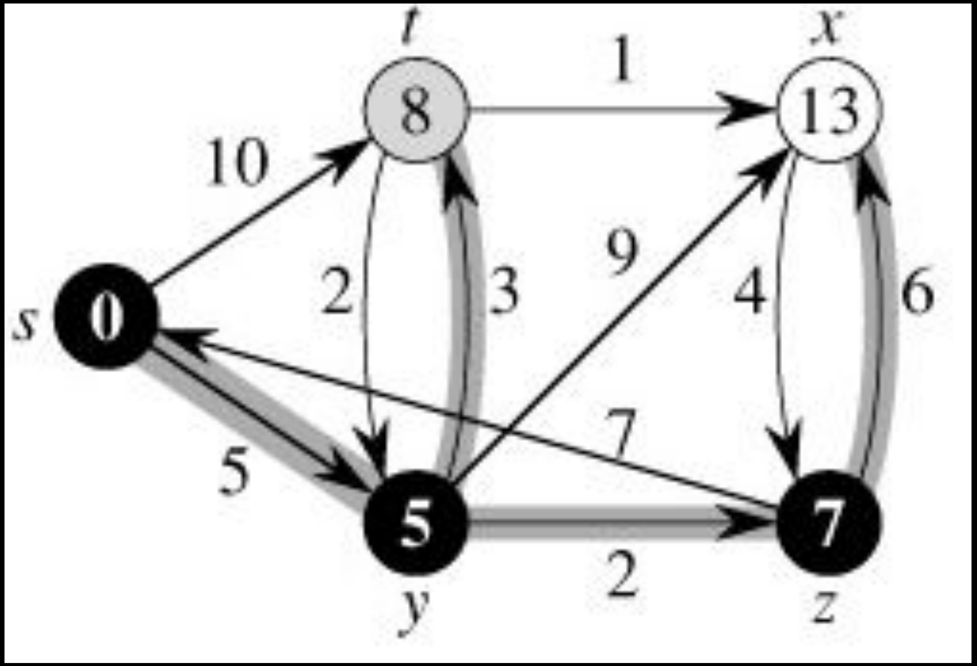


Fonte: Wikimedia Commons

VÉRTICE	s	t	x	y	z
d	0	8	13	5	7
π	NIL	y	z	s	y
Q		✔	✔		
S	✔			✔	✔

O algoritmo de Dijkstra

DIJKSTRA (V, A, w, s)
1. INITIALIZE SINGLE-SOURCE (V,A, s)
2. S ← { }
3. Q ← V
4. while Q is not empty do
5. u ← EXTRACT_MIN(Q)
6. S ← S U {u}
// Relaxar cada vértice adjacente a u
7. for each vertex v in Adj[u] do
8. RELAX (u, v, w)

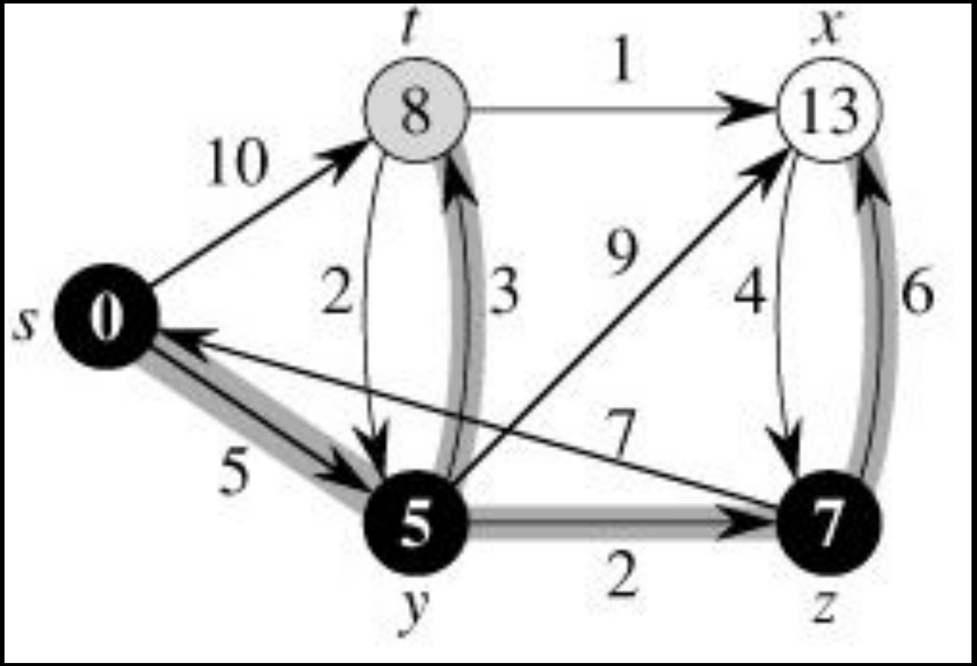


Fonte: Wikimedia Commons

VÉRTICE	s	t	x	y	z
d	0	8	13	5	7
π	NIL	y	z	s	y
Q			✔		
S	✔	✔		✔	✔

O algoritmo de Dijkstra

```
DIJKSTRA (V, A, w, s)
INITIALIZE SINGLE-SOURCE (V,A, s)
S ← { }
Q ← V
while Q is not empty do
        u ← EXTRACT_MIN(Q)
        S ← S U {u}
        // Relaxar cada vértice adjacente a u
        for each vertex v in Adj[u] do
                RELAX (u, v, w)
```

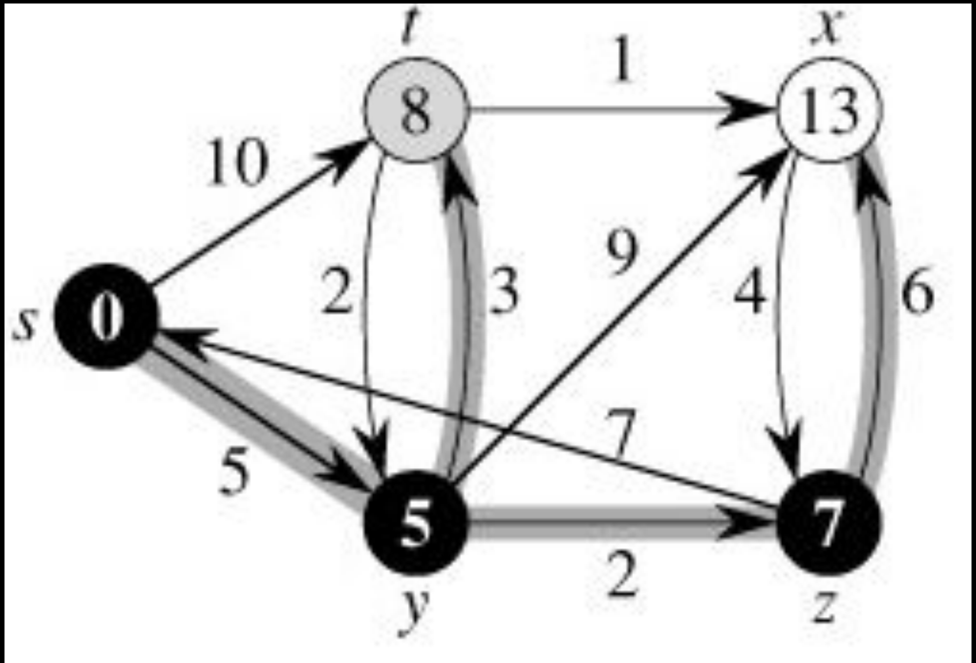


A distância definitiva (mínima) até o vértice t é 8.

VÉRTICE	s	t	x	y	z
d	0	8	13	5	7
π	NIL	y	z	s	y
Q			✔		
S	✔	✔		✔	✔

O algoritmo de Dijkstra

DIJKSTRA (V, A, w, s)
1. INITIALIZE SINGLE-SOURCE (V,A, s)
2. S ← { }
3. Q ← V
4. while Q is not empty do
5. u ← EXTRACT_MIN(Q)
6. S ← S ∪ {u}
// Relaxar cada vértice adjacente a u
7. for each vertex v in Adj[u] do
8. RELAX (u, v, w)

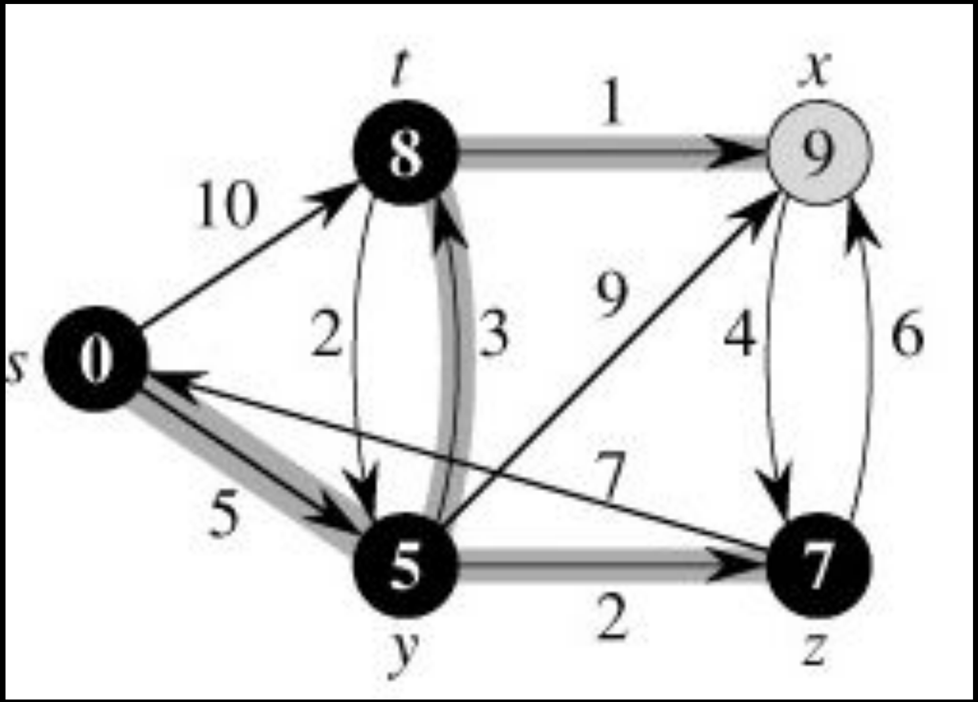


Será que podemos **melhorar o caminho mais curto para x** encontrado até agora pela passagem através de t?

VÉRTICE	s	t			
d	0	8	[illegible]	[illegible]	[illegible]
π	NIL	y	z	s	y
Q			✔		
S	✔	✔		✔	✔

O algoritmo de Dijkstra

```
DIJKSTRA (V, A, w, s)
1. INITIALIZE SINGLE-SOURCE (V,A, s)
2. S ← { }
3. Q  ←  V
4. while Q  is not empty do
5.         u  ←  EXTRACT_MIN(Q)
6.         S  ←  S  U  {u}
            // Relaxar cada vértice adjacente a u
7.          for each vertex v in Adj[u] do
8.                   RELAX (u, v, w)
```

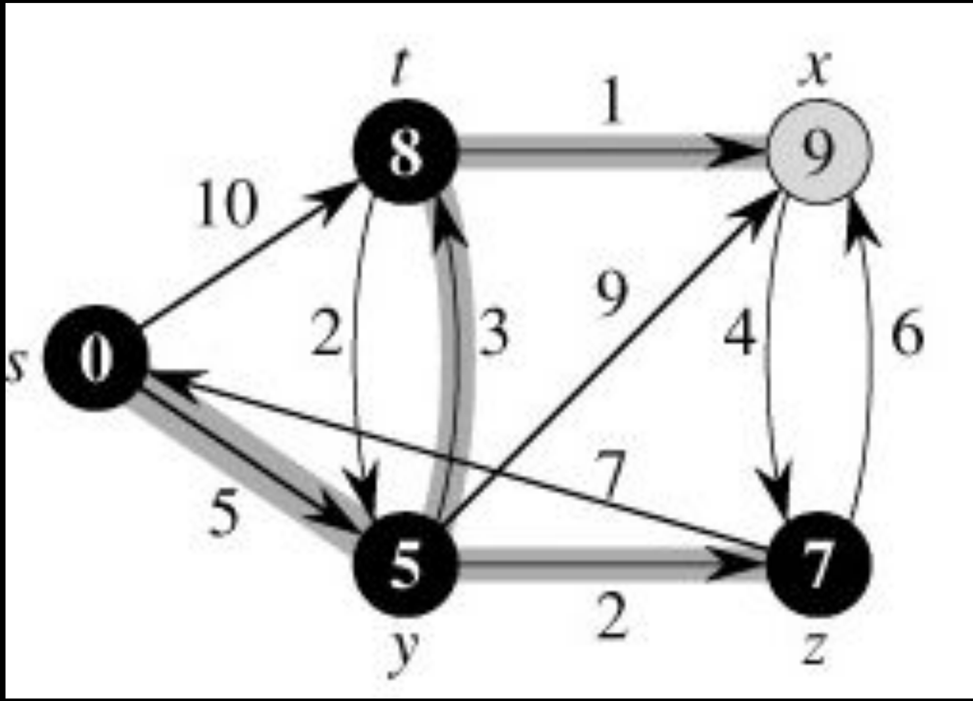


Fonte: Wikimedia Commons

VÉRTICE	s	t	x	y	z
d	0	8	9	5	7
π	NIL	y	t	s	y
Q			✔		
S	✔	✔		✔	✔

O algoritmo de Dijkstra

DIJKSTRA (V, A, w, s)
1. INITIALIZE SINGLE-SOURCE (V,A, s)
2. S ← { }
3. Q ← V
4. while Q is not empty do
5. u ← EXTRACT_MIN(Q)
6. S ← S U {u}
// Relaxar cada vértice adjacente a u
7. for each vertex v in Adj[u] do
8. RELAX (u, v, w)

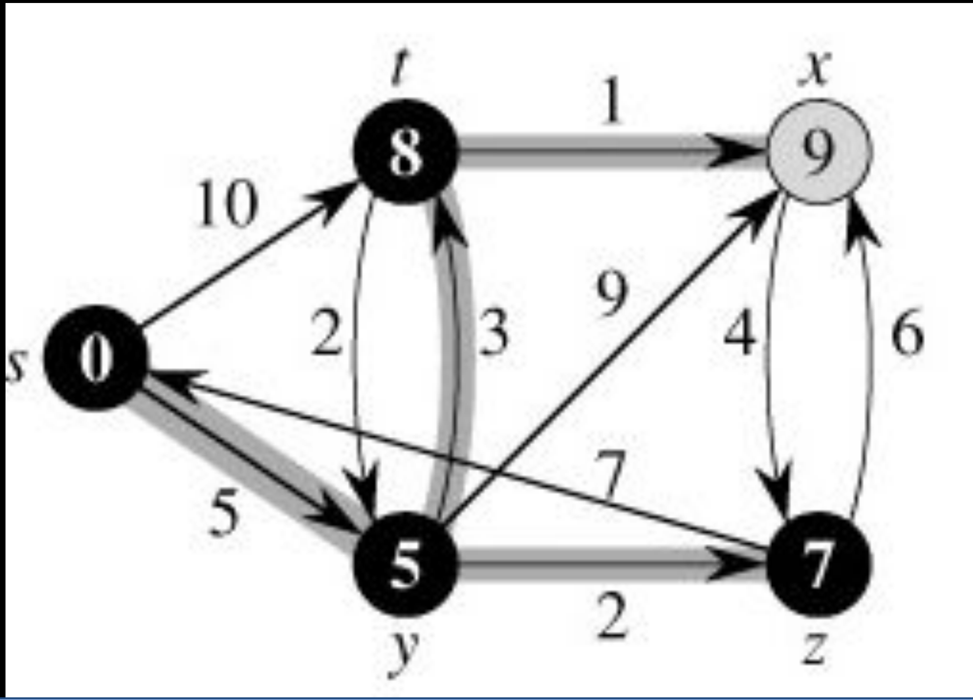


Fonte: Wikimedia Commons

VÉRTICE	s	t	x	y	z
d	0	8	9	5	7
π	NIL	y	t	s	y
Q					
S	✔	✔	✔	✔	✔

O algoritmo de Dijkstra

DIJKSTRA (V, A, w, s)
1. INITIALIZE SINGLE-SOURCE (V,A, s)
2. S ← { }
3. Q ← V
4. while Q is not empty do
5. u ← EXTRACT_MIN(Q)
6. S ← S U {u}
   // Relaxar cada vértice adjacente a u
7. for each vertex v in Adj[u] do
8. RELAX (u, v, w)

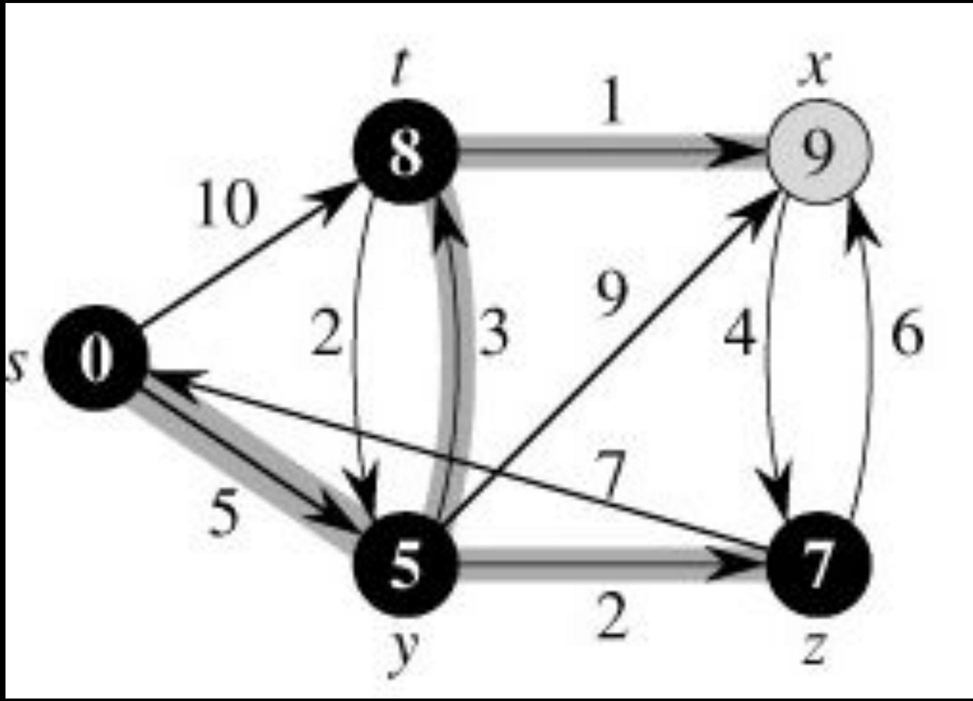


A distância definitiva (mínima) até o vértice x é 9.

VÉRTICE	s	t	x	y	z
d	0	8	9	5	7
π	NIL	y	t	s	y
Q					
S	✔	✔	✔	✔	✔

O algoritmo de Dijkstra

DIJKSTRA (V, A, w, s)
1. INITIALIZE SINGLE-SOURCE (V,A, s)
2. S ← { }
3. Q ← V
4. while Q is not empty do
5. u ← EXTRACT_MIN(Q)
6. S ← S U {u}
// Relaxar cada vértice adjacente a u
7. for each vertex v in Adj[u] do
8. RELAX (u, v, w)

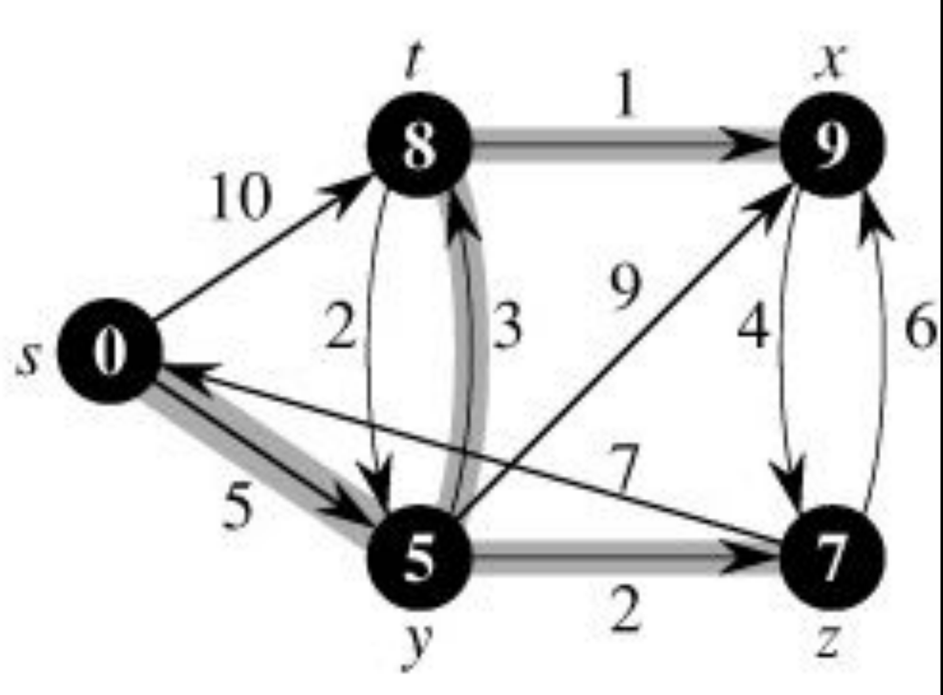


Fonte: Wikimedia Commons

VÉRTICE	s	t	x	y	z
d	0	8	9	5	7
π	NIL	y	t	s	y
Q					
S	✔	✔	✔	✔	✔

O algoritmo de Dijkstra

DIJKSTRA (V, A, w, s)
1. INITIALIZE SINGLE-SOURCE (V,A, s)
2. S ← { }
3. Q ← V
4. while Q is not empty do
5. u ← EXTRACT_MIN(Q)
6. S ← S U {u}
// Relaxar cada vértice adjacente a u
7. for each vertex v in Adj[u] do
8. RELAX (u, v, w)



Fonte: Wikimedia Commons

VÉRTICE	s	t	x	y	z
d	0	8	9	5	7
π	NIL	y	t	s	y
Q					
S	✔	✔	✔	✔	✔

Problemas:

- O algoritmo de Dijkstra pode ser optimizado para encontrar o menor caminho entre um par de vértices?
- Qual estrutura de dados pode ser usada para Q?
- Como encontrar os caminhos mais curtos de destino único?

AULA 4

Problema do caminho mais curto de uma única origem em grafos

Karina Valdivia Delgado