Grafos: caminhos mínimos

SCE-183 Algoritmos e Estruturas de Dados 2

Thiago A. S. Pardo
Maria Cristina
Gustavo Batista

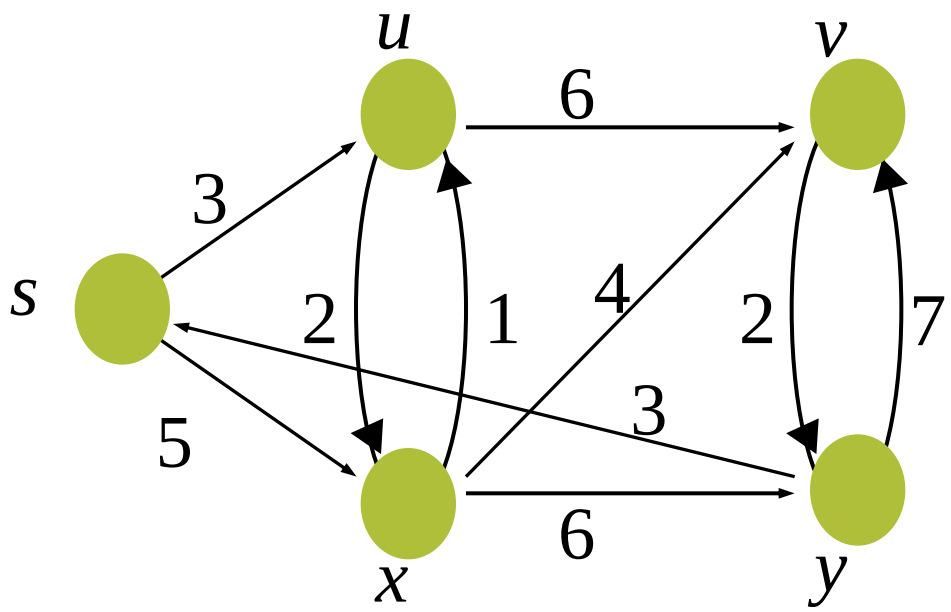
O problema do menor caminho

- Um motorista deseja encontrar o caminho mais curto possível entre duas cidades do Brasil
- Caso ele receba um mapa das estradas de rodagem do Brasil, no qual a distância entre cada par adjacente de cidades está exposta, como poderíamos determinar uma rota mais curta entre as cidades desejadas?
- Uma maneira possível é enumerar todas as rotas possíveis que levam de uma cidade à outra, e então selecionar a menor

Caminhos mínimos

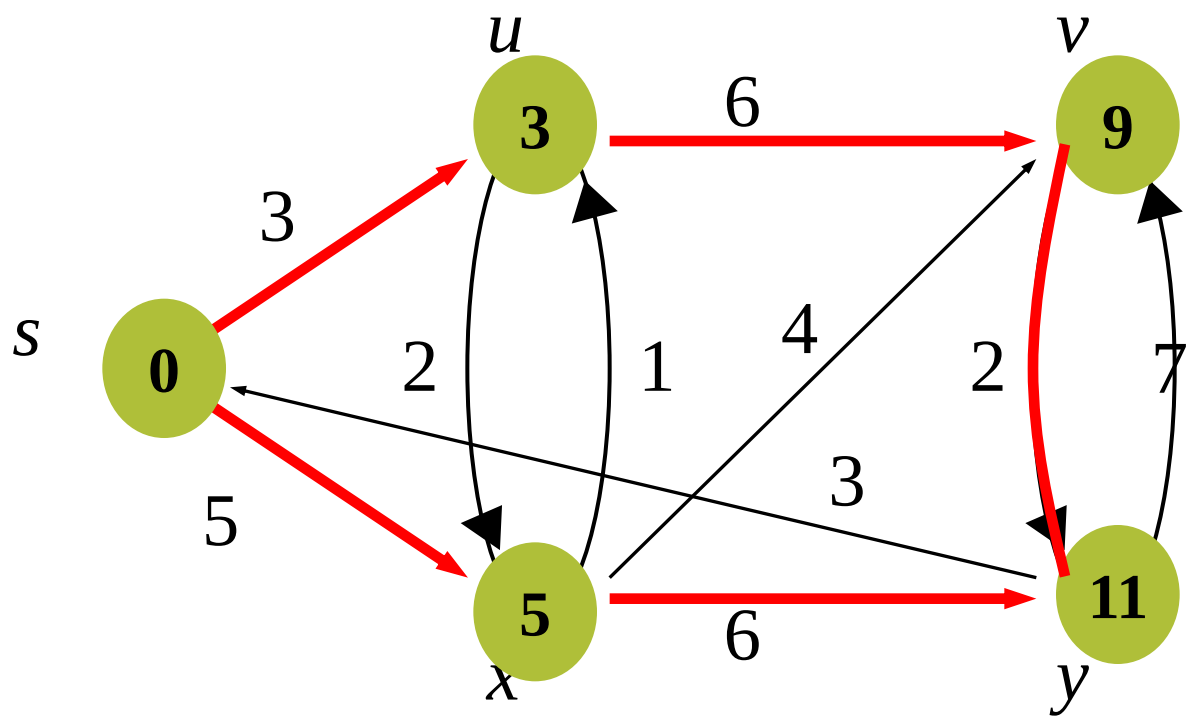
- O problema do caminho mínimo consiste em determinar um menor caminho entre um vértice de origem u e um vértice de destino v

- Qual o menor caminho entre s e y?



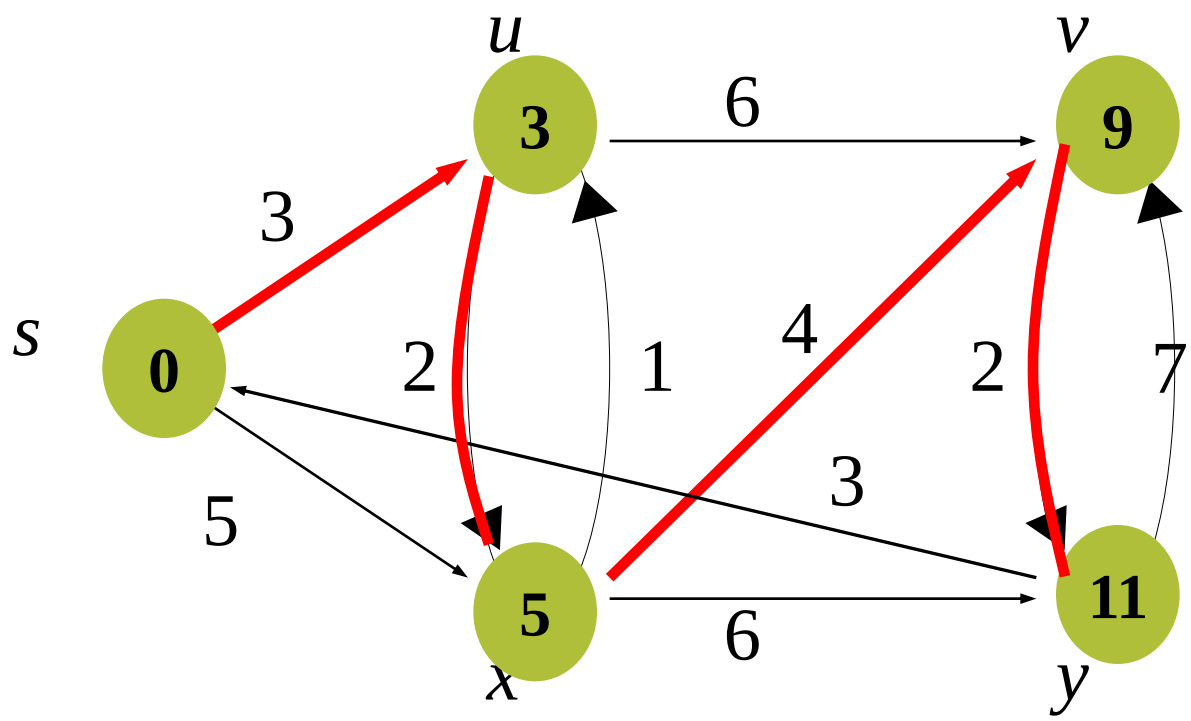
Caminhos mínimos

- Possíveis caminhos



Caminhos mínimos

- Possíveis caminhos



Caminho mínimo

- Duas abordagens para caminho mínimo
  - Se grafo não valorado (assume-se que cada aresta tem peso 1), busca em largura é uma boa opção
  - Se grafo valorado, outros algoritmos são necessários

Caminho mínimo

- Grafo dirigido G(V,E) com função peso *w*: E$\rightarrow\Re$ que mapeia as arestas em pesos
- Peso (custo) do caminho p = <v_0, v_1, ..., v_k>

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

- Caminho de menor peso entre u e v:

$$\delta(u,v) = \begin{cases} \min\{w(p) : u \overset{p}{\Rightarrow} v\} \; se \exists \; rota \; de \; u \;\; p/v \\ \infty \; cc \end{cases}$$

Caminho mínimo

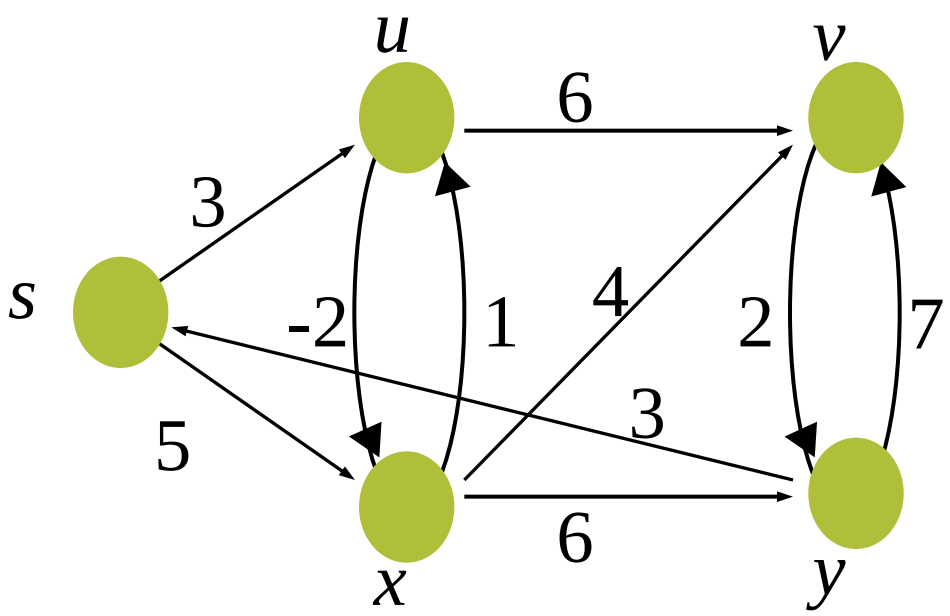
- Menor caminho entre os vértices *u* e *v* definido como qualquer rota p com um peso

$$w(p) = \delta(u,v)$$

- Atenção especial com ciclos e pesos negativos

Caminho mínimo

- Qual o menor caminho entre s e v?

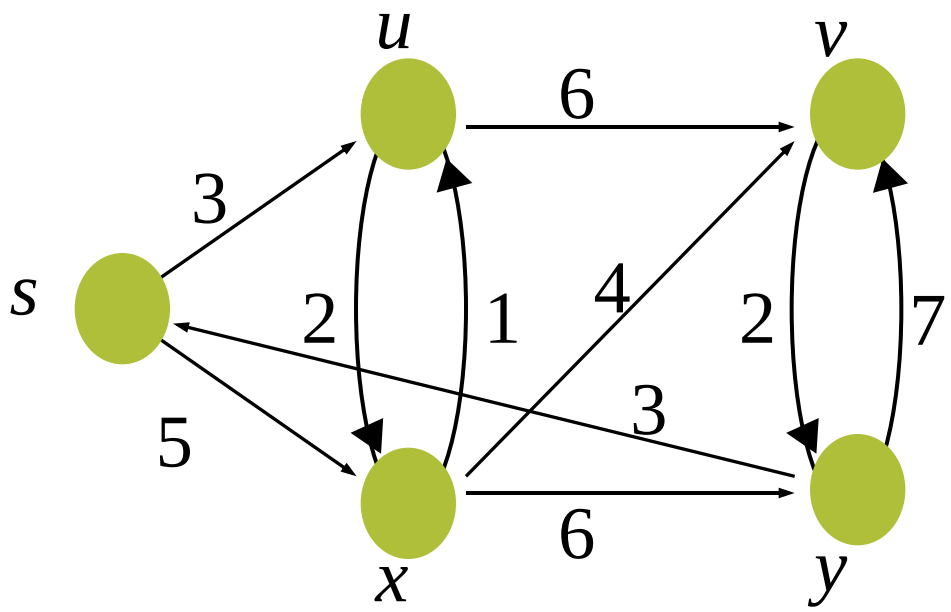


Camino mínimo

- Se há um ciclo positivo no caminho, ele não faz parte do caminho mínimo
  - Por quê?

Caminhos mínimos

- Conceitos
  - Parte-se do vértice s, associando-se a cada vértice um número d(v) indicando a menor distância entre s e v
  - Quando chegamos ao vértice v, na figura abaixo, d(v) será min(d(u)+6, d(x)+4, d(y)+7)



Caminhos mínimos

- Conceitos
  - *Relaxamento* de arestas
    - Faz-se uma estimativa pessimista para o caminho mínimo até cada vértice: d(v)=∞
    - O processo de relaxar uma aresta consiste em verificar se é possível melhorar esta estimativa passando-se pelo vértice u

Caminhos mínimos

- Conceitos
  - *Relaxamento* de arestas
    - Faz-se uma estimativa pessimista para o caminho mínimo até cada vértice: d(v)=∞
    - O processo de relaxar uma aresta consiste em verificar se é possível melhorar esta estimativa passando-se pelo vértice u

Caminhos mínimos

- Sub-rotina para relaxamento de arestas

```
relax(u, v, w)
    início
    se d[v] > d[u]+w(u,v) então
            d[v] = d[u]+w(u,v)
            antecessor[v]=u
    fim
```

Caminho mínimo

- Várias possibilidades de caminhos
  - Caminhos mais curtos de origem única
  - Caminhos mais curtos de destino único
  - Caminho mais curto de par único
  - Caminhos mais curtos de todos os pares

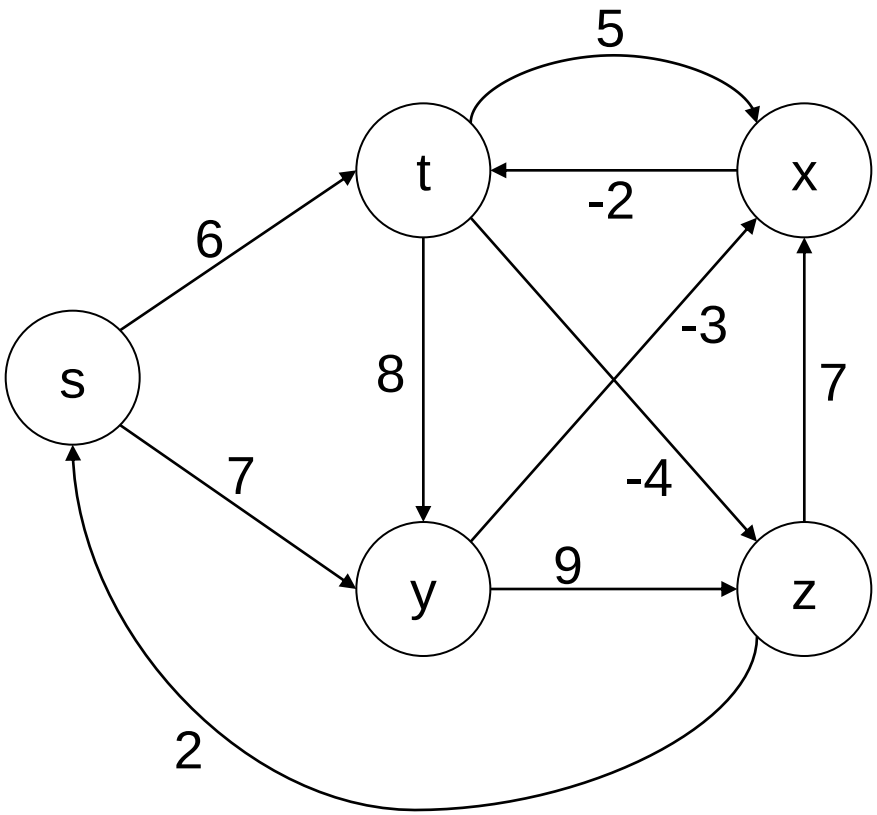
Algoritmo de Bellman-Ford

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

- Método
  1. Faz estimativas pessimistas para cada vértice
  2. Faz |V|-1 vezes o relaxamento de todas as arestas, garantindo que os caminhos mínimos prevaleçam

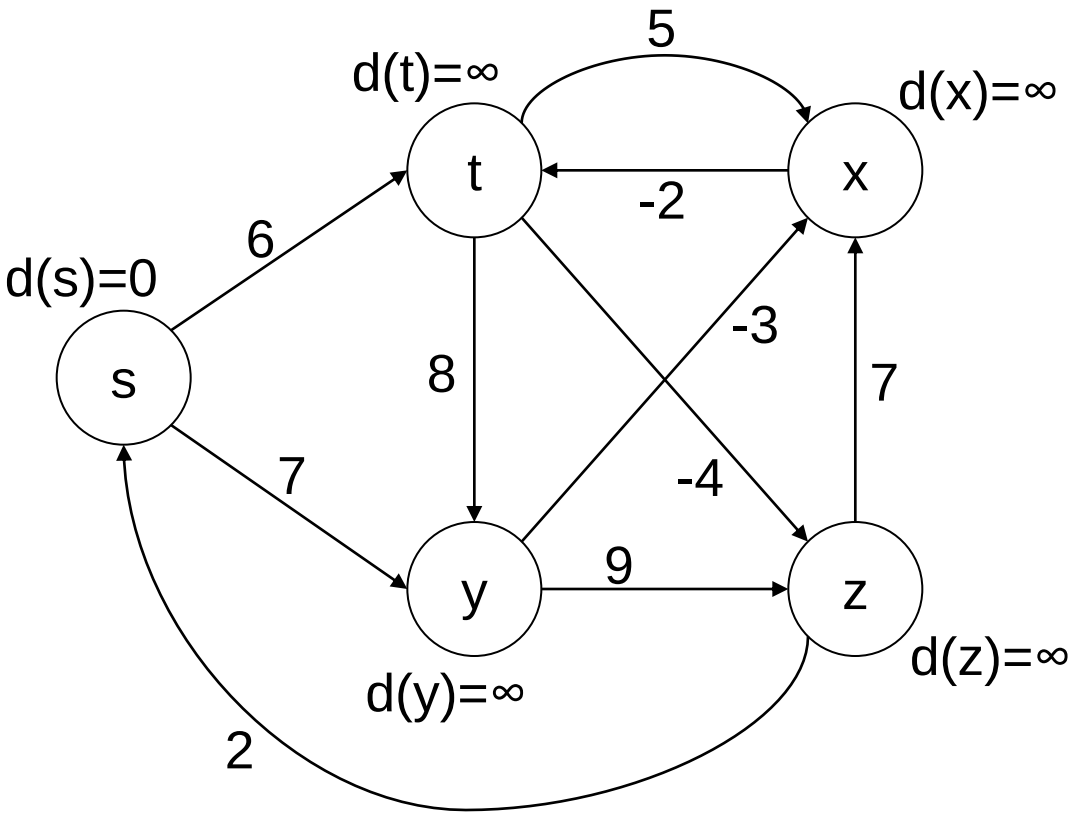
Algoritmo de Bellman-Ford

- Exemplo (a partir de s)



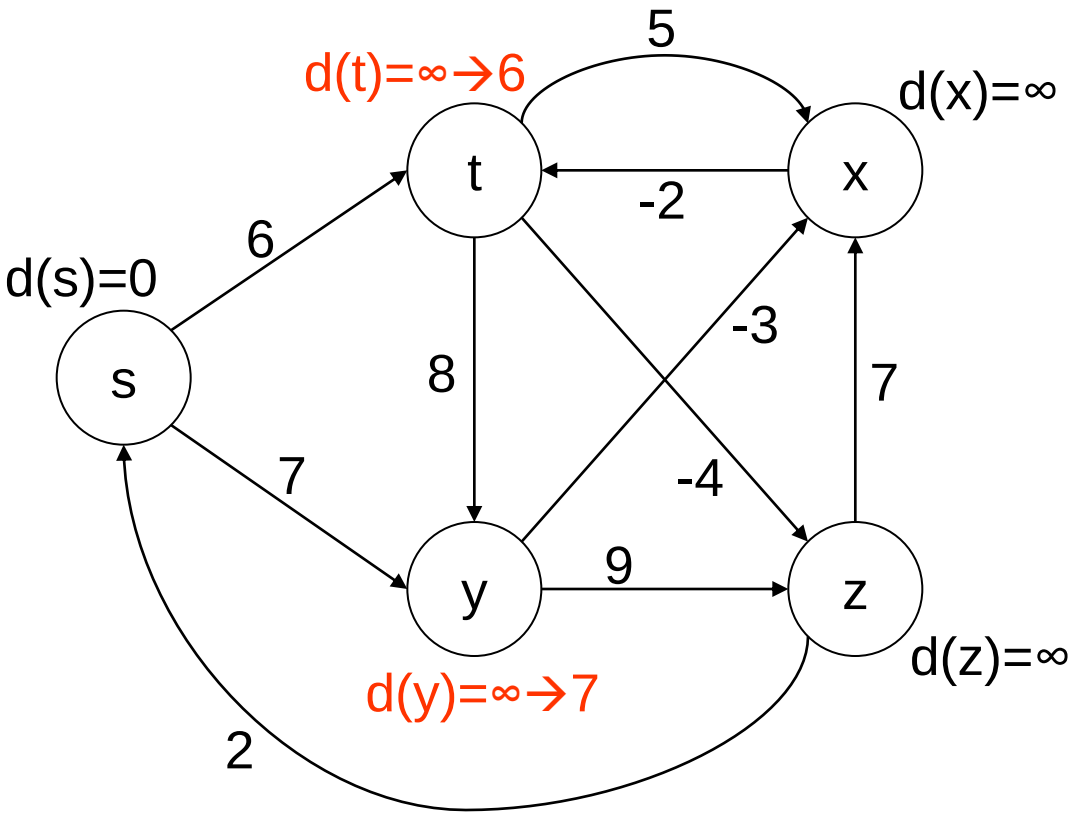
Algoritmo de Bellman-Ford

- Exemplo (a partir de s)
  - Estimativas pessimistas



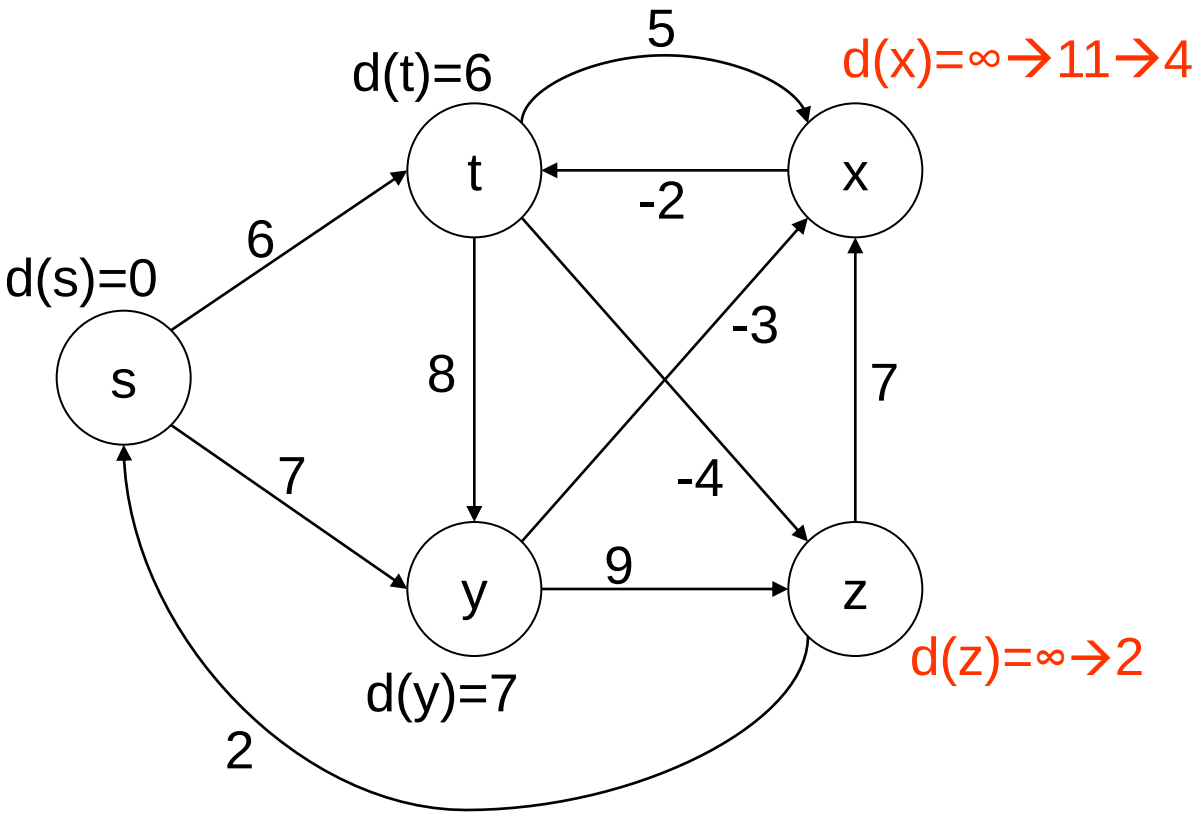
Algoritmo de Bellman-Ford

- Exemplo (a partir de s)
  - Rodada 1 (de |V|-1=4) de relaxamento
    - (t,x), (t,y), (t,z)
    - (x,t)
    - (y,x), (y,z)
    - (z,x), (z,s)
    - (s,t), (s,y)



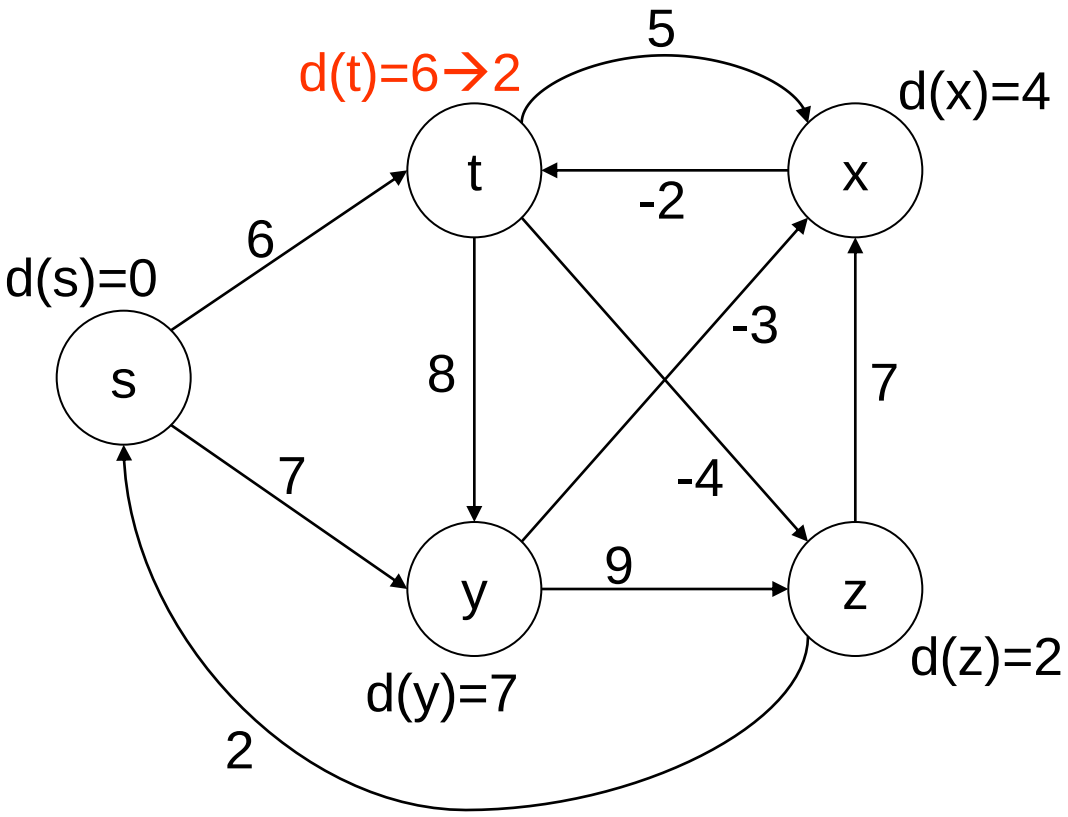
Algoritmo de Bellman-Ford

- Exemplo (a partir de s)
  - Rodada 2 (de |V|-1=4) de relaxamento
    - (t,x), (t,y), (t,z)
    - (x,t)
    - (y,x), (y,z)
    - (z,x), (z,s)
    - (s,t), (s,y)



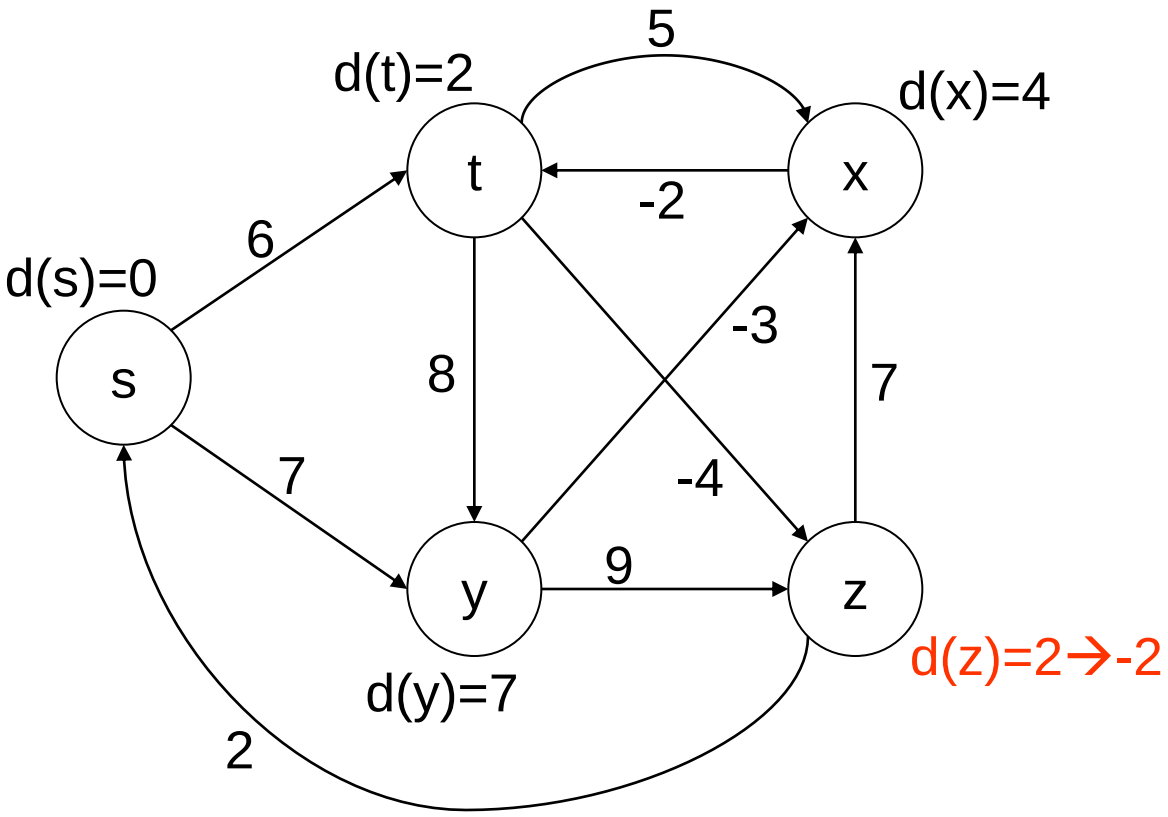
Algoritmo de Bellman-Ford

- Exemplo (a partir de s)
  - Rodada 3 (de |V|-1=4) de relaxamento
    - (t,x), (t,y), (t,z)
    - (x,t)
    - (y,x), (y,z)
    - (z,x), (z,s)
    - (s,t), (s,y)



Algoritmo de Bellman-Ford

- Exemplo (a partir de s)
  - Rodada 4 (de |V|-1=4) de relaxamento
    - (t,x), (t,y), (t,z)
    - (x,t)
    - (y,x), (y,z)
    - (z,x), (z,s)
    - (s,t), (s,y)



Algoritmo de Bellman-Ford

- Na implementação, uso do antecessor
  - Sempre que um relaxamento de aresta (u,v) é realizado, o antecessor é armazenado
    - antecessor[v]=u
  - Para se saber o caminho mínimo da origem s até qualquer vértice v, basta que se percorra o antecessor de v até s

Algoritmo de Bellman-Ford

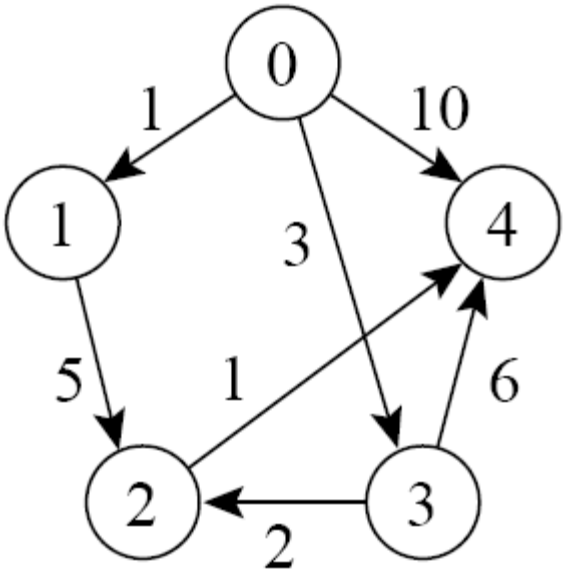
```
BELLMAN-FORD(G, w, s)
    início
    //inicializa variáveis
    para cada vértice v faça
            d[v]=∞
            antecessor[v]=-1
    d[s]=0
    //faz relaxamento de arestas e determino caminhos mais curtos
    para i=1 até |V| faça
            para cada aresta (u,v) faça
                    relax(u,v,w)
    //verifica se há ciclos negativos
    para cada aresta (u,v) faça
            se d[v] > d[u]+w(u,v) então
                    retorna FALSO
    retorna VERDADEIRO
    fim
```

Algoritmo de Bellman-Ford

- Complexidade de tempo: O(|V| x |E|)
  - Por quê?

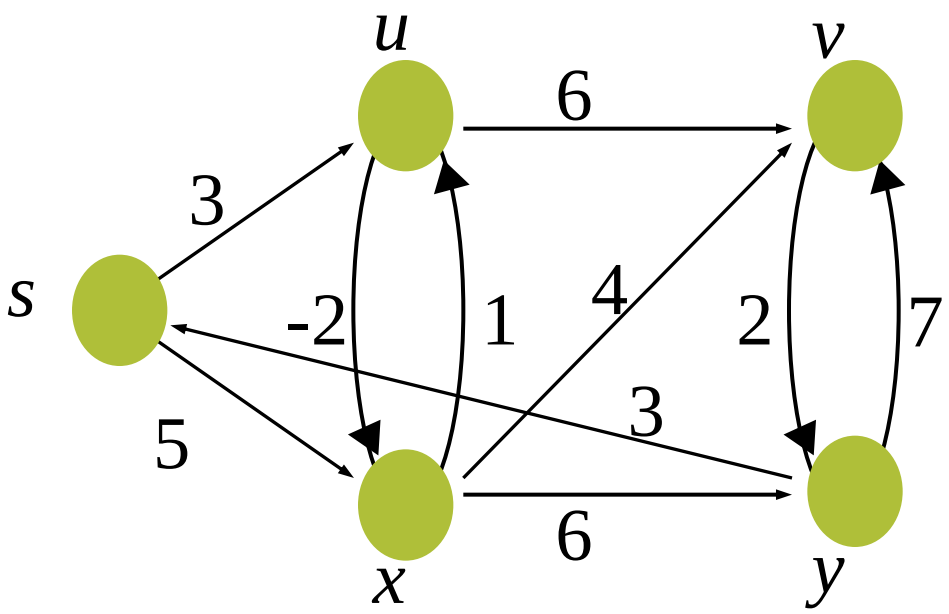
Exercício

- Calcule os caminhos mínimos para o grafo abaixo a partir do vértice 0 aplicando o algoritmo de Bellman-Ford



Exercício

- Calcule os caminhos mínimos para o grafo abaixo a partir do vértice s aplicando o algoritmo de Bellman-Ford



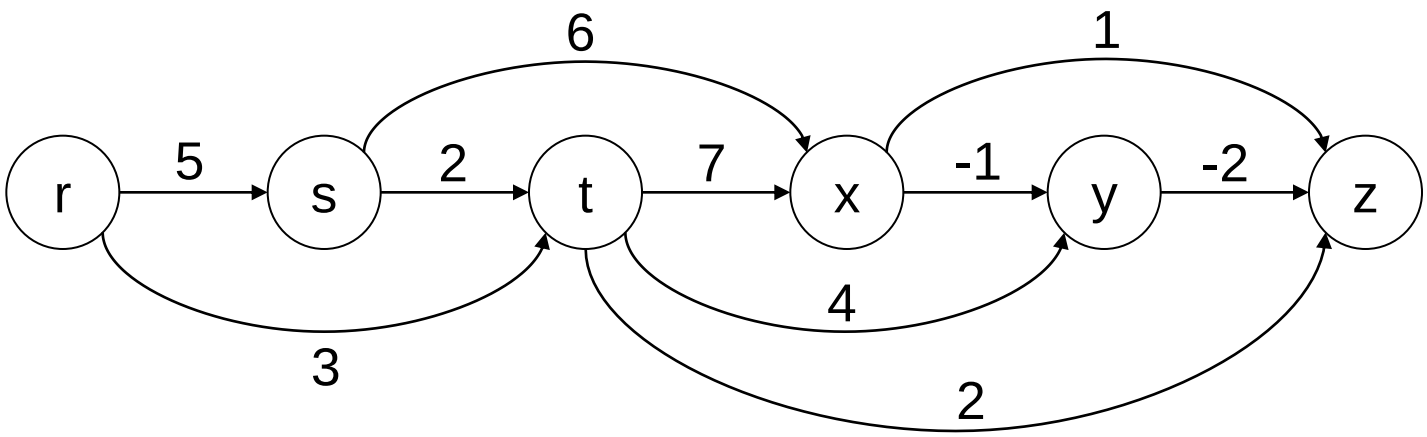
Algoritmo baseado na ordenação topológica

- Características
  - Caminho mais curto de origem única
  - Grafos sem ciclos
  - Podem haver pesos negativos

- Método
  - Faz-se a ordenação topológica do grafo
  - Percorre-se a lista de vértices relaxando-se todas as arestas que partem de cada vértice

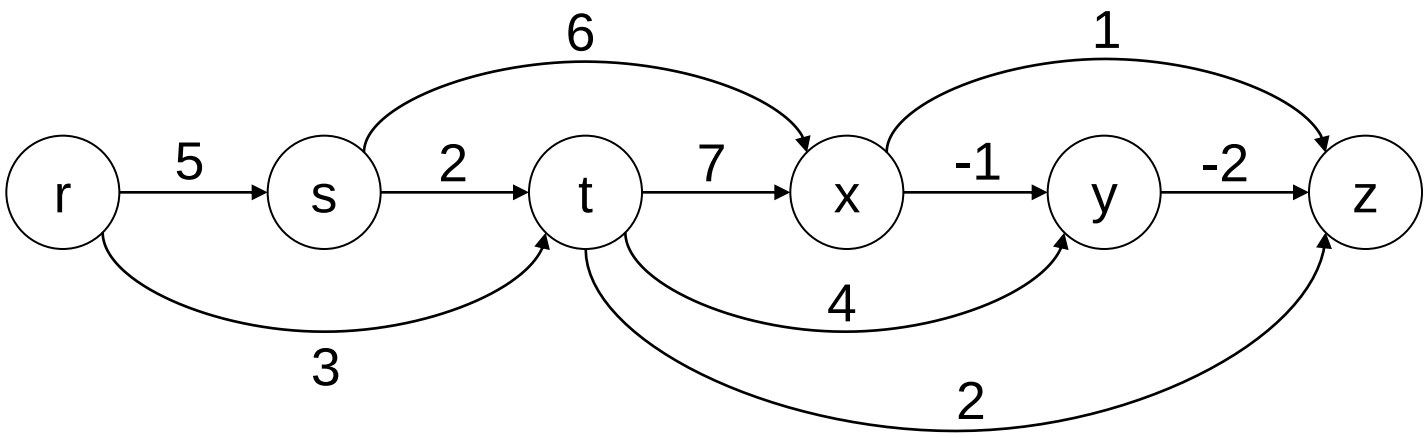
Algoritmo baseado na ordenação topológica

- Exemplo (a partir de s)



Algoritmo baseado na ordenação topológica

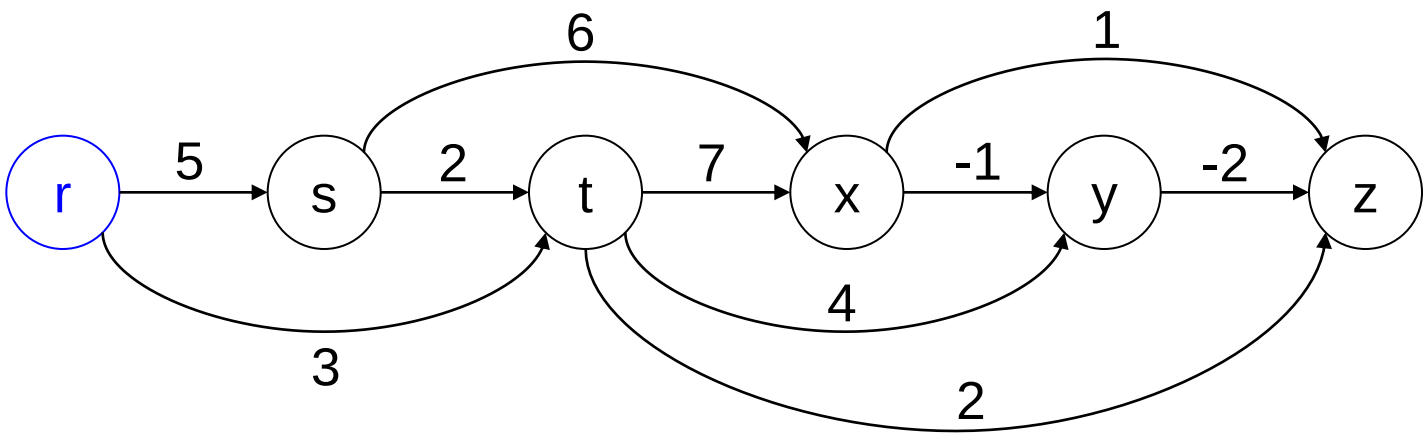
- Exemplo (a partir de s)
  - Estimativas pessimistas



d(r)=∞ d(s)=0 d(t)=∞ d(x)=∞ d(y)=∞ d(z)=∞

Algoritmo baseado na ordenação topológica

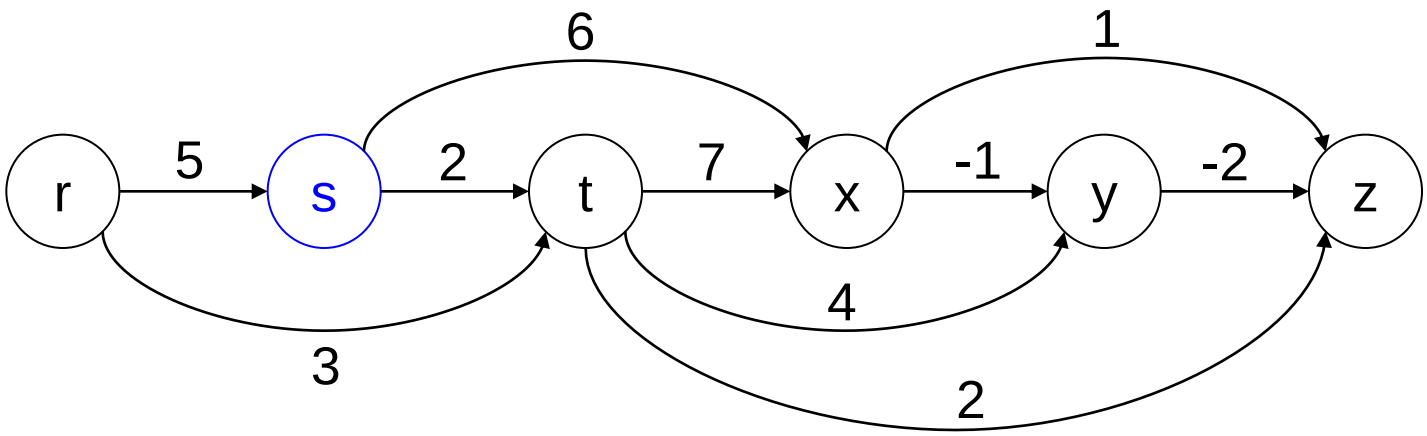
- Exemplo (a partir de s)
  - Arestas adjacentes a r



d(r)=∞ d(s)=0 d(t)=∞ d(x)=∞ d(y)=∞ d(z)=∞

Algoritmo baseado na ordenação topológica

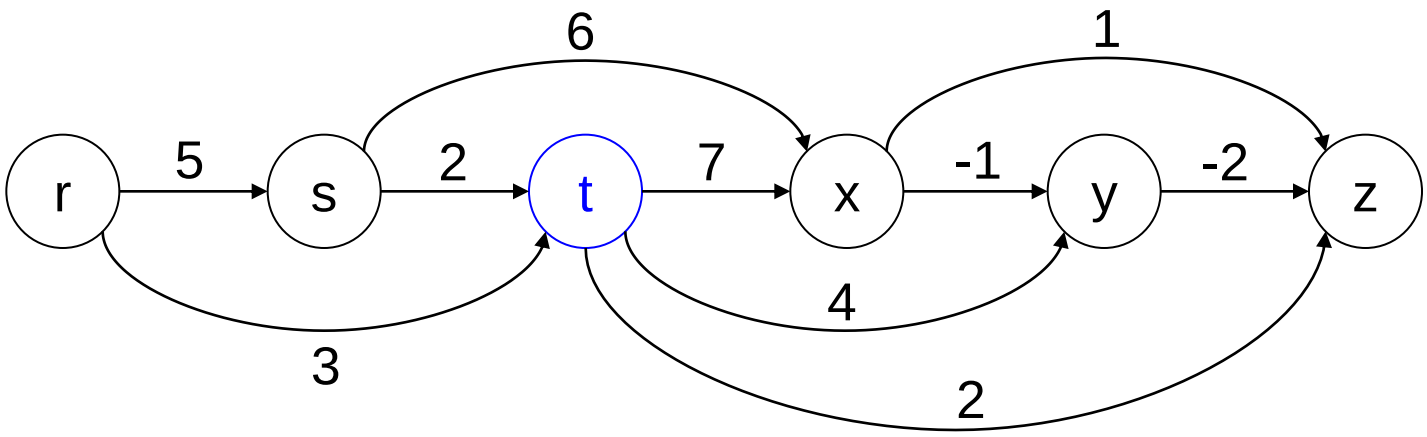
- Exemplo (a partir de s)
  - Arestas adjacentes a s



d(r)=∞ d(s)=0 d(t)=∞→2 d(x)=∞→6 d(y)=∞ d(z)=∞

Algoritmo baseado na ordenação topológica

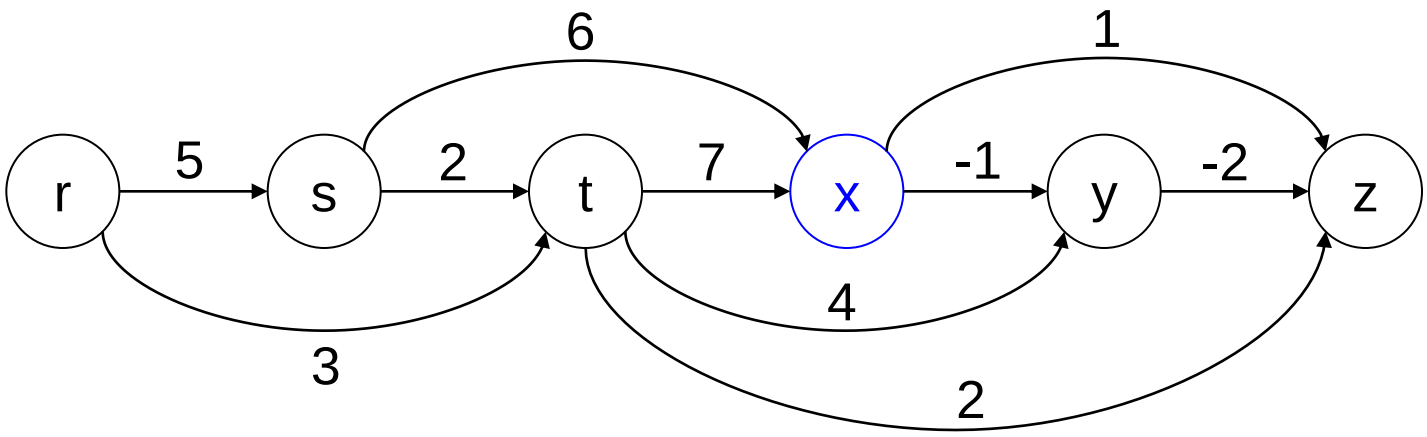
- Exemplo (a partir de s)
  - Arestas adjacentes a t



d(r)=∞ d(s)=0 d(t)=2 d(x)=6 d(y)=∞→6 d(z)=∞→4

Algoritmo baseado na ordenação topológica

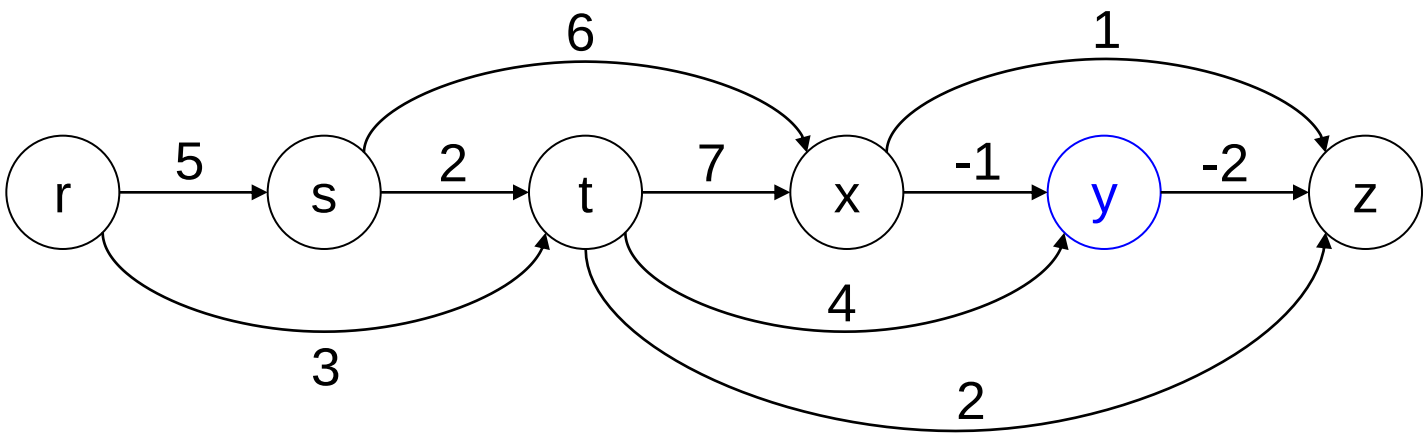
- Exemplo (a partir de s)
  - Arestas adjacentes a x



d(r)=∞ d(s)=0 d(t)=2 d(x)=6 d(y)=6→5 d(z)=4

Algoritmo baseado na ordenação topológica

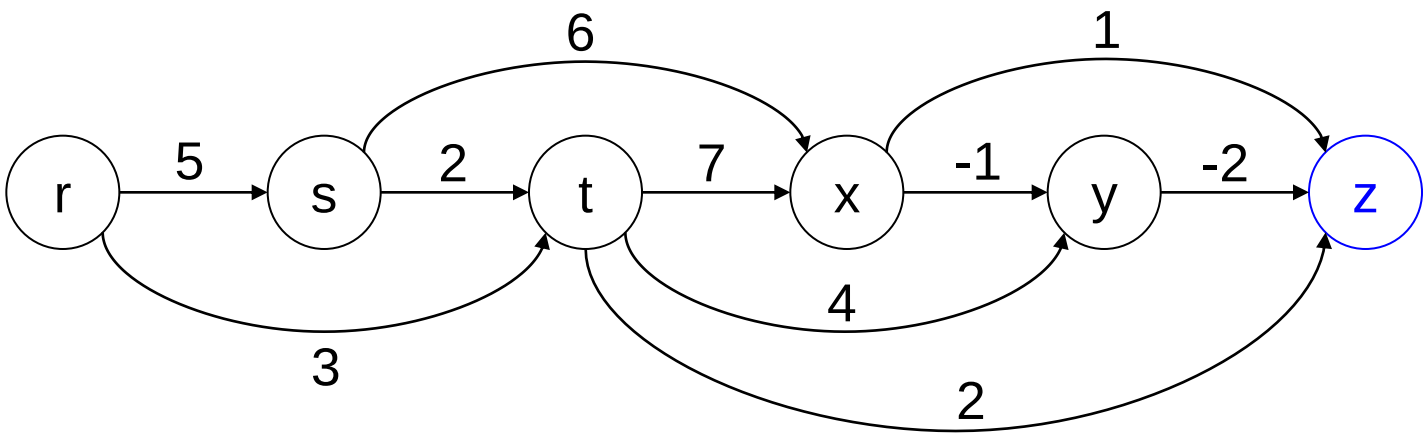
- Exemplo (a partir de s)
  - Arestas adjacentes a y



d(r)=∞ d(s)=0 d(t)=2 d(x)=6 d(y)=5 d(z)=4→3

Algoritmo baseado na ordenação topológica

- Exemplo (a partir de s)
  - Arestas adjacentes a z



d(r)=∞ d(s)=0 d(t)=2 d(x)=6 d(y)=5 d(z)=3

Algoritmo baseado na ordenação topológica

Caminho mínimo baseado na ordenação topológica(G, w, s)

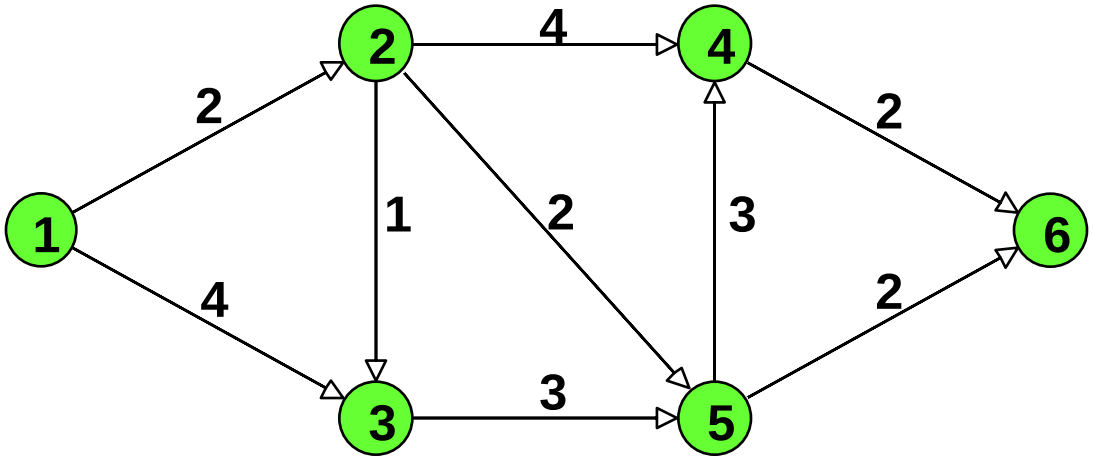
```
início
//ordenação topológica
ordenar topologicamente o grafo
//inicializa variáveis
para cada vértice v faça
        d[v]=∞
        antecessor[v]=-1
d[s]=0
//faz relaxamento de arestas e determino caminhos mais curtos
para cada vértice u tomado em seqüência topológica faça
        para cada vértice v adjacente a u faça
                relax(u,v,w)
fim
```

Algoritmo baseado na ordenação topológica

- Complexidade de tempo: O(|V| + |E|)
  - Por quê?

Exercício

- Calcule os caminhos mínimos para o grafo abaixo a partir do vértice 1 aplicando o algoritmo baseado na ordenação topológica



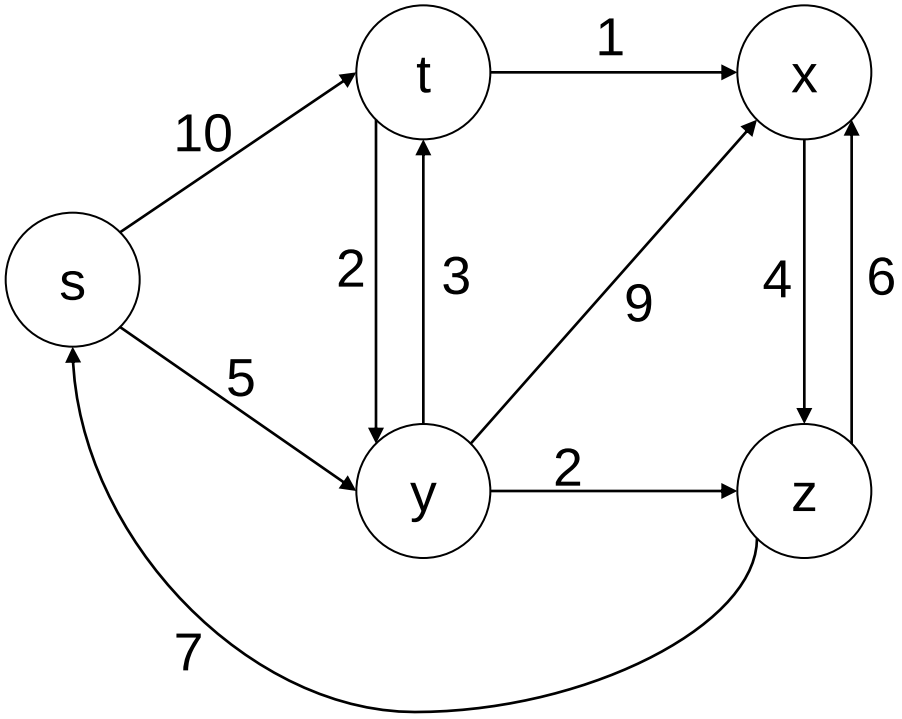
Algoritmo de Dijkstra

- Características
  - Caminho mais curto de origem única
  - Podem haver ciclos
  - Somente pesos positivos

- Método
  - A cada passo, adiciona um vértice u de menor estimativa de caminho mínimo a um conjunto S com vértices com caminhos mínimos desde a origem já definidos
  - Relaxam-se as arestas adjacentes a u

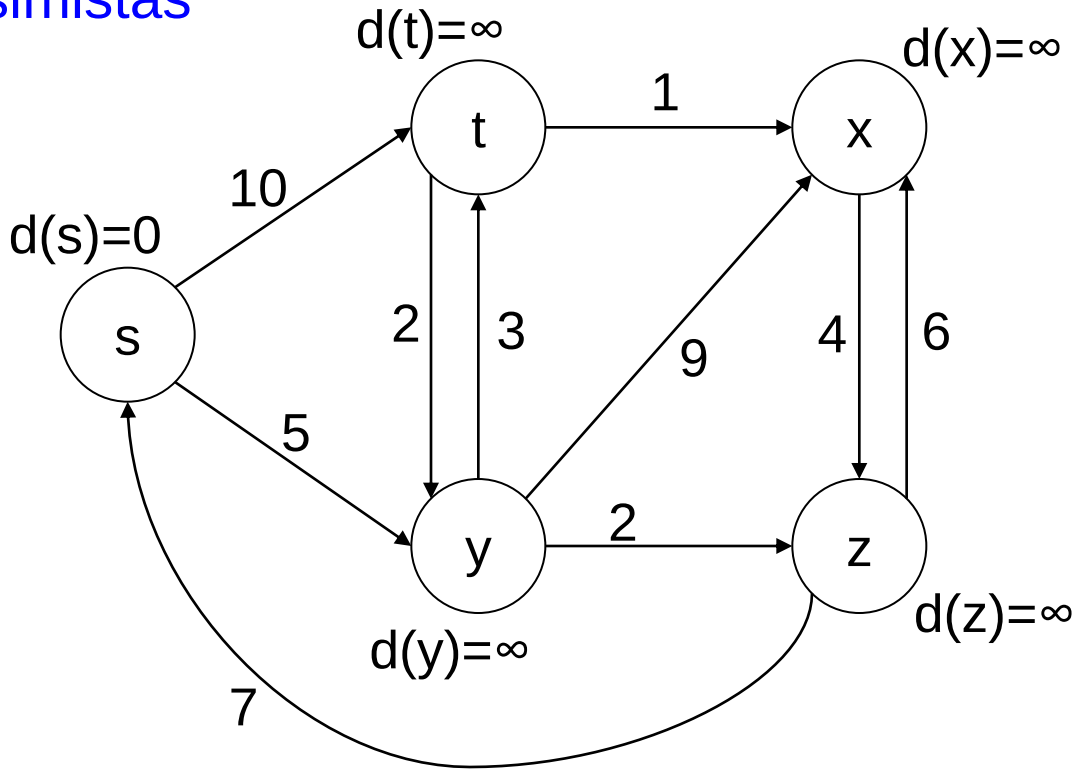
Algoritmo de Dijkstra

- Exemplo (a partir de s)



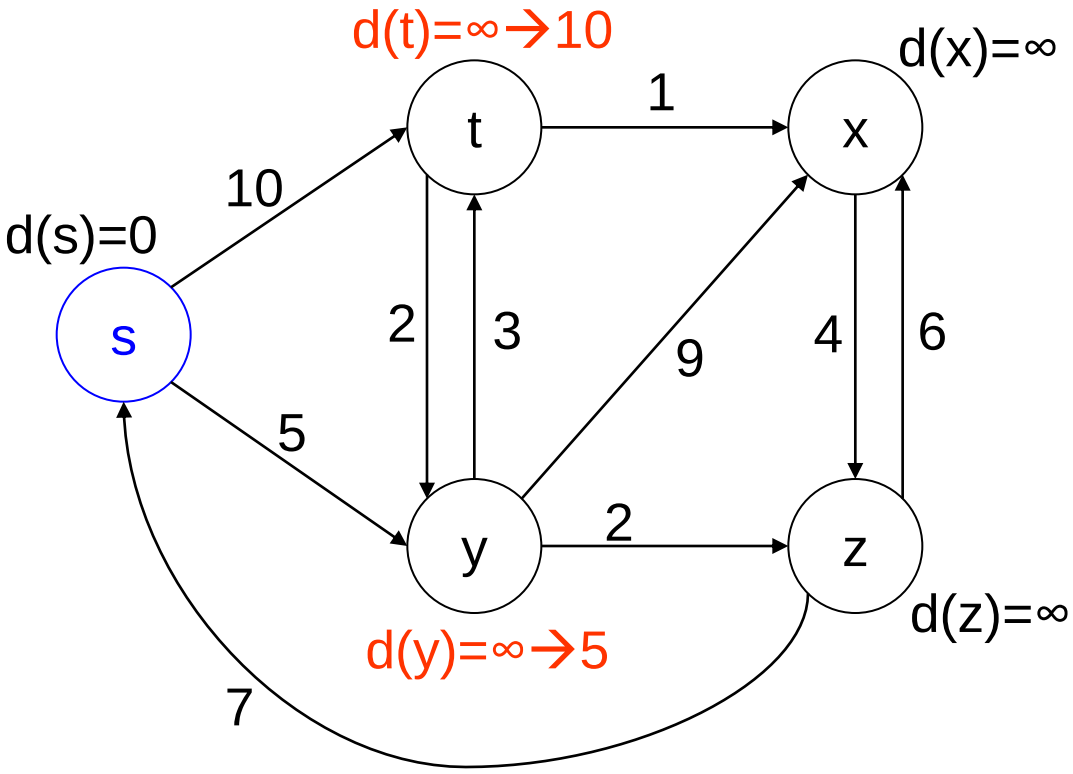
Algoritmo de Dijkstra

- Exemplo (a partir de s)
  - Estimativas pessimistas
  - S=Ø



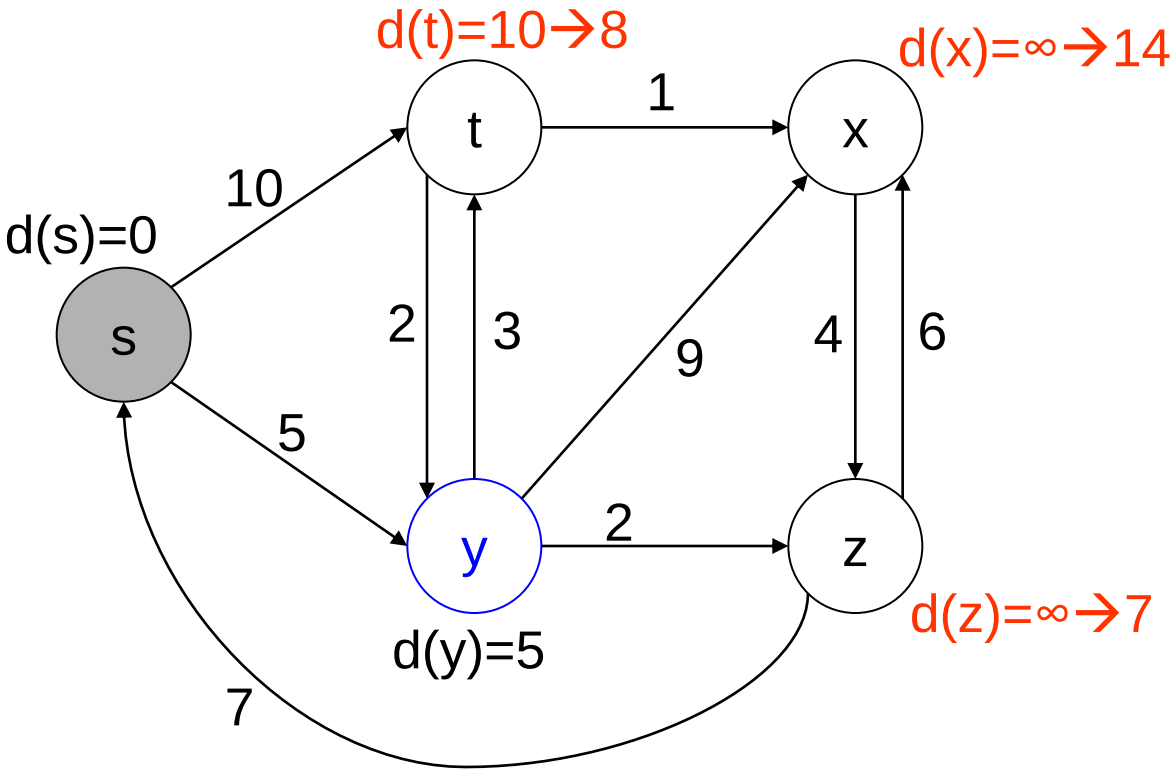
Algoritmo de Dijkstra

- Exemplo (a partir de s)
  - Adiciona s a S
  - S={**s**}



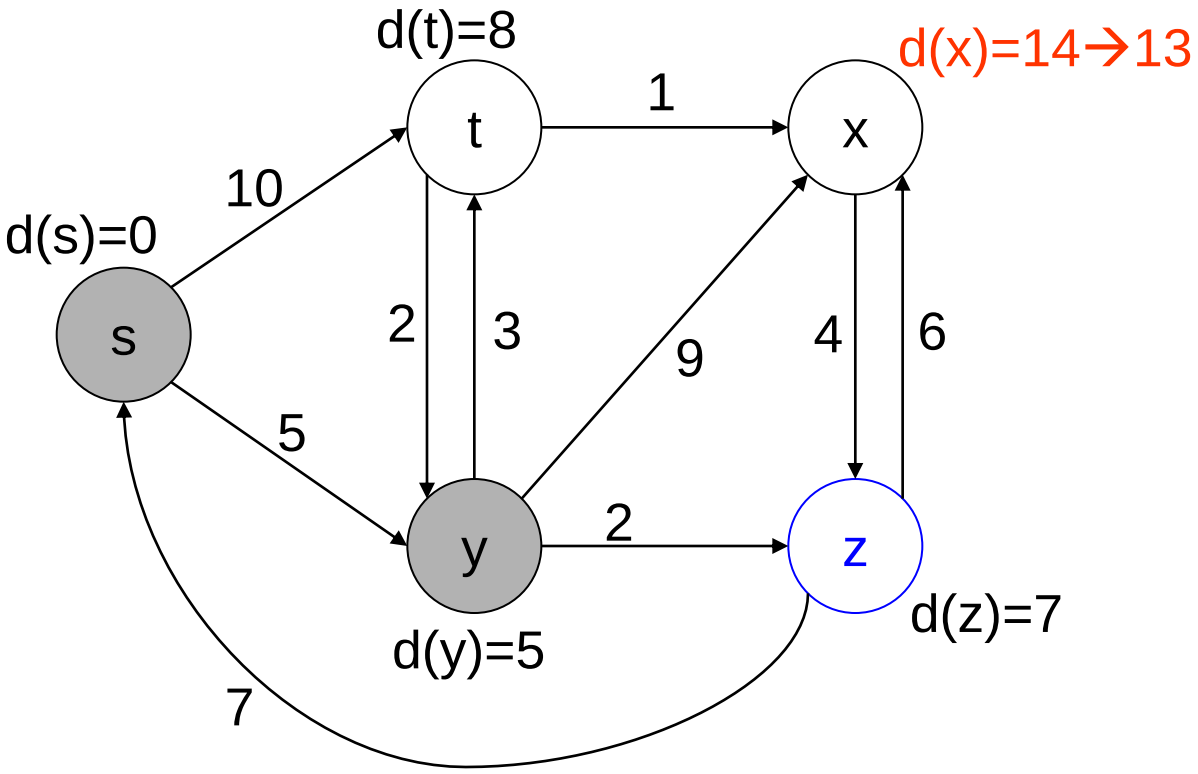
Algoritmo de Dijkstra

- Exemplo (a partir de s)
  - Adiciona y a S
  - S={s,**y**}



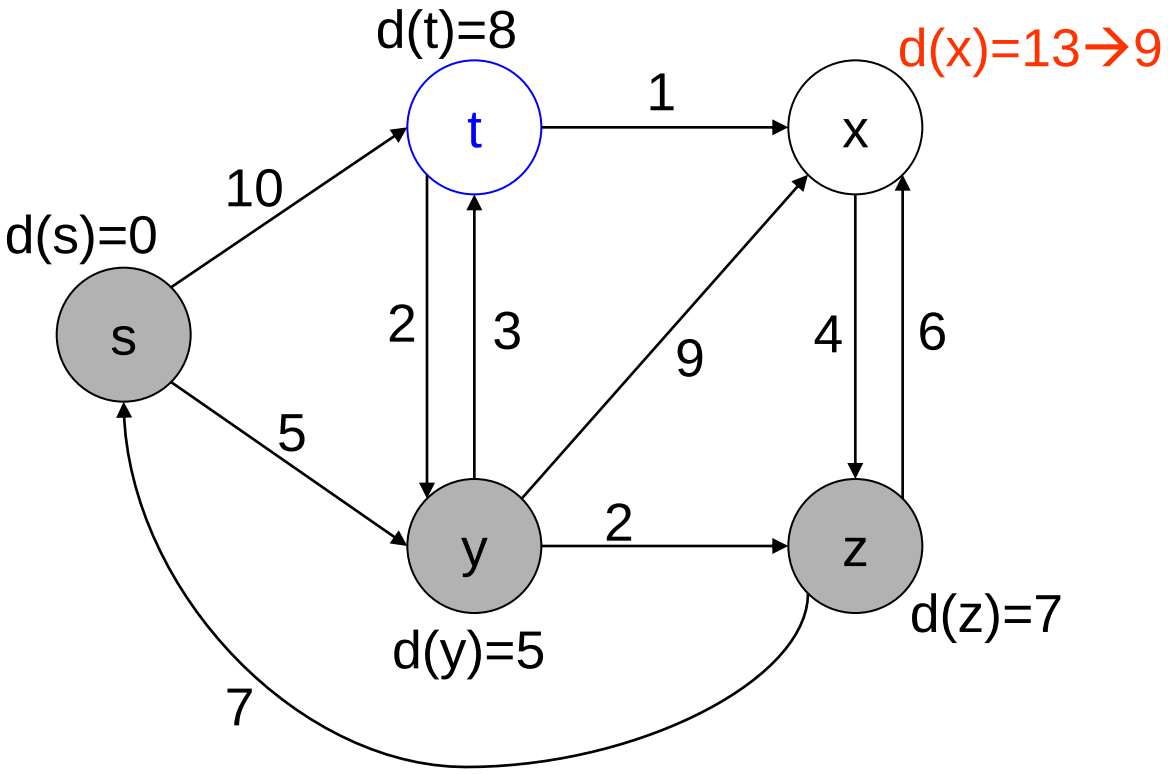
Algoritmo de Dijkstra

- Exemplo (a partir de s)
  - Adiciona z a S
  - S={s,y,**z**}



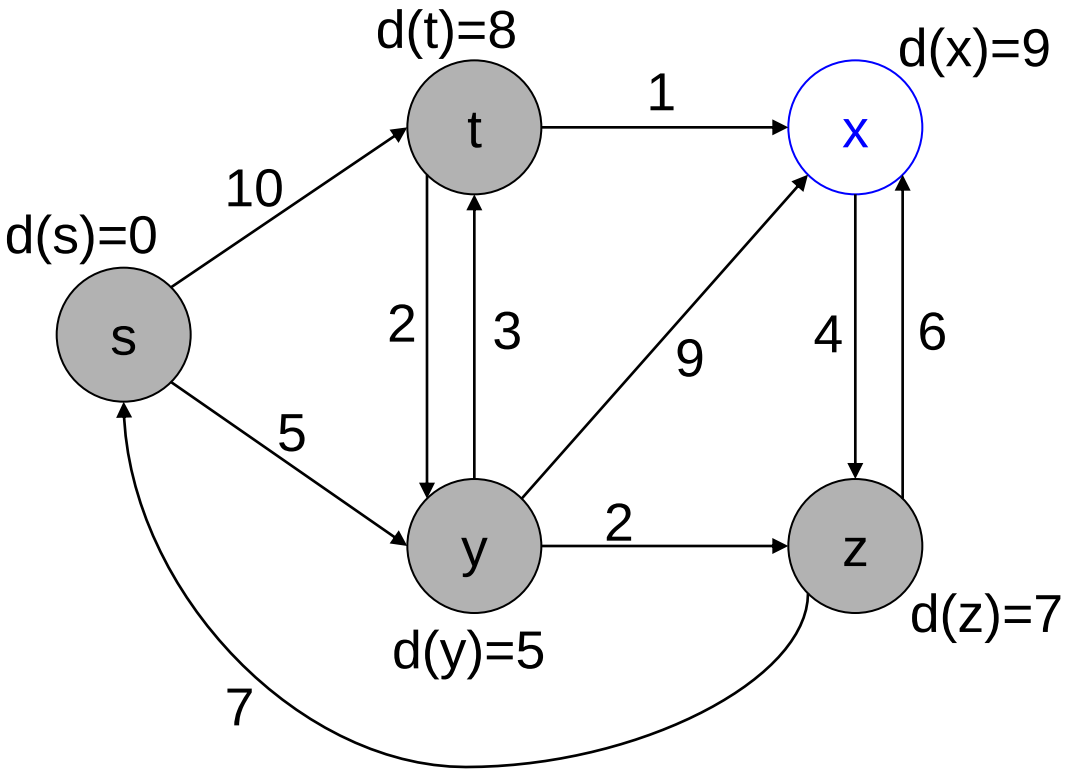
Algoritmo de Dijkstra

- Exemplo (a partir de s)
  - Adiciona t a S
  - S={s,y,z,**t**}



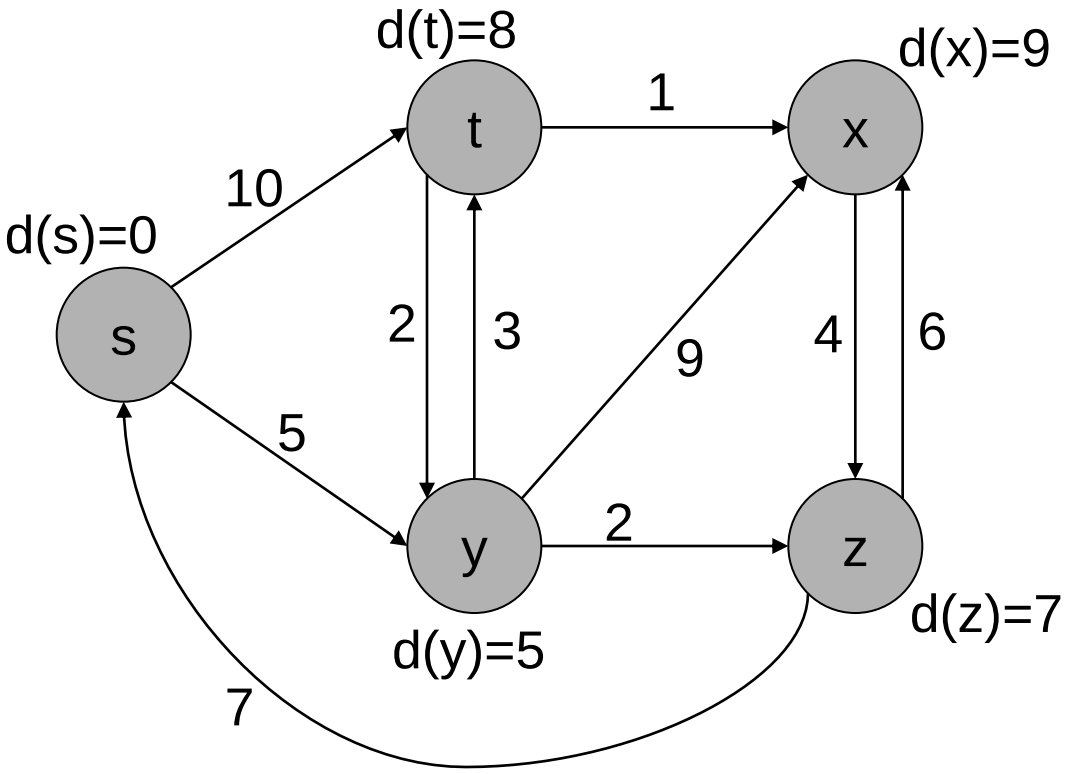
Algoritmo de Dijkstra

- Exemplo (a partir de s)
  - Adiciona x a S
  - S={s,y,z,t,**x**}



Algoritmo de Dijkstra

- Exemplo (a partir de s)
  - S={s,y,z,t,x}



Algoritmo de Dijkstra

- Implementação
  - Uso de uma fila de prioridades com vértices organizados em função da estimativa d de caminho mínimo

Algoritmo de Dijkstra

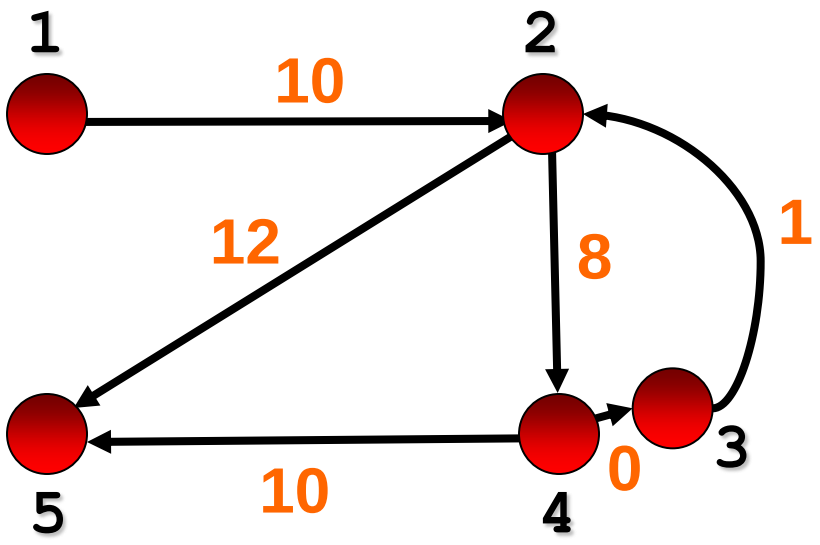
```
DIJKSTRA(G, w, s)
    início
    //inicializa variáveis
    para cada vértice v faça
        d[v]=∞
        antecessor[v]=-1
    d[s]=0
    S=Ø
    cria fila de prioridade F com vértices do grafo
    //insere vértice u em S e faz relaxamento das arestas adjacentes
    enquanto F≠Ø faça
        u=retirar vértice de F
        S=S+{u}
        para cada vértice v adjacente a u faça
            relax(u,v,w)
    fim
```

Algoritmo de Dijkstra

- Complexidade de tempo: O(|V| log|V| + |E|), se fila de prioridade bem implementada

Exercício

- Calcule os caminhos mínimos para o grafo abaixo a partir do vértice 1 aplicando o algoritmo de Dijkstra



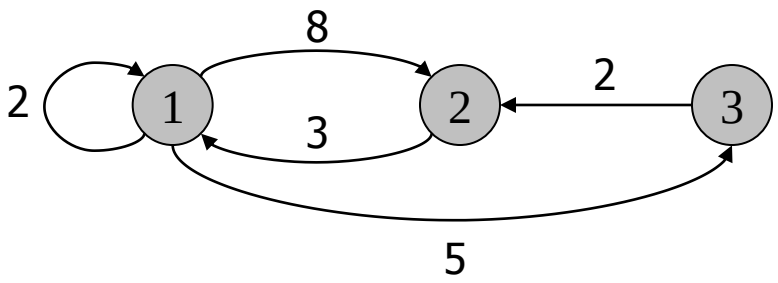
Caminhos Mais Curtos de Todos os Pares

- Suponha que um grafo orientado ponderado representa as possíveis rotas de uma companhia aérea conectando diversas cidades
- O objetivo é construir uma tabela com os menores caminhos entre todas as cidades
- Esse e um exemplo de problema que exige encontrar os **caminhos mais curtos para todos os pares de vértices**

Caminhos Mais Curtos de Todos os Pares

- Uma possível solução é utilizar o algoritmo de Dijkstra utilizando cada vértice como origem alternadamente
- Uma solução mais direta é utilizar o **algoritmo de Floyd**
- O algoritmo de Floyd utiliza uma matriz ***A*** **$|V|\times|V|$** para calcular e armazenar os tamanhos dos caminhos mais curtos

Caminhos Mais Curtos de Todos os Pares: Algoritmo de Floyd

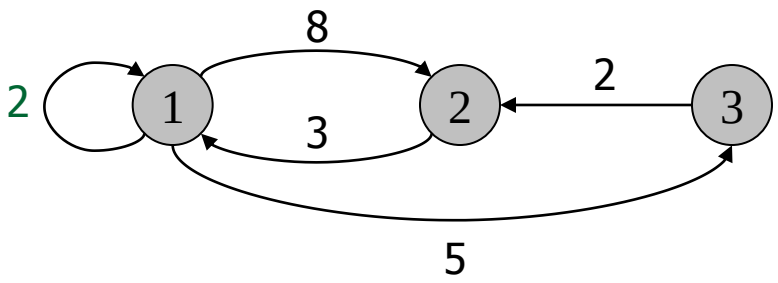


	1	2	3
1			
2			
3			

A

- Inicialmente os custos entre vértices adjacentes são inseridos na tabela A
- Pesos de self-loops não são considerados

Caminhos Mais Curtos de Todos os Pares: Algoritmo de Floyd

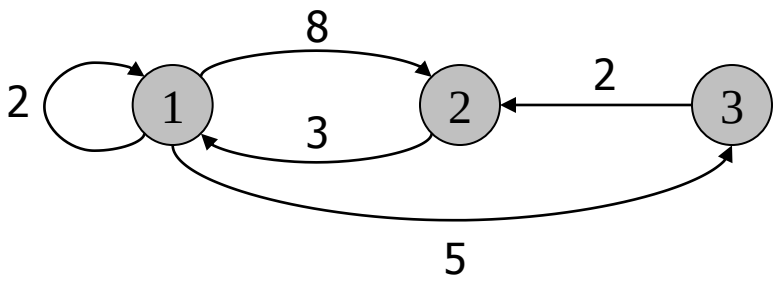


	1	2	3
1	0	8	5
2	3	0	∞
3	∞	2	0

A

- Inicialmente os custos entre vértices adjacentes são inseridos na tabela A
- Pesos de self-loops não são considerados

Caminhos Mais Curtos de Todos os Pares: Algoritmo de Floyd

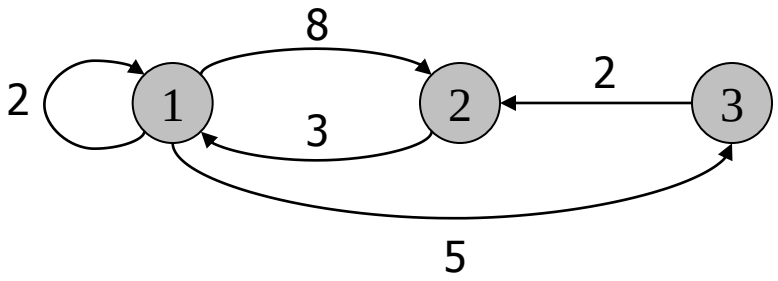


	1	2	3
1	0	8	5
2	3	0	∞
3	∞	2	0

A

- A matriz *A* é percorrida |*V*| vezes
- A cada iteração *k*, verifica-se se um caminho entre dois vértices *(v,w)* que passa também pelo vértice *k* é mais curto do que o caminho mais curto conhecido

Caminhos Mais Curtos de Todos os Pares: Algoritmo de Floyd



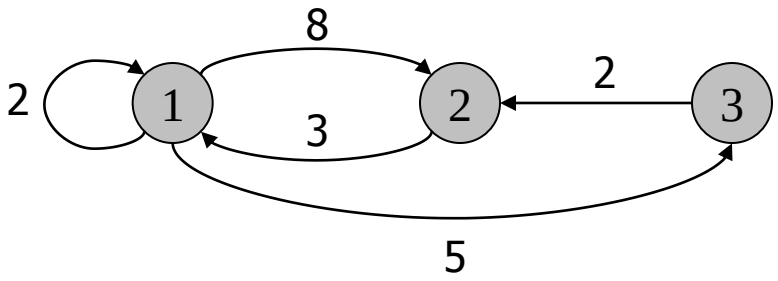
Ou seja:

A[v,w] = min(A[v,w],
A[v,k] + A[k,w]).

	1	2	3
1	0	8	5
2	3	0	∞
3	∞	2	0

A

Caminhos Mais Curtos de Todos os Pares: Algoritmo de Floyd



Ou seja:

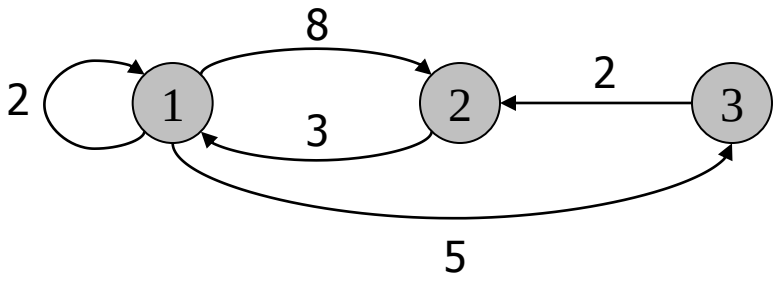
A[1,1] = min(A[1,1],
A[1,1] + A[1,1]).

	1	2	3
1	0	8	5
2	3	0	∞
3	∞	2	0

A

k = 1

Caminhos Mais Curtos de Todos os Pares: Algoritmo de Floyd



Ou seja:

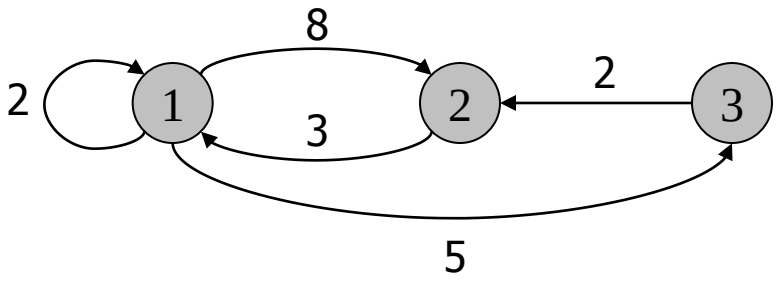
A[1,2] = min(A[1,2],
A[1,1] + A[1,2]).

	1	2	3
1	0	8	5
2	3	0	∞
3	∞	2	0

A

k = 1

Caminhos Mais Curtos de Todos os Pares: Algoritmo de Floyd



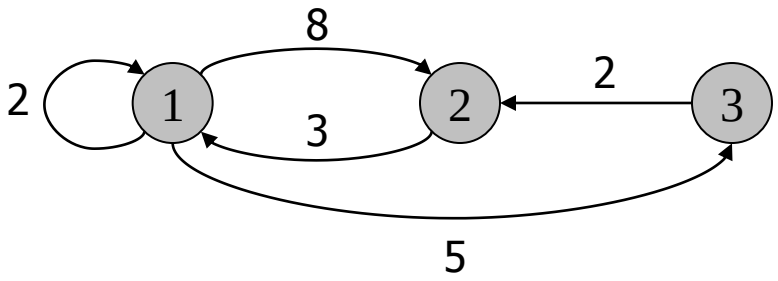
Ou seja:

A[1,3] = min(A[1,3],
A[1,1] + A[1,3]).

	1	2	3
1	0	8	5
2	3	0	∞
3	∞	2	0

k = 1

Caminhos Mais Curtos de Todos os Pares: Algoritmo de Floyd



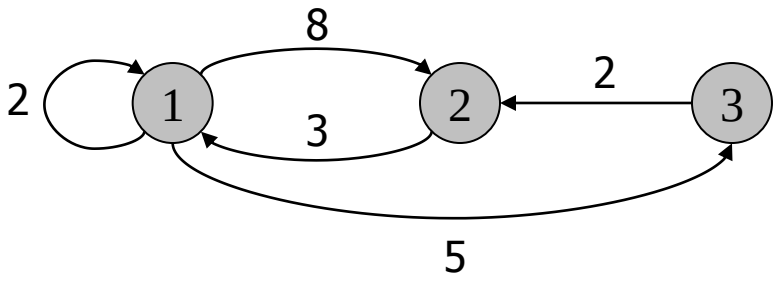
Ou seja:

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

	1	2	3
1	0	8	5
2	3	0	∞
3	∞	2	0

k = 1

Caminhos Mais Curtos de Todos os Pares: Algoritmo de Floyd



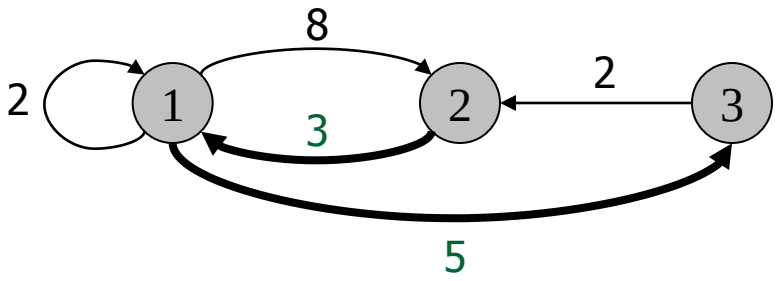
Ou seja:

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

	1	2	3
1	0	8	5
2	3	0	∞
3	∞	2	0

k = 1

Caminhos Mais Curtos de Todos os Pares: Algoritmo de Floyd



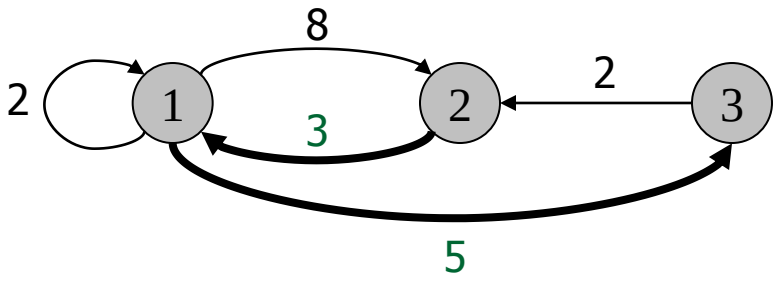
Ou seja:

A[2,3] = min(A[2,3],
A[2,1] + A[1,3]).

	1	2	3
1	0	8	5
2	3	0	∞
3	∞	2	0

k = 1

Caminhos Mais Curtos de Todos os Pares: Algoritmo de Floyd



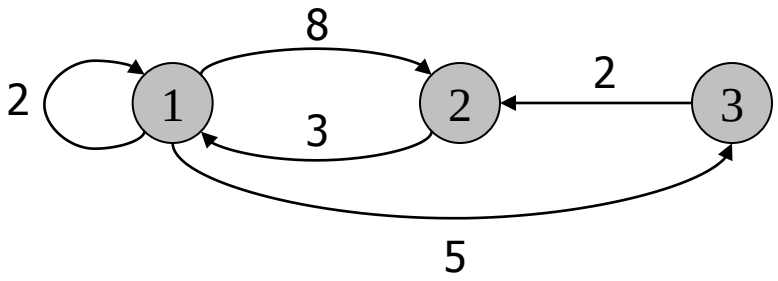
Ou seja:

A[2,3] = min(A[2,3],
A[2,1] + A[1,3]).

	1	2	3
1	0	8	5
2	3	0	8
3	∞	2	0

k = 1

Caminhos Mais Curtos de Todos os Pares: Algoritmo de Floyd



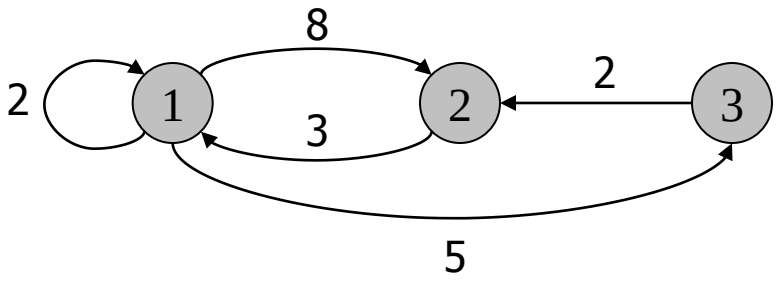
Ou seja:

A[3,1] = min(A[3,1],
A[3,1] + A[1,3]).

	1	2	3
1	0	8	5
2	3	0	8
3	∞	2	0

k = 1

Caminhos Mais Curtos de Todos os Pares: Algoritmo de Floyd



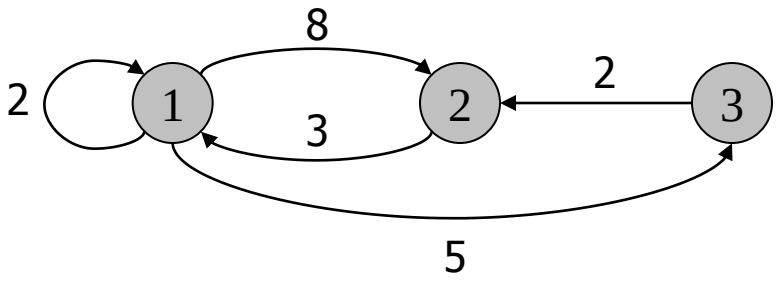
Ou seja:

A[3,2] = min(A[3,2],
A[3,1] + A[1,2]).

	1	2	3
1	0	8	5
2	3	0	8
3	∞	2	0

k = 1

Caminhos Mais Curtos de Todos os Pares: Algoritmo de Floyd



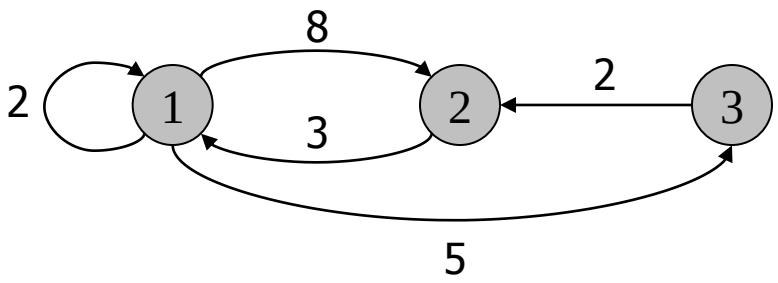
Ou seja:

A[3,3] = min(A[3,3],
A[3,1] + A[1,3]).

	1	2	3
1	0	8	5
2	3	0	8
3	∞	2	0

k = 1

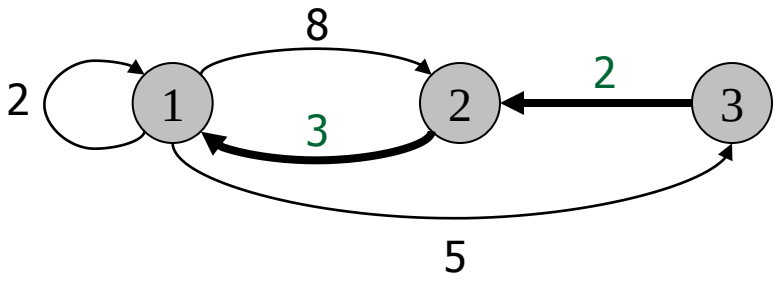
Caminhos Mais Curtos de Todos os Pares: Algoritmo de Floyd



	1	2	3
1	0	8	5
2	3	0	8
3	∞	2	0

- Ao final da iteração *k=1* tem-se todos os caminhos mais curtos entre *v* e *w* que podem passar pelo vértice 1
- O processo se repete para *k=2* e *k=3*

Caminhos Mais Curtos de Todos os Pares: Algoritmo de Floyd

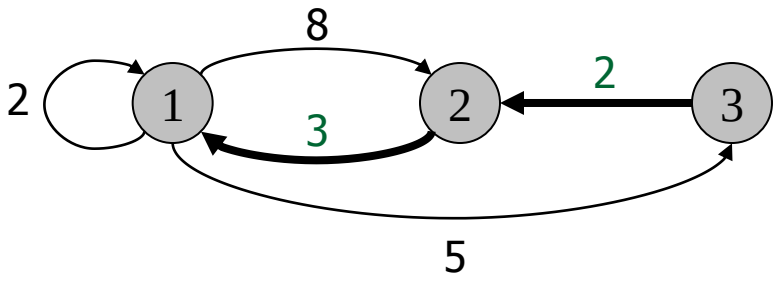


A[3,1] = min(A[3,1],
A[3,2] + A[2,1]).

	1	2	3
1	0	8	5
2	3	0	8
3	∞	2	0

k = 2

Caminhos Mais Curtos de Todos os Pares: Algoritmo de Floyd

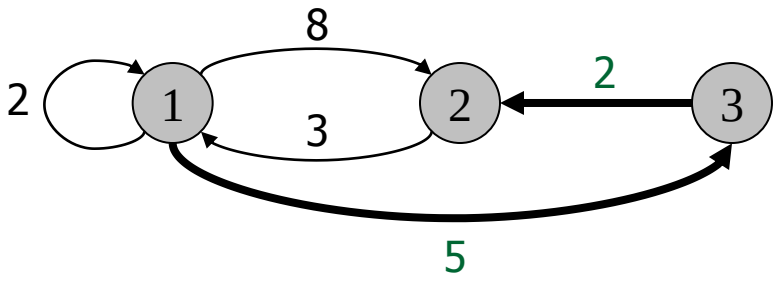


A[3,1] = min(A[3,1],
A[3,2] + A[2,1]).

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

k = 2

Caminhos Mais Curtos de Todos os Pares: Algoritmo de Floyd

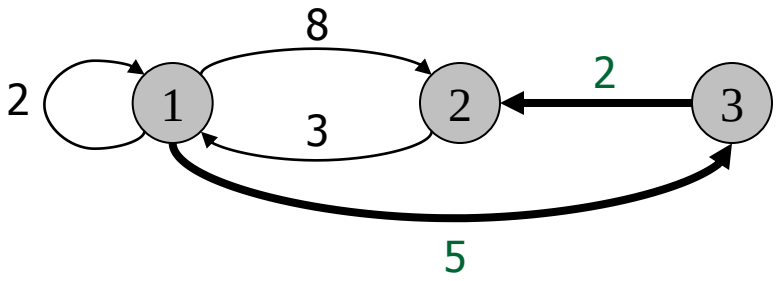


A[1,2] = min(A[1,2],
A[1,3] + A[3,2]).

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

k = 3

Caminhos Mais Curtos de Todos os Pares: Algoritmo de Floyd



A[1,2] = min(A[1,2],
A[1,3] + A[3,2]).

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

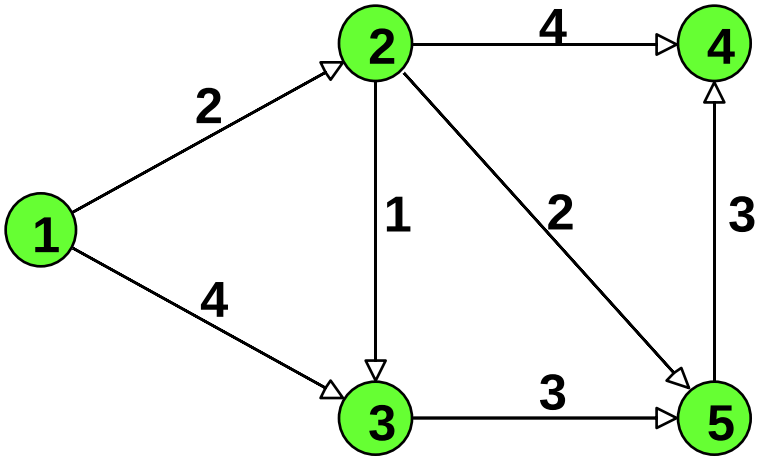
k = 3

Caminhos Mais Curtos de Todos os Pares: Algoritmo de Floyd

```
procedimento Floyd(var A: array[TVertice, TVertice]
                   de reais; var G: TGrafo)
variáveis
   v,w,k: TVertice;
início
   para v:=1 até G.NumVertices faça
        para w:=1 até G.NumVertices faça
              se v = w então
                    A[v,w] := 0;
              senão
                    A[v,w] := peso da aresta (v, w);

   para k:=1 até G.NumVertices faça
        para v:=1 até G.NumVertices faça
              para w:=1 até G.NumVertices faça
                    se A[v,k] + A[k,w] < A[v,w] então
                          A[v,w] := A[v,k] + A[k,w];

fim;
```

Caminhos Mais Curtos de Todos os Pares: Algoritmo de Floyd

- Exercício: aplique o algoritmo de Floyd para o grafo abaixo



Caminhos Mais Curtos de Todos os Pares: Algoritmo de Floyd

- O algoritmo de Floyd é $O(|V|^3)$

- No caso do algoritmo de Dijksta for utilizado para solucionar o problema de caminhos mais curtos entre todos os pares, o algoritmo de Floyd tem desempenho similar ao algoritmo de Dijkstra com matriz de adjacência

- Se o algoritmo de Dijkstra for implementado com listas de adjacências, então sua complexidade para encontrar os caminhos mais curtos entre todos os pares é $O(|V|\ |A|\ log\ n)$

Fechamento Transitivo: Algoritmo de Warshall

- Para alguns problemas pode ser interessante simplesmente saber se existe um caminho de qualquer tamanho entre dois vértices

- O algoritmo de Floyd pode ser especializado para esse problema, resultando no algoritmo de Warshall (o qual é mais antigo que o algoritmo de Floyd)

Fechamento Transitivo: Algoritmo de Warshall

- A partir da matriz de adjacência *M*, deseja-se criar uma matriz *A*, tal que *A[v, w] = 1*, se existe um caminho de qualquer tamanho entre *v* e *w*

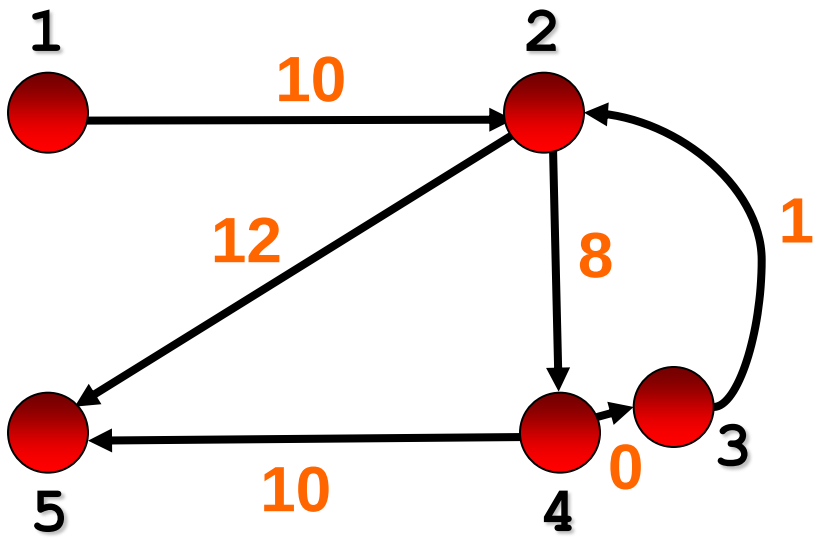
- Essa matriz é chamada de fechamento transitivo da matriz de adjacência

Fechamento Transitivo: Algoritmo de Warshall

```
procedimento Warshall(var A: array[TVertice,
             TVertice] de lógico; var G: TGrafo)
variáveis
   v,w,k: TVertice;

início
   para v:=1 até G.NumVertices faça
        para w:=1 até G.NumVertices faça
               A[v,w] := peso da aresta (v,w)>0;

   para k:=1 até G.NumVertices faça
        para v:=1 até G.NumVertices faça
               para w:=1 até G.NumVertices faça
                      se não A[v,w] então
                             A[v,w] := A[v,k] e A[k,w];

fim;
```

Fechamento Transitivo: Algoritmo de Warshall

- Exercício: aplique o algoritmo de Warshall para o grafo abaixo



Exercício

- Implemente à mão (em C) o algoritmo de Dijkstra para entregar na próxima aula