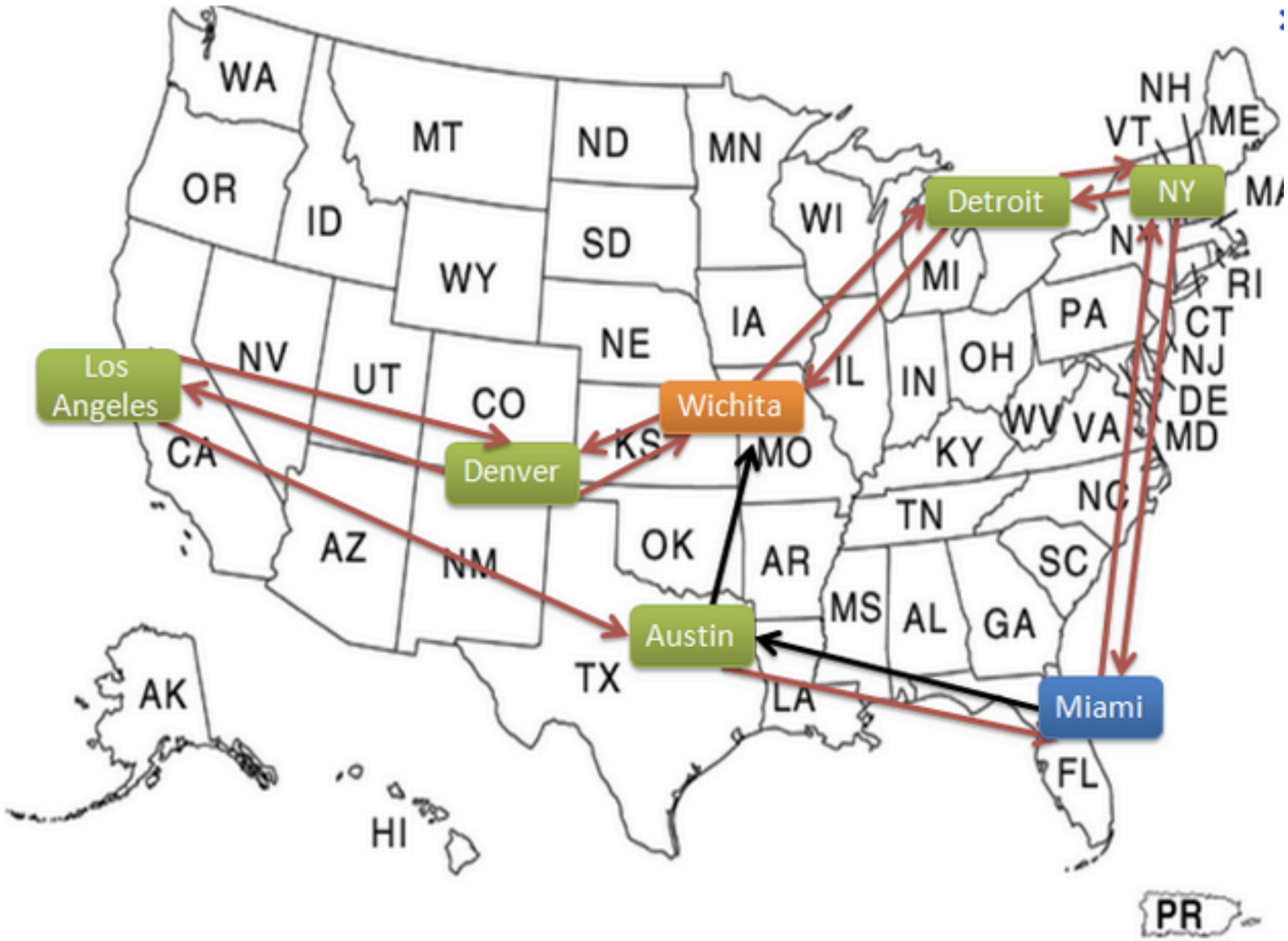
Caminho Mínimo

Prof. Leandro Alvim

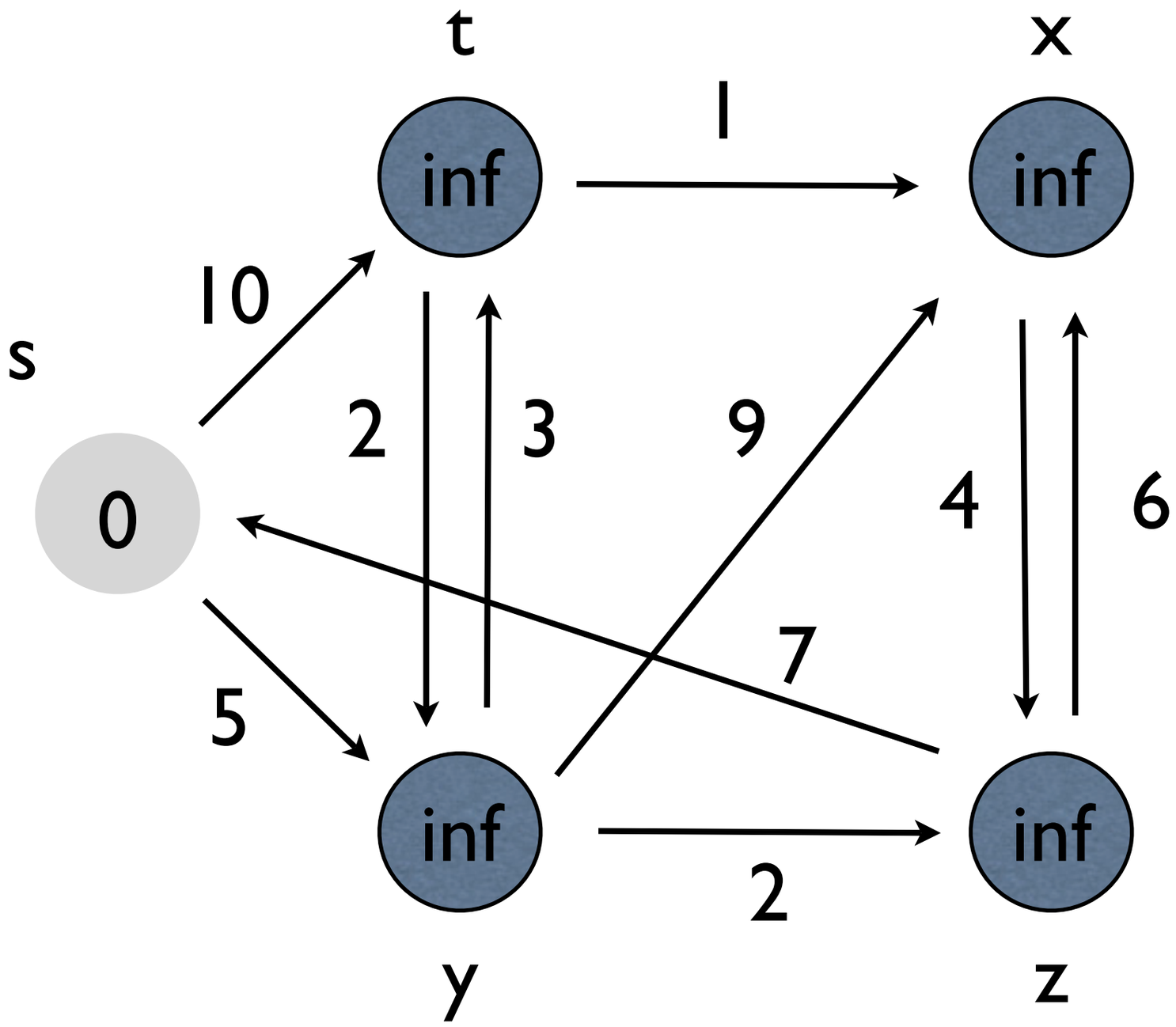
Agenda

- Problema
- Conceitos Básicos
- Djkstra
- Bellman-Ford

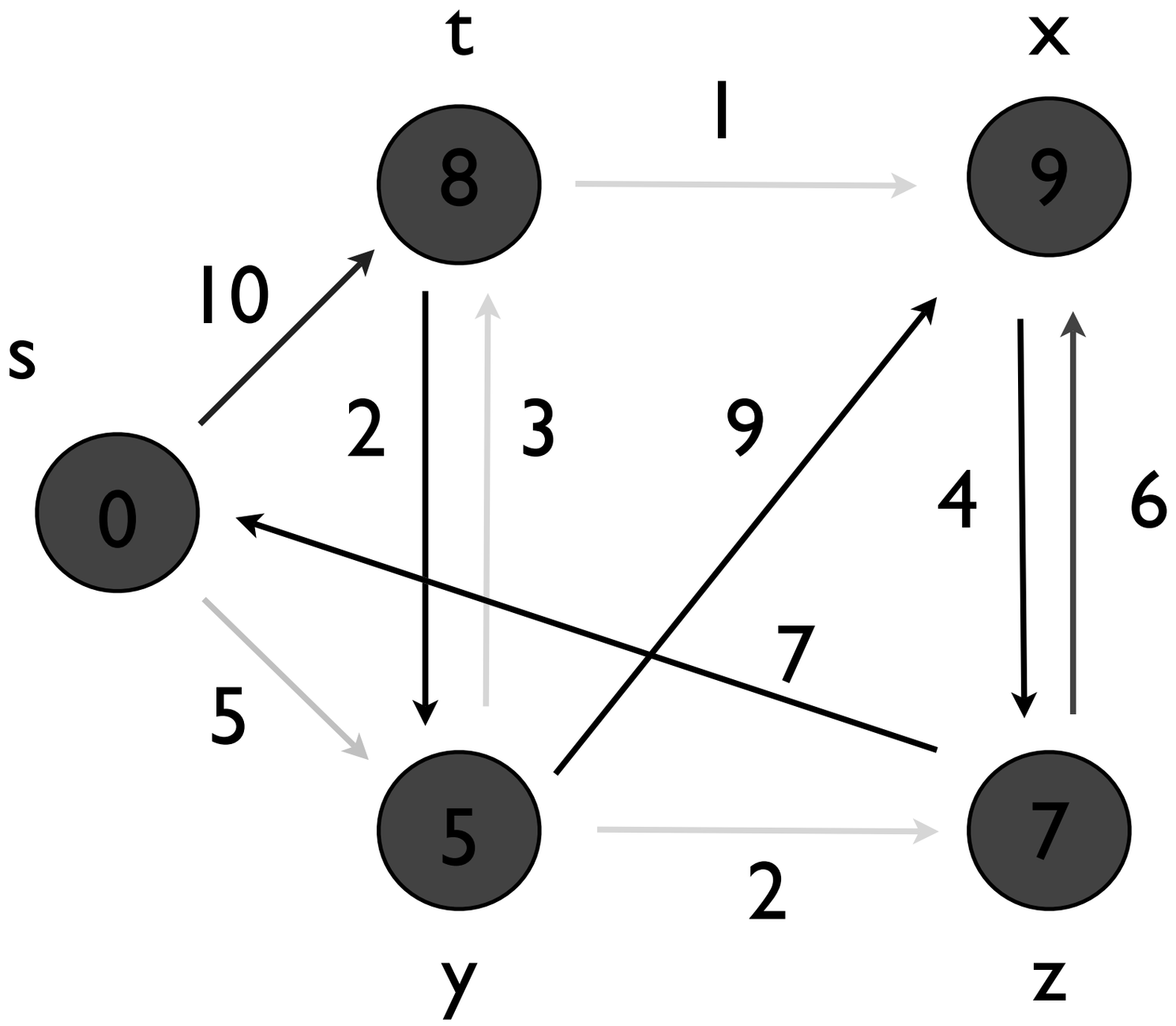
Problema


WA
OR
CA
ID
NV
MT
WY
UT
AZ
CO
NM
ND
SD
NE
KS
OK
TX
MN
IA
MO
AR
LA
WI
IL
MI
IN
OH
KY
TN
MS
AL
GA
WV
VA
NC
SC
PA
NY
VT
NH
ME
MA
RI
CT
NJ
DE
MD
FL
AK
HI
PR
Detroit
Los Angeles
Denver
Wichita
Austin
Miami

Modelagem

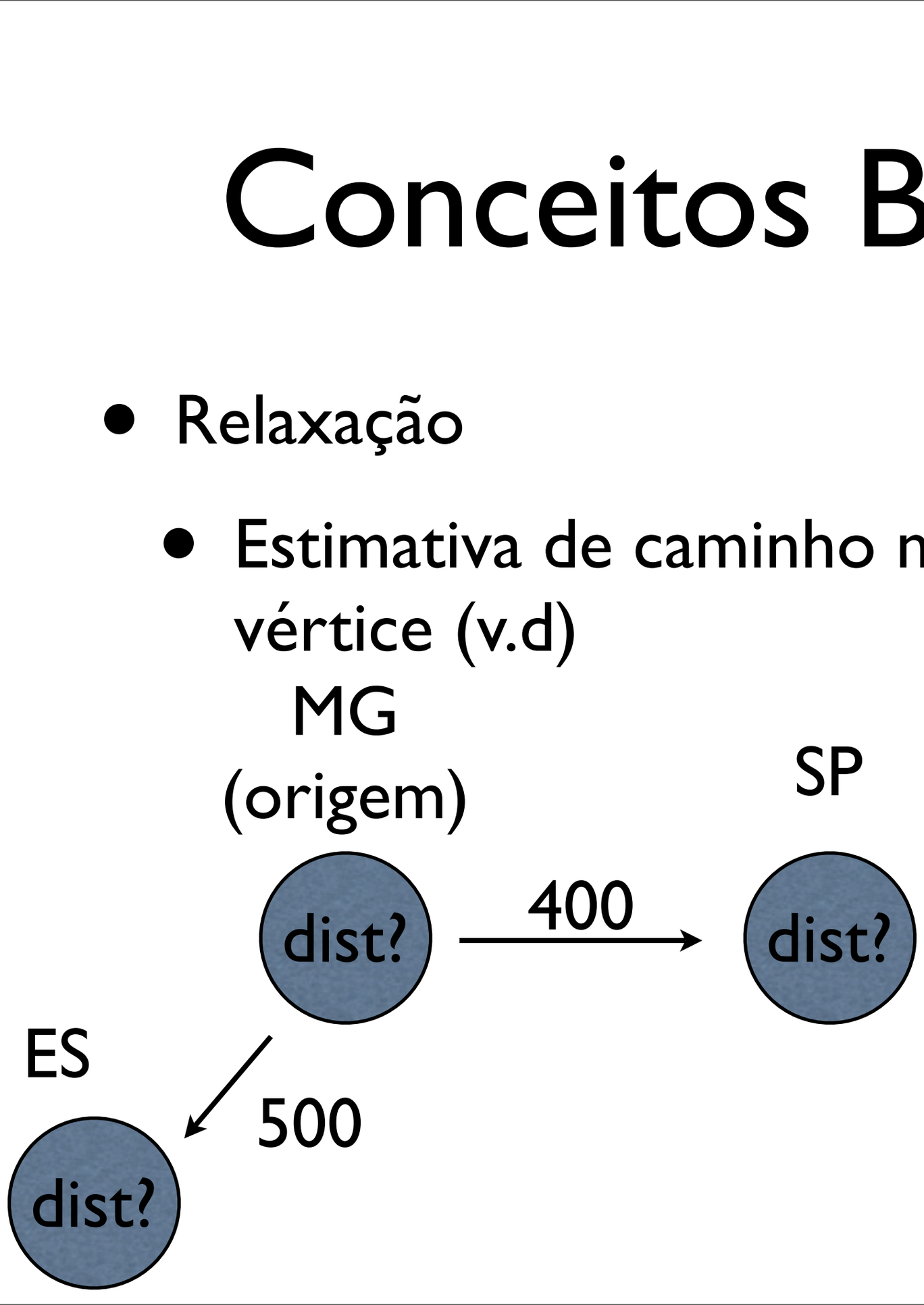


Solução



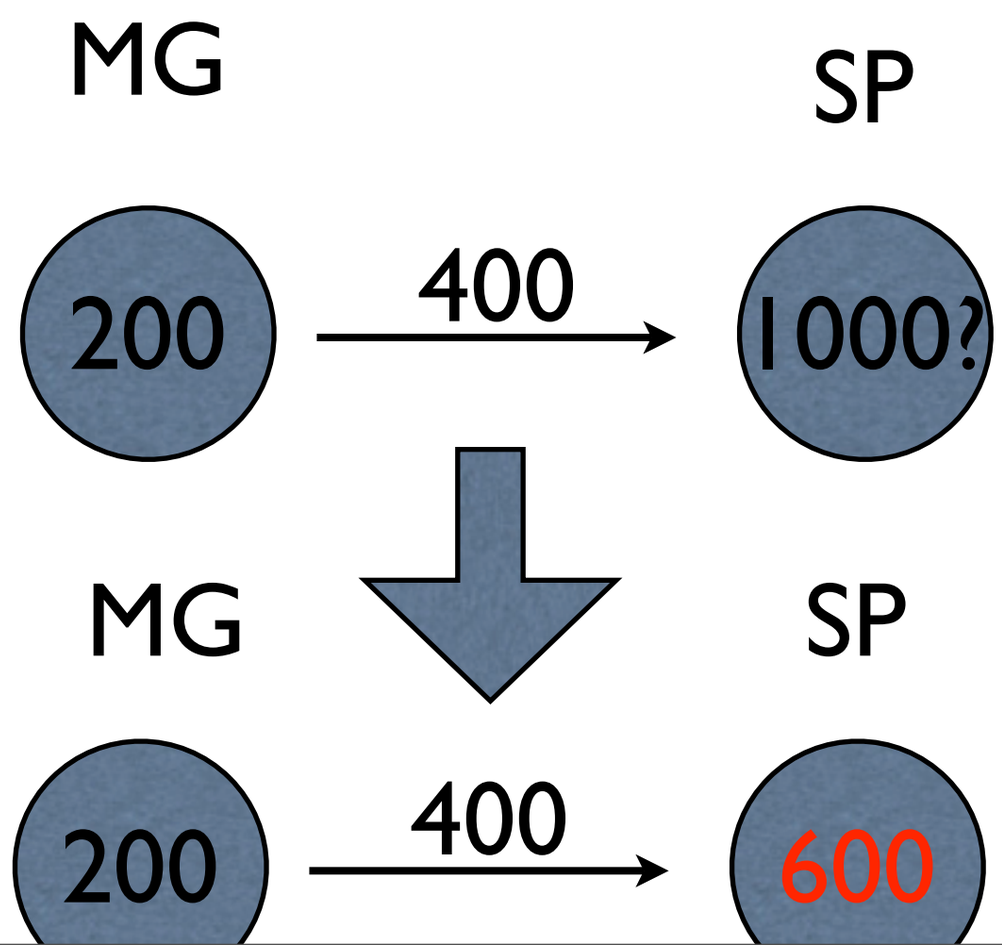
Conceitos Básicos

- Relaxação
  - Estimativa de caminho mínimo para cada vértice (v.d)



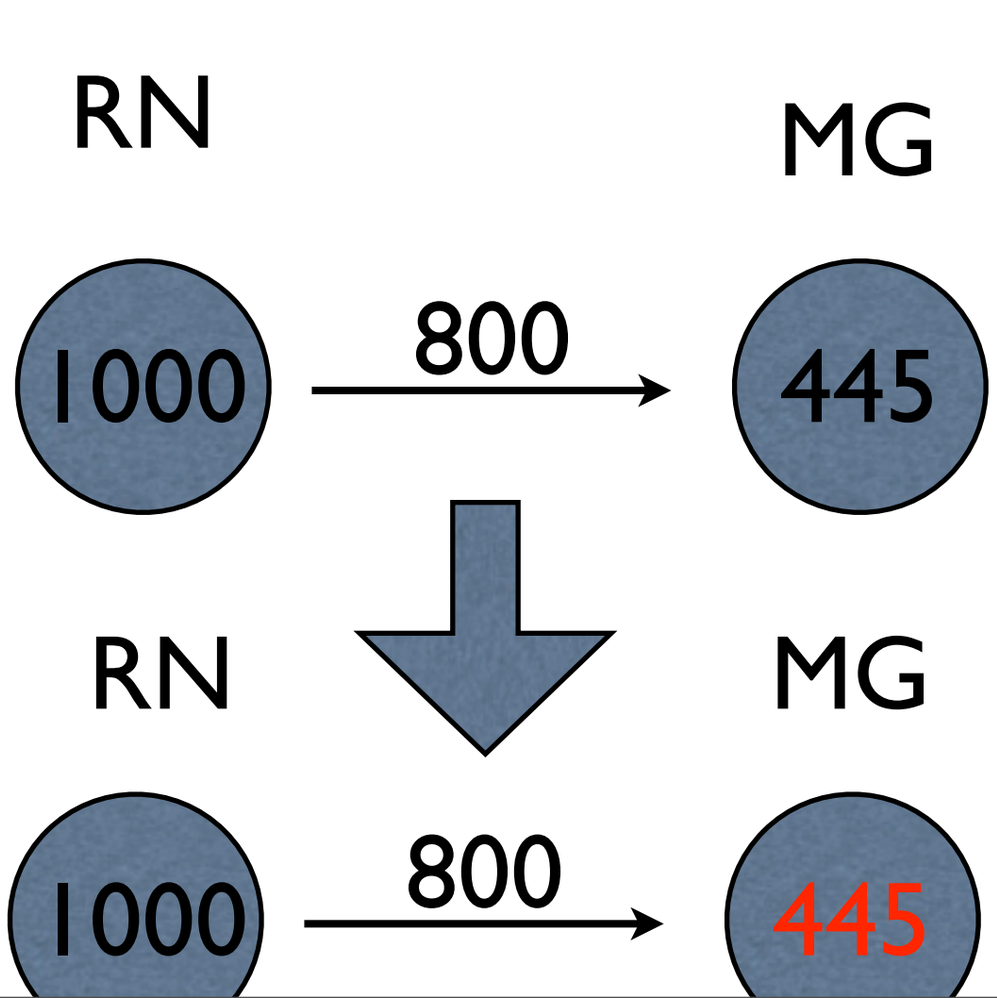
Conceitos Básicos

- Relaxar
  - Estimativa de caminho mínimo para um vértice (v.d)



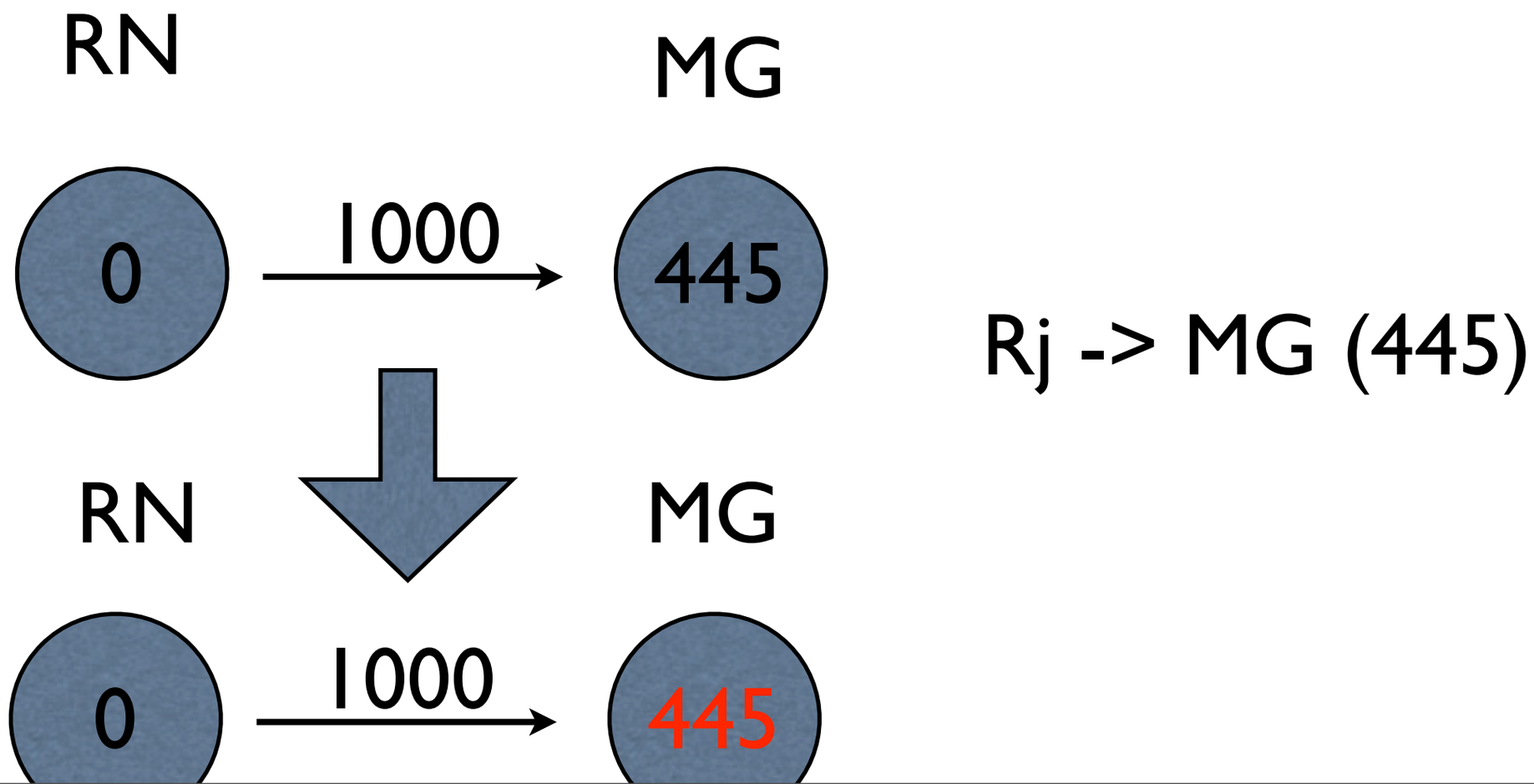
Conceitos Básicos

- Relaxar
  - Estimativa de caminho mínimo para um vértice (v.d)



Conceitos Básicos

- Relaxar
  - Estimativa de caminho mínimo para um vértice (v.d)



Relaxação

Inicializa(G,s)

Para cada v e G.V **Faça**

v.d = inf

v.pred = nil

s.d = 0

Relaxação

Relaxa(u,v,w)

Se u.d + w(u,v) < v.d **Então**

v.d = u.d + w(u,v)

v.pred = u

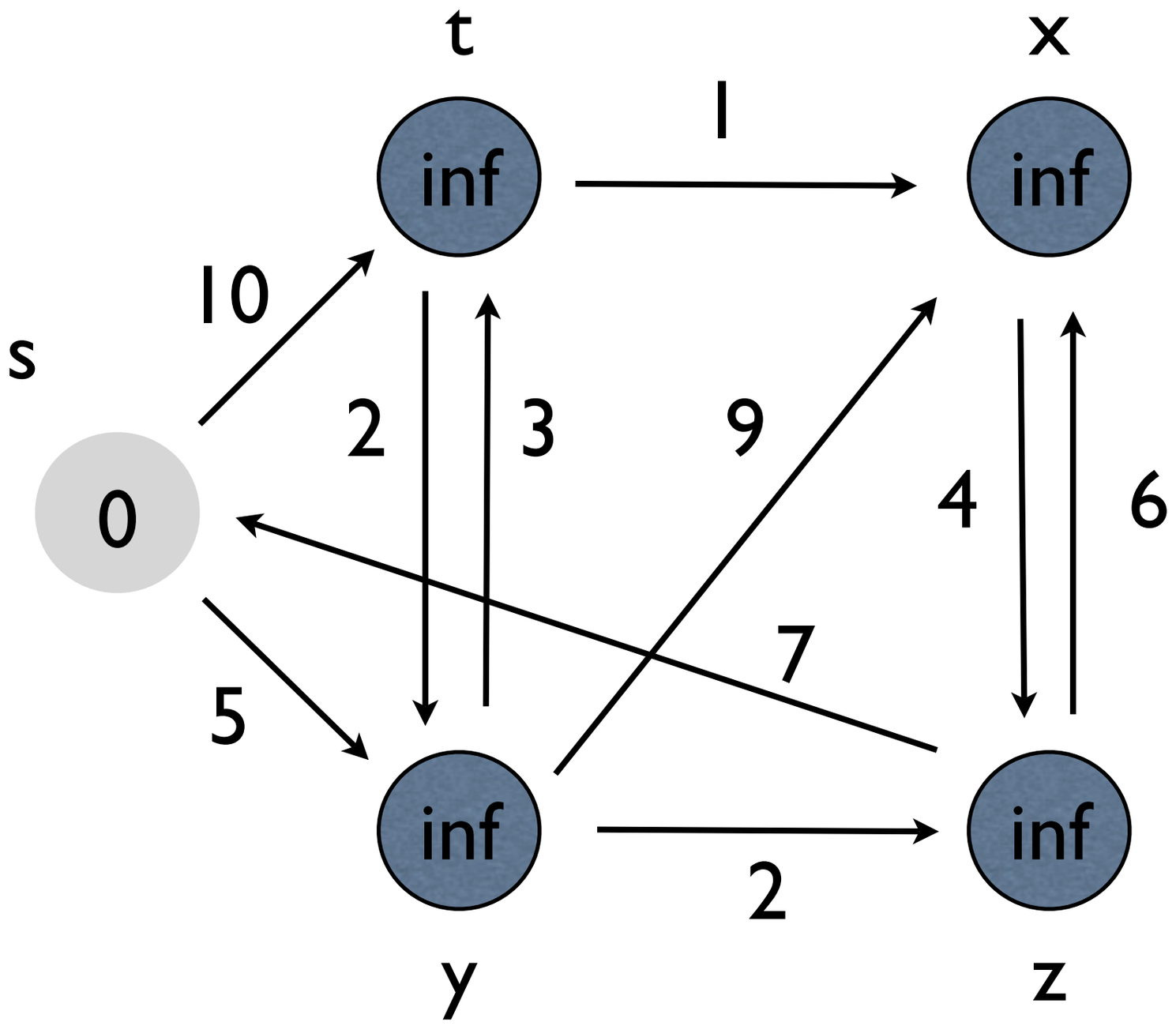
Dijkstra

- Caminho Mínimo
  - Origem até todos os outros vértices
- Grafo Orientado Ponderado
  - Pesos não-negativos
- Grafo pode ter ciclos

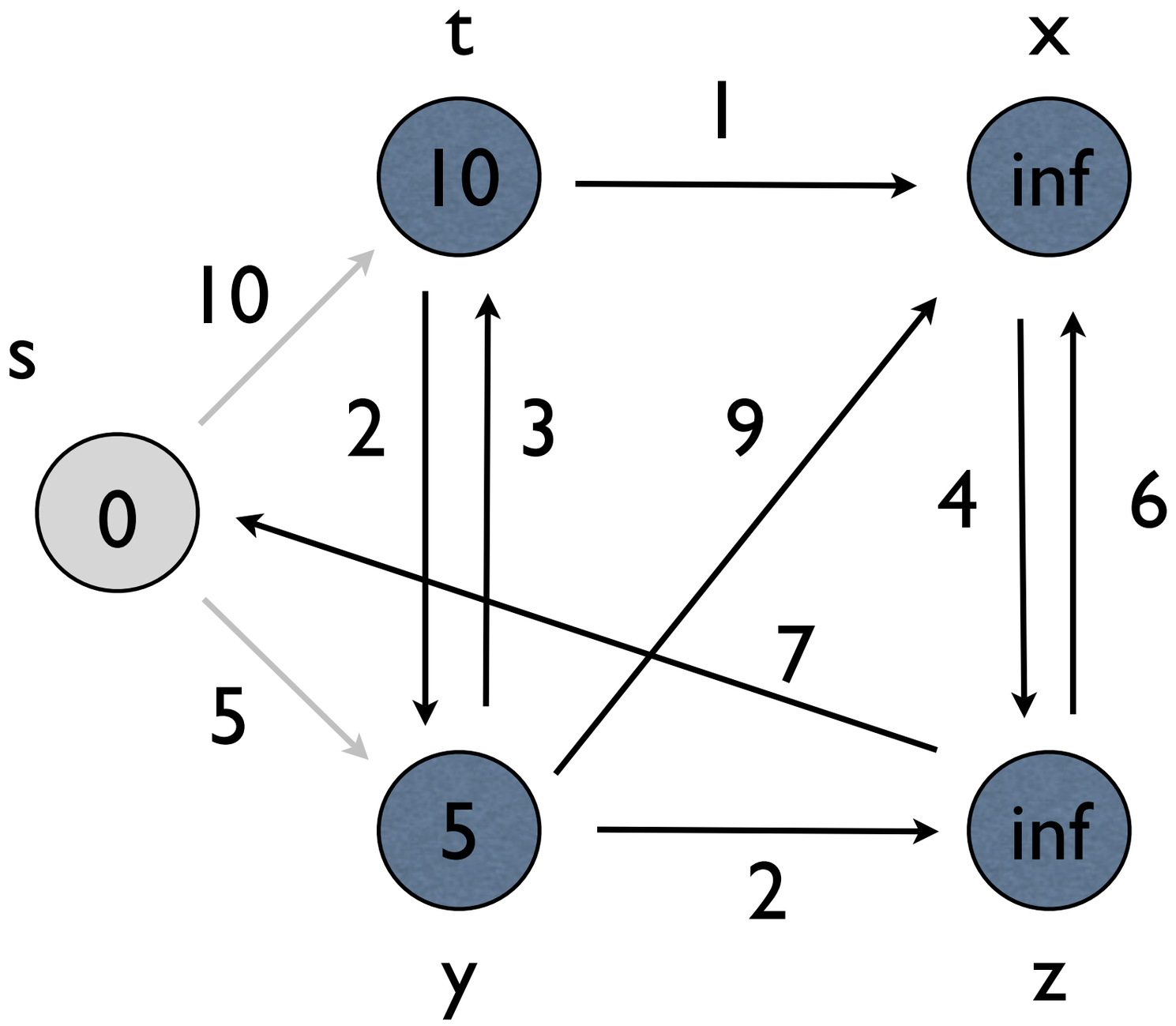
Dijkstra

- Como funciona?
  - Coloca todos os nós organizados numa **fila de prioridades**
    - A prioridade é a distância
  - A cada passo um nó é visitado
    - O nó visitado é sempre o que possui **menor prioridade**
    - Os vizinhos deste são **relaxados**

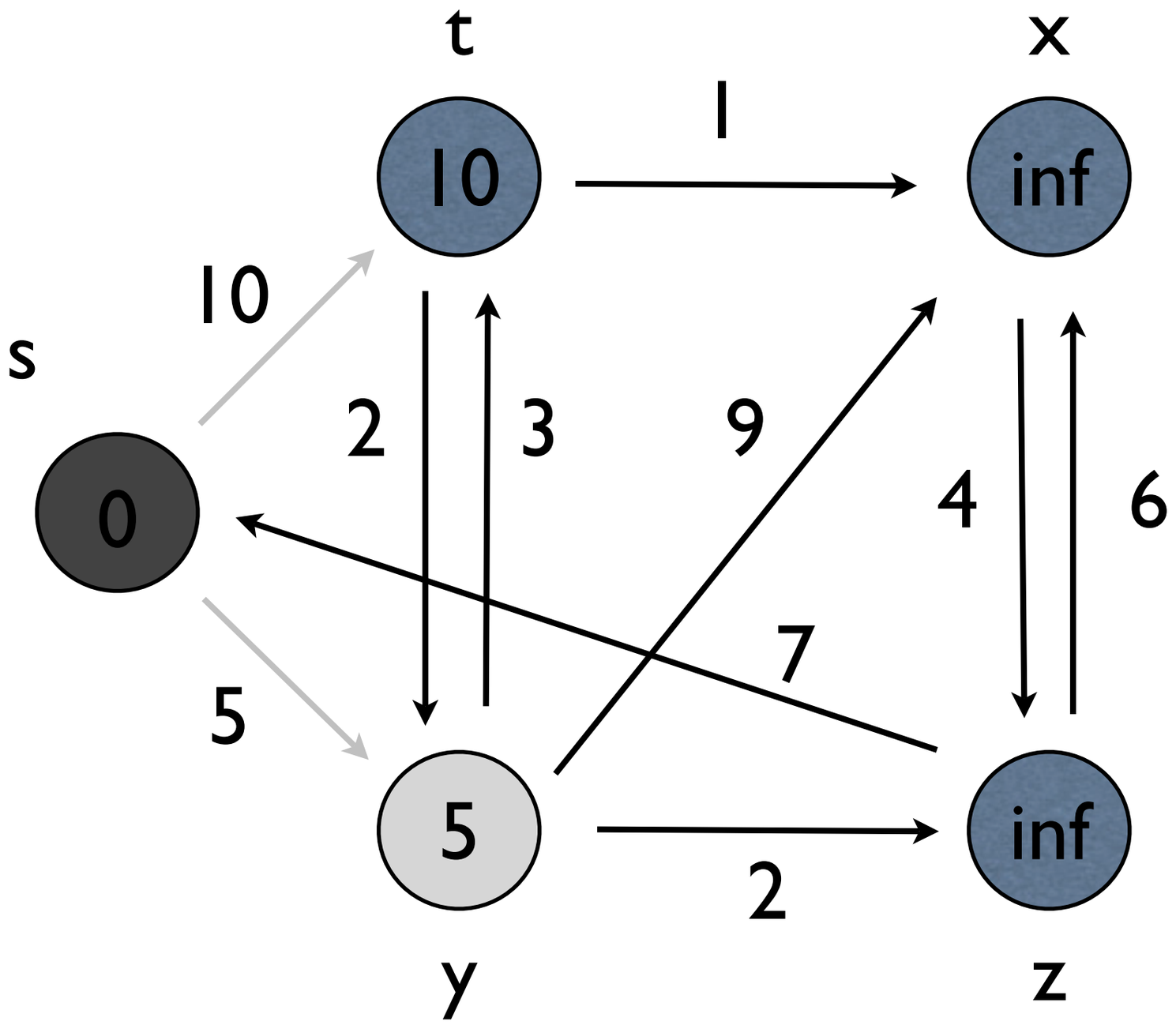
Dijkstra



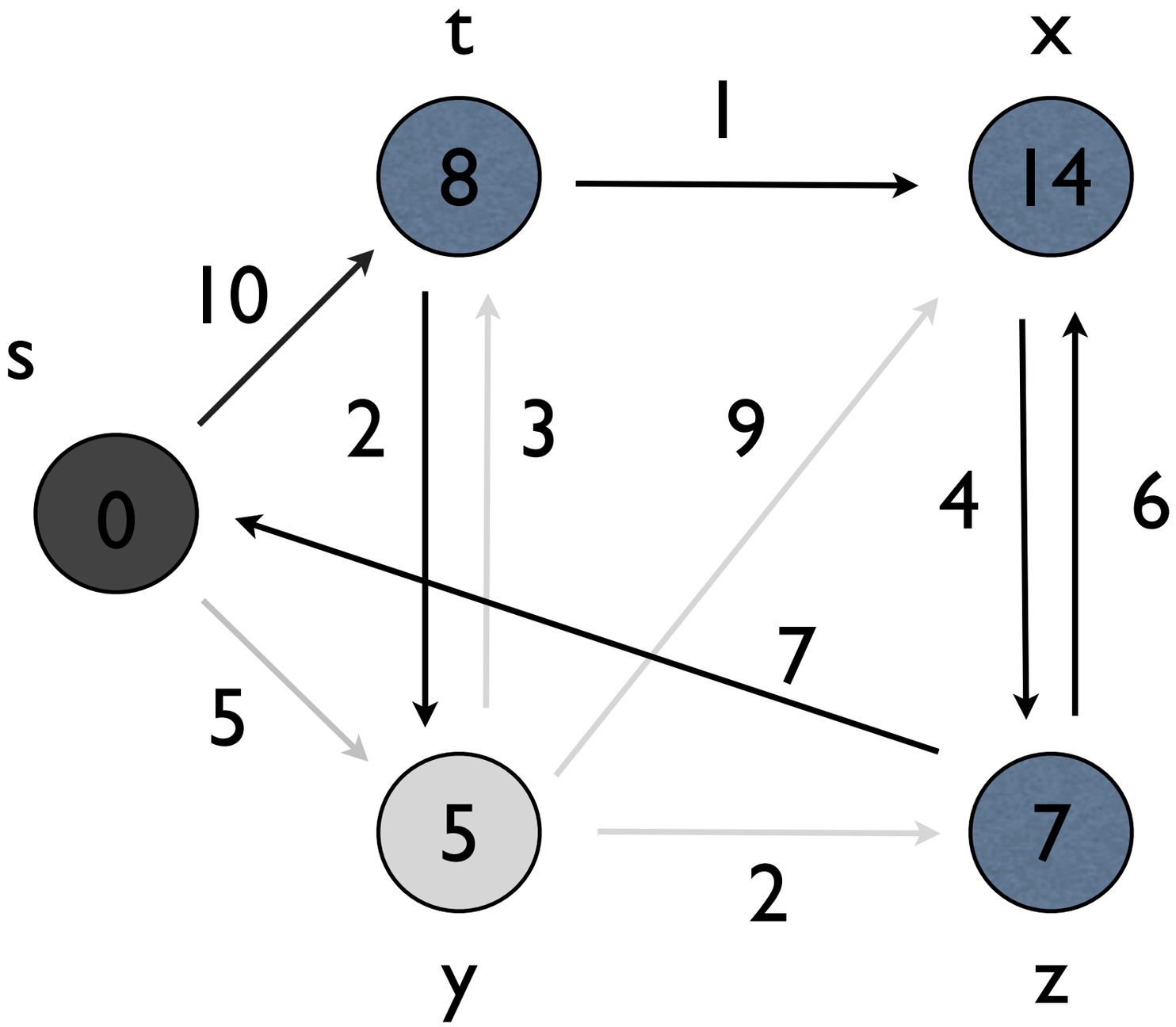
Dijkstra



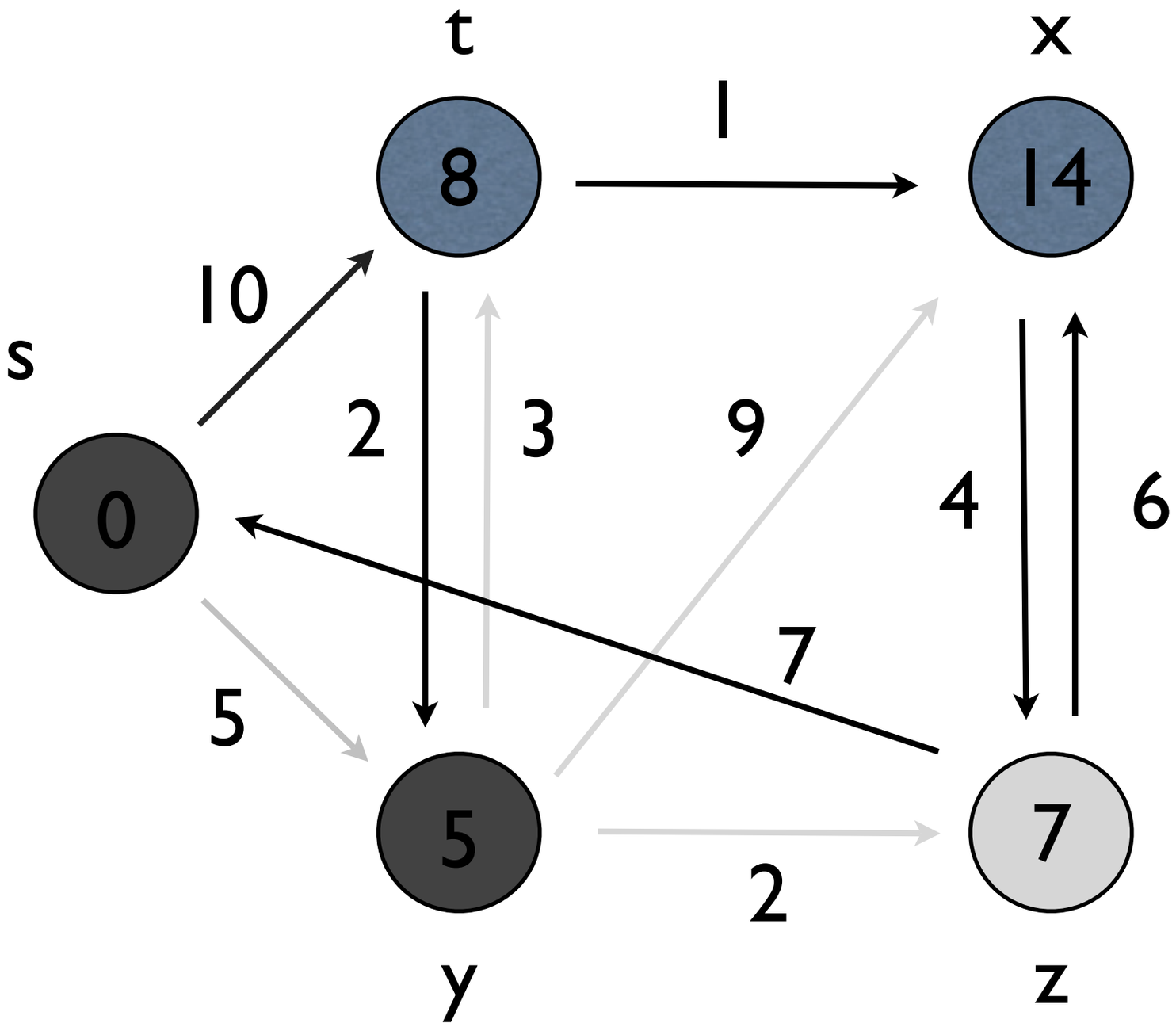
Dijkstra



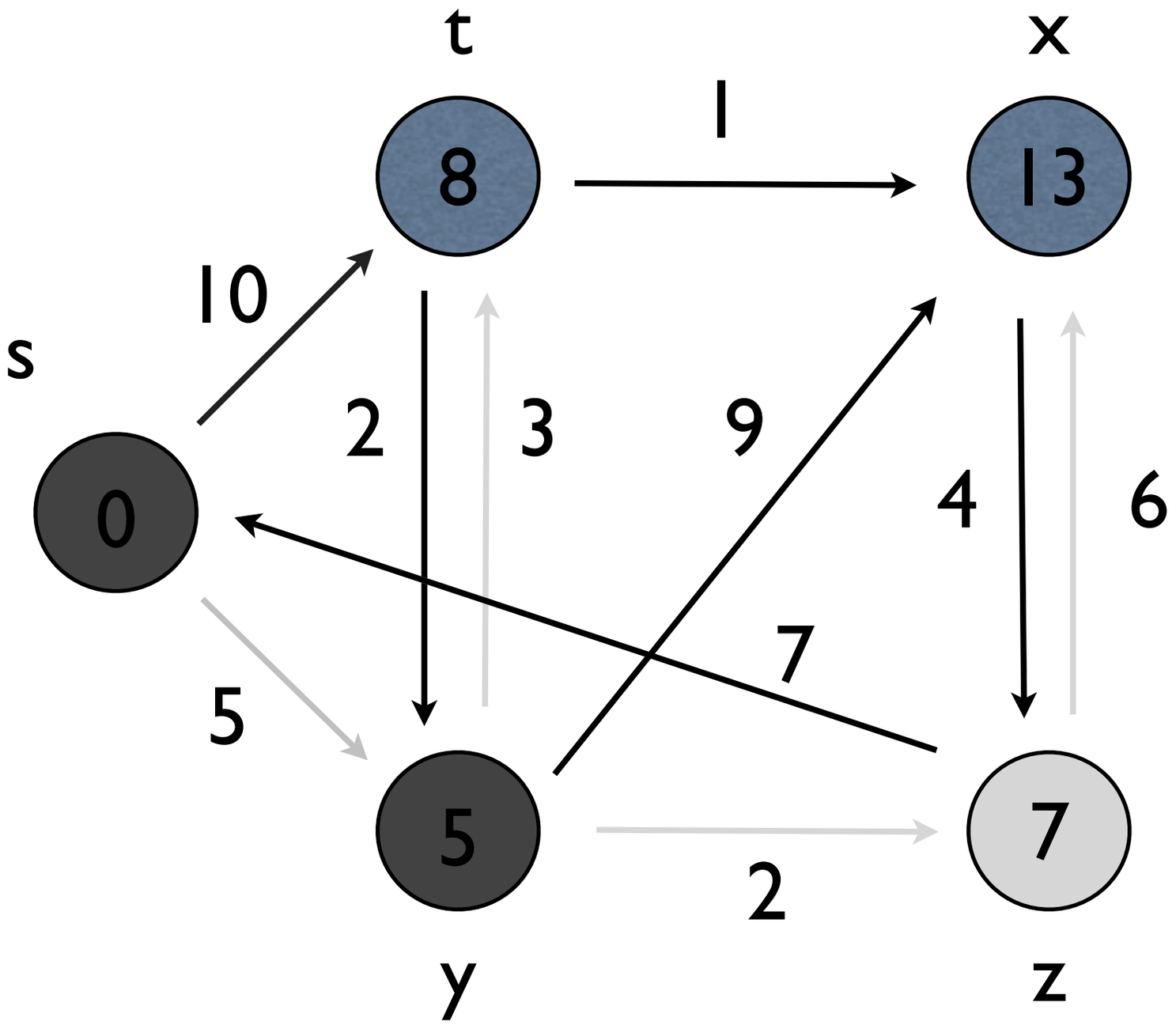
Dijkstra



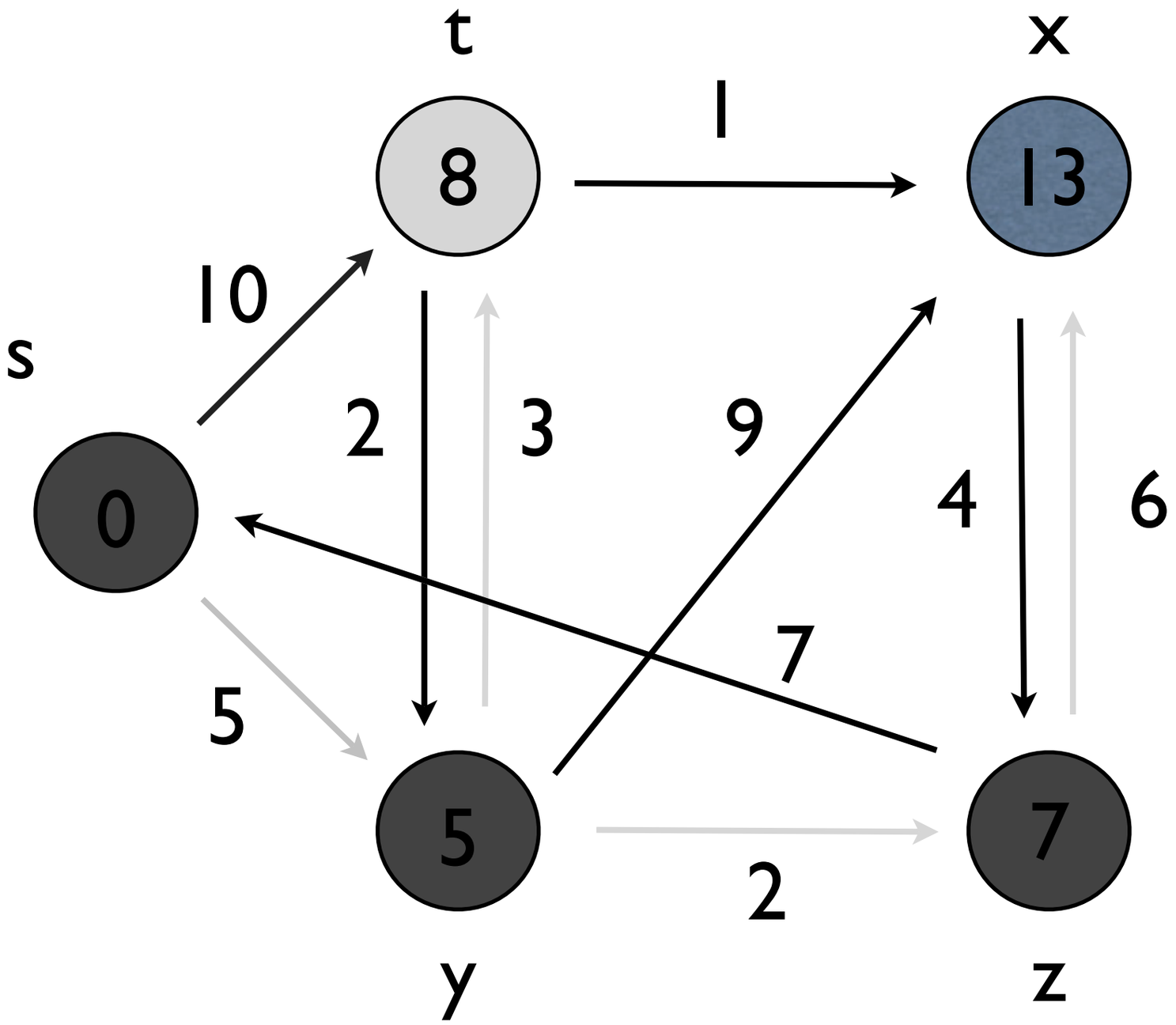
Dijkstra



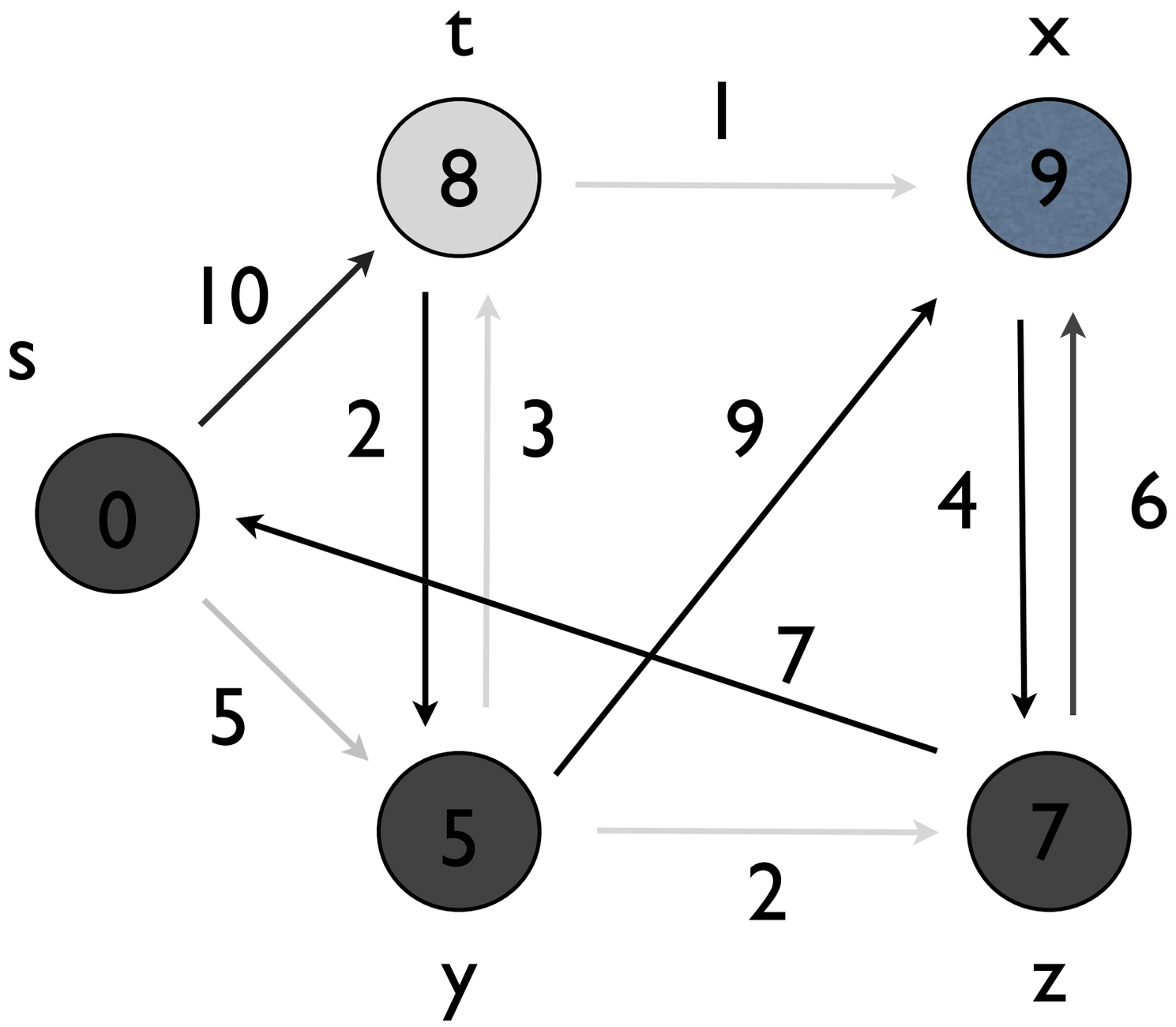
Dijkstra



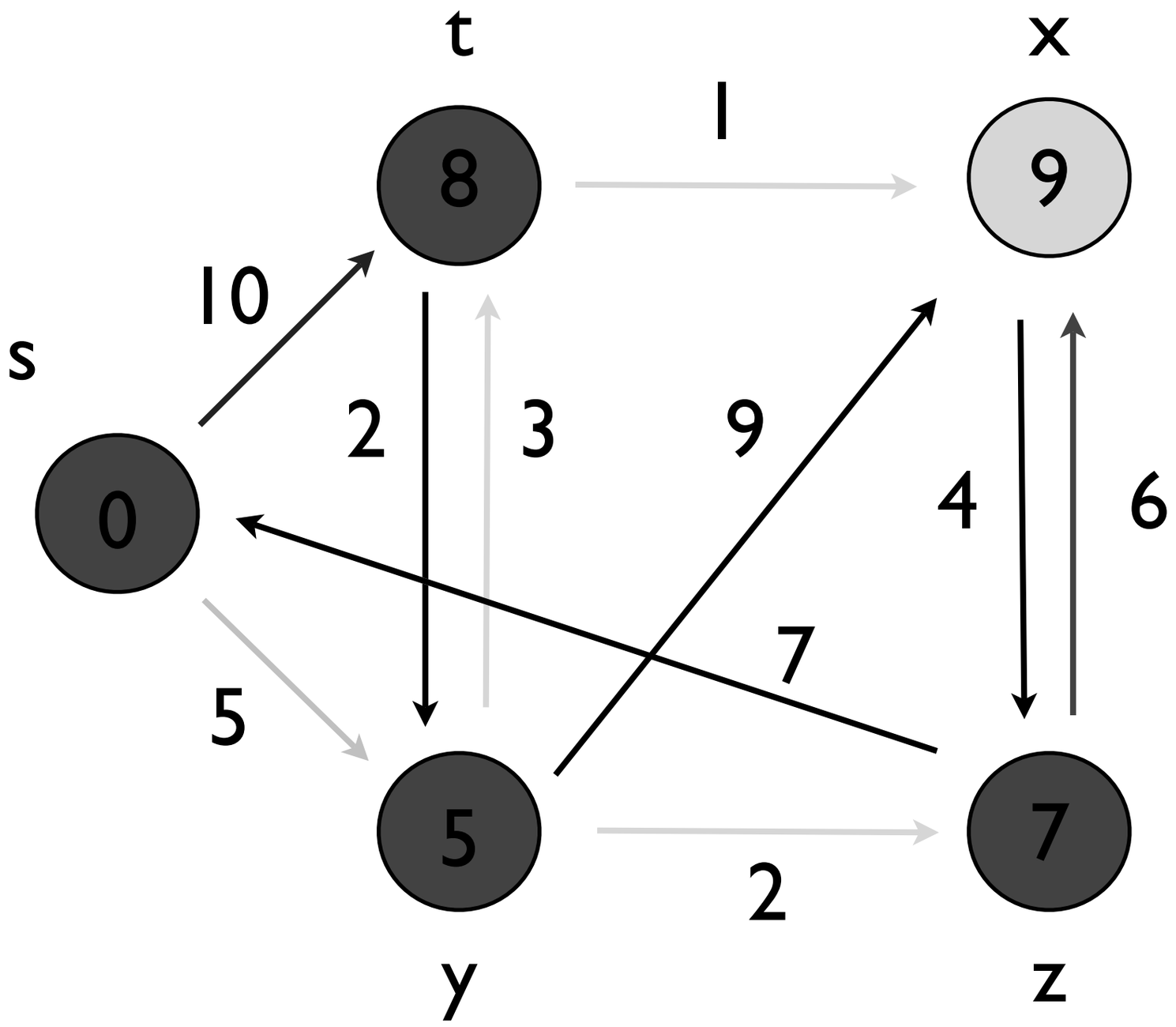
Dijkstra



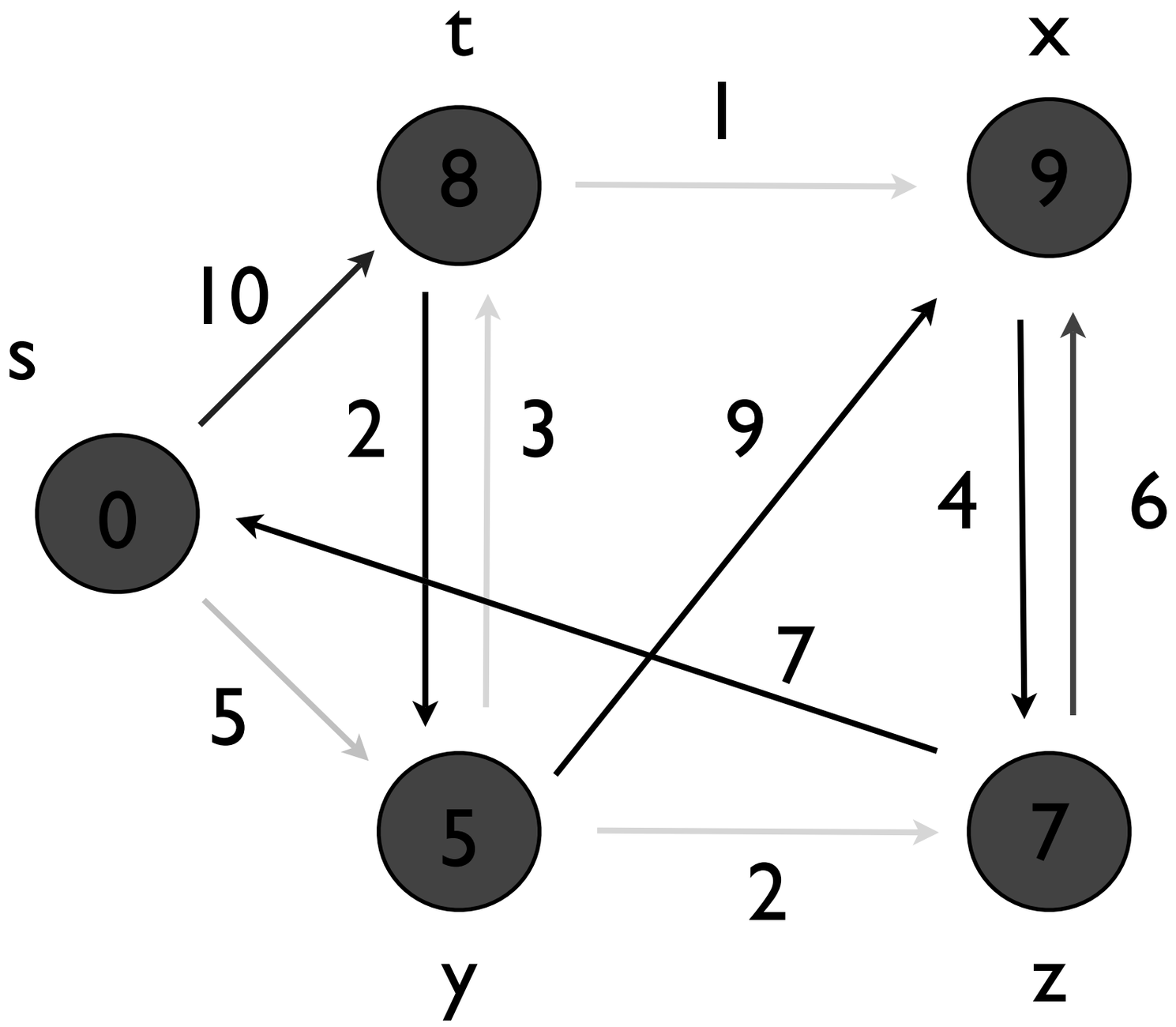
Dijkstra



Dijkstra



Dijkstra



Algoritmo

```
Dijkstra(G,s)
        Inicializa(G,s)
        Fila = G.V
        Enquanto Fila != {} Faça
            u = minimo(Fila)
             Para cada v e G.adj[u] Faça
                relaxa(u,v,w)
```

Análise

- O tempo de execução depende da implementação fila de prioridades e do número de arestas
- Tempo
  - O(E x O(relaxa) + V x O(mínimo))

Análise

```
Dijkstra(G,s)
    Inicializa(G,s) // O(V)
    Fila = G.V // lista O(V)
Enquanto Fila != {} Faça
    u = minimo(Fila) // lista O(V)
    Para cada v e G.adj[u] Faça
        relaxa(u,v,w) // O(1)
```

Análise

- O tempo de execução depende da implementação fila de prioridades e do número de arestas
- Grafo
  - Fila implementada por lista
    - mínimo O(V)
    - V x mínimo() = O(V^2)
  - Tempo total
    - O(E+V^2) = O(V^2)

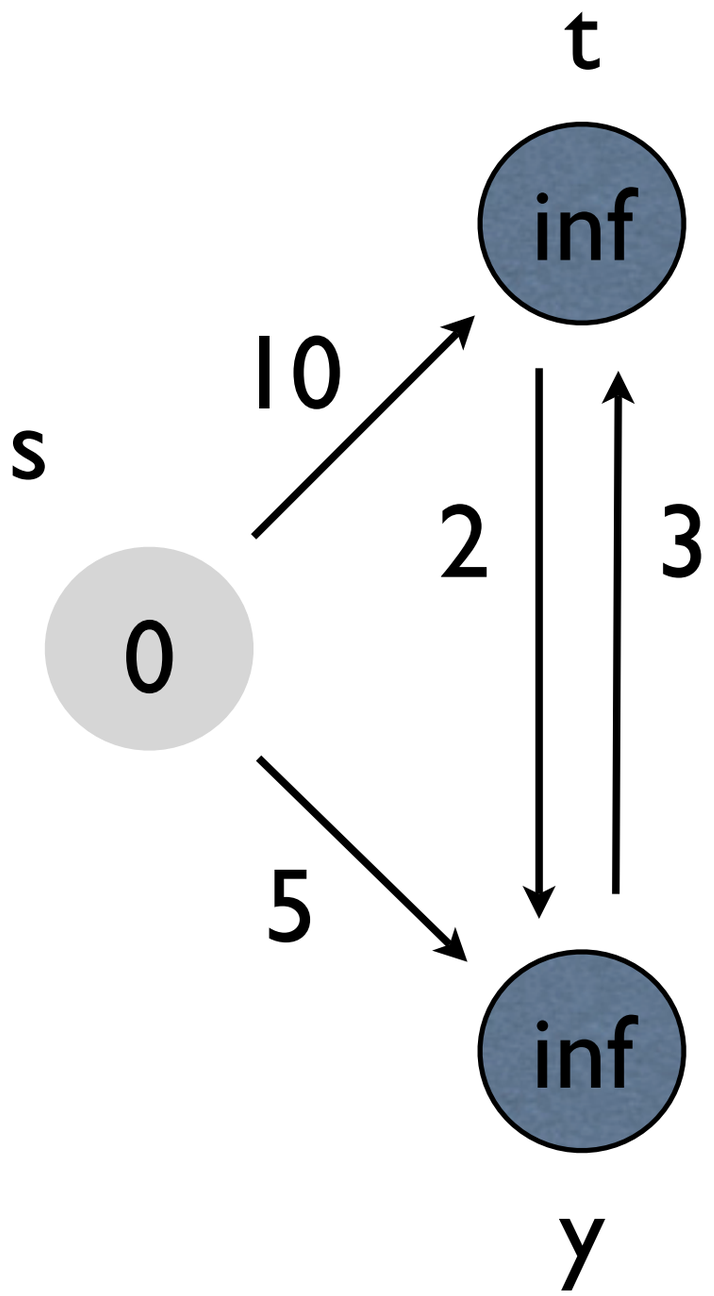
Análise

- O tempo de execução depende da implementação fila de prioridades e do número de arestas
- Fila implementada por Heap
  - Mínimo: O(log(V))
  - Total: O((E+V)log(V)) = O(E log(V))
- Fila implementada por Heap Fibonacci
  - Total: O(E + Vlog(V))

Bellman-Ford

- Caminho Mínimo
  - Origem até todos os outros vértices
- Grafo Orientado Ponderado
  - Aceita Pesos negativos
    - Sem ciclos de peso negativo

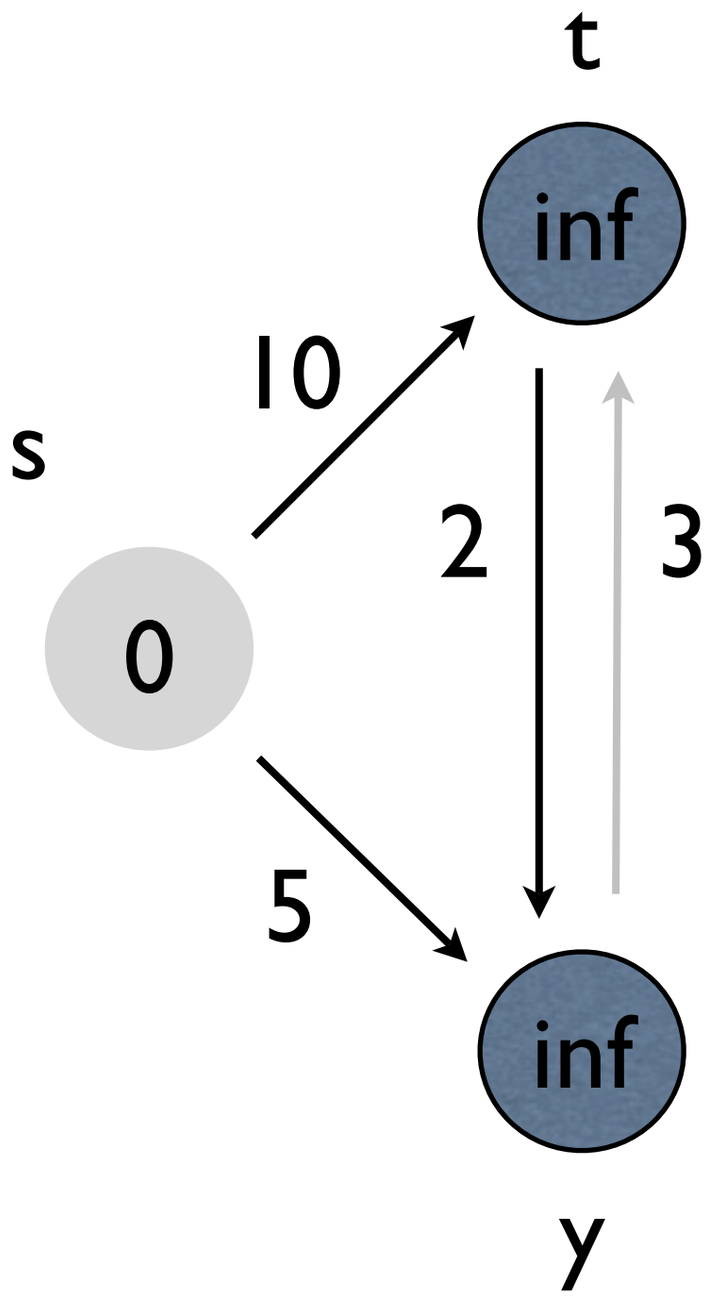
Bellman-Ford



Relaxe |V|-1 vezes todas as arestas

Escolha aleatória das arestas

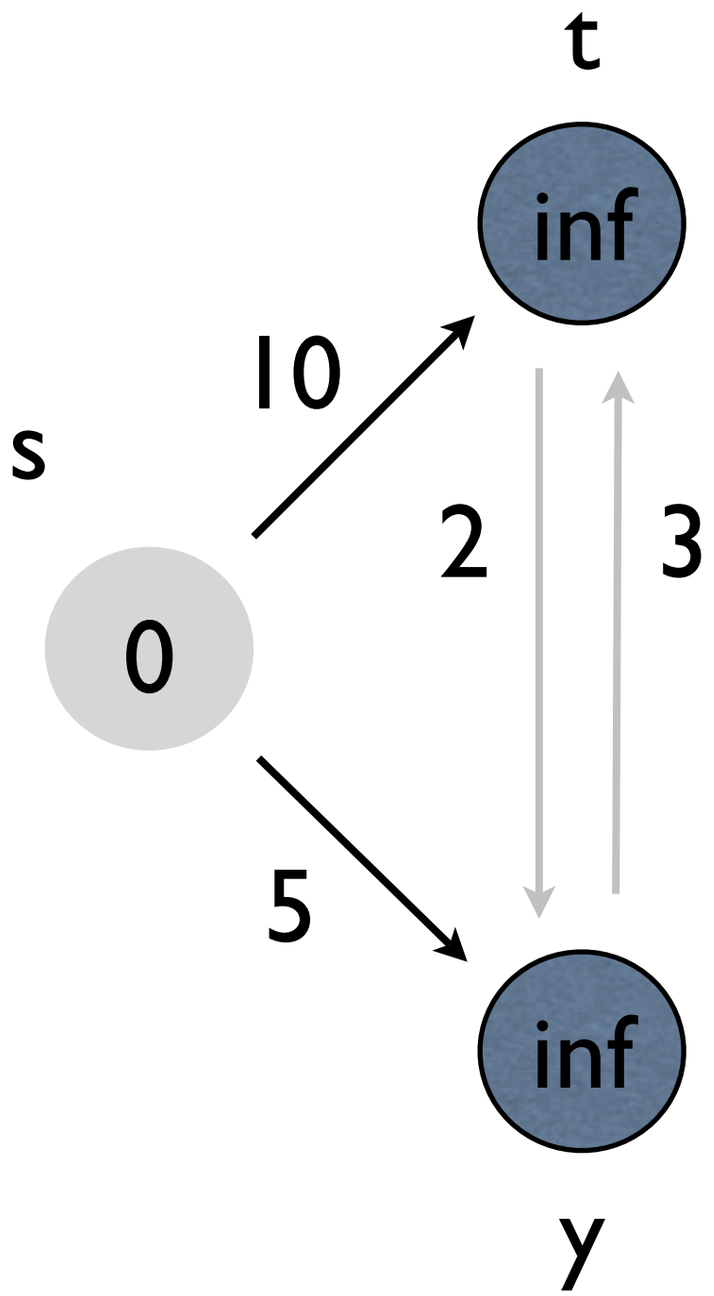
Bellman-Ford



Relaxe |V|-1 vezes todas as arestas

Escolha aleatória das arestas

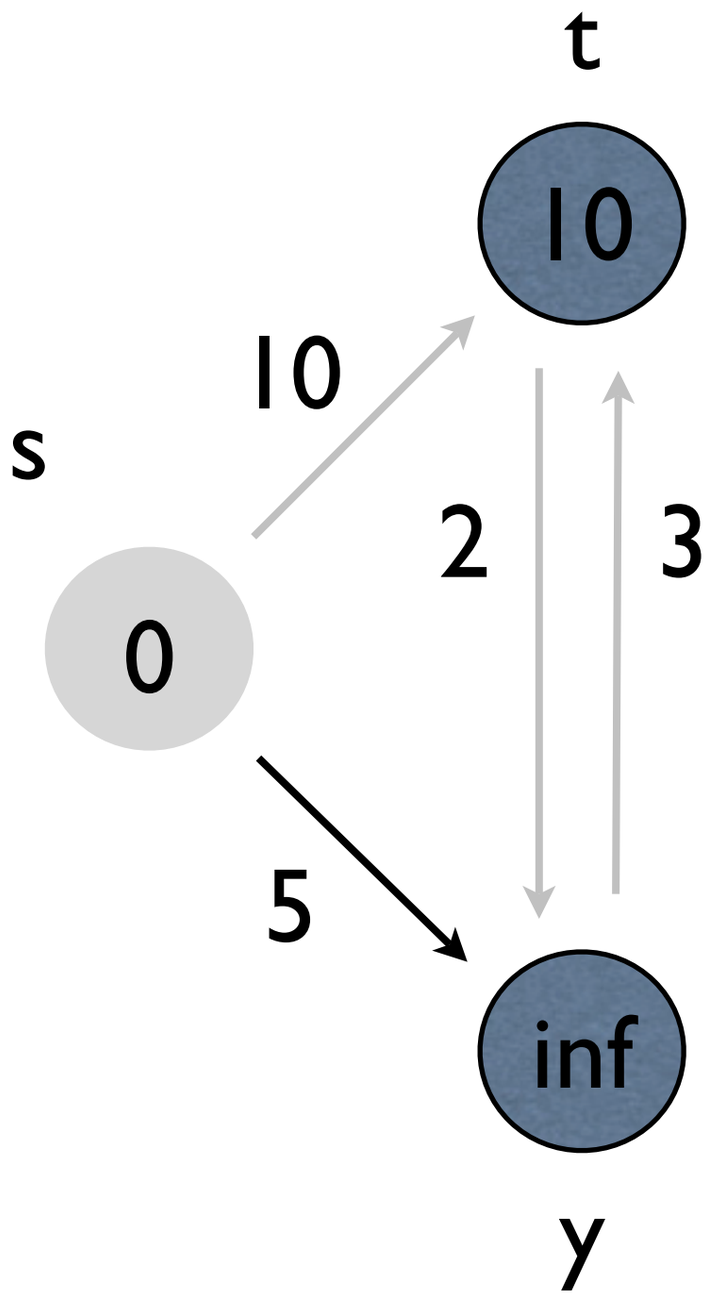
Bellman-Ford



Relaxe |V|-1 vezes todas as arestas

Escolha aleatória das arestas

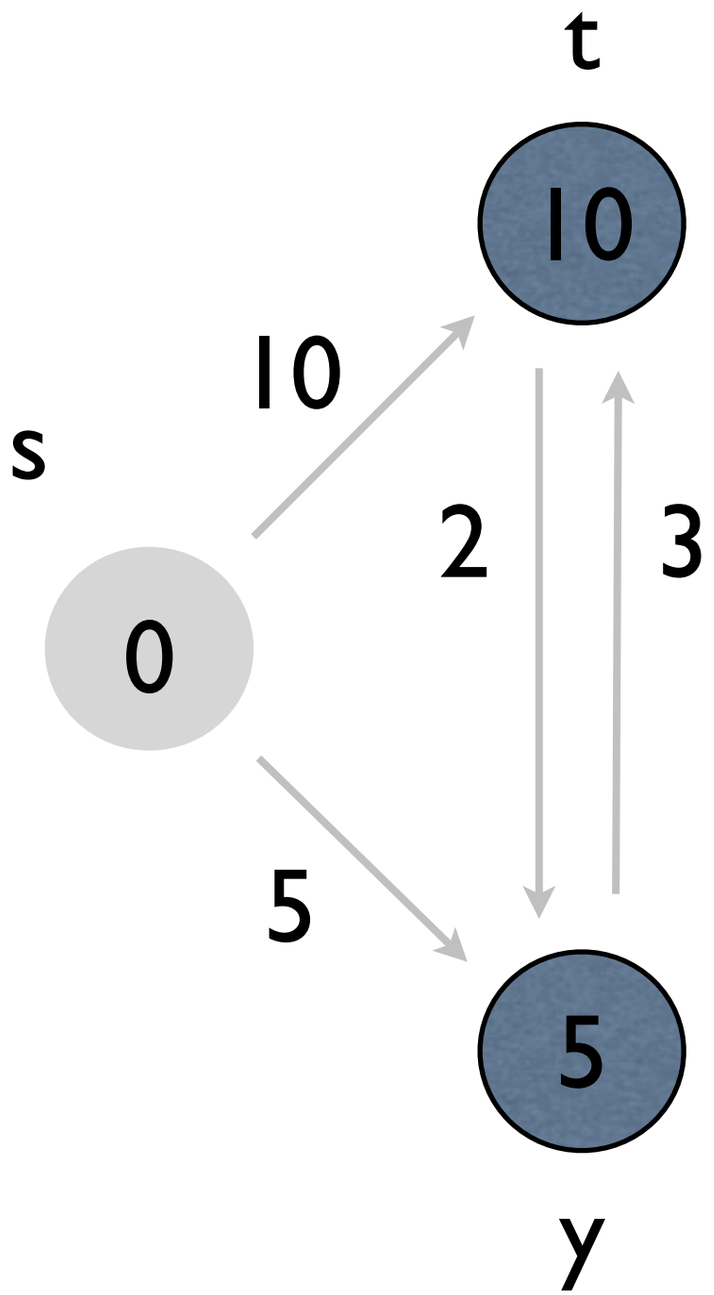
Bellman-Ford



Relaxe |V|-1 vezes todas as arestas

Escolha aleatória das arestas

Bellman-Ford

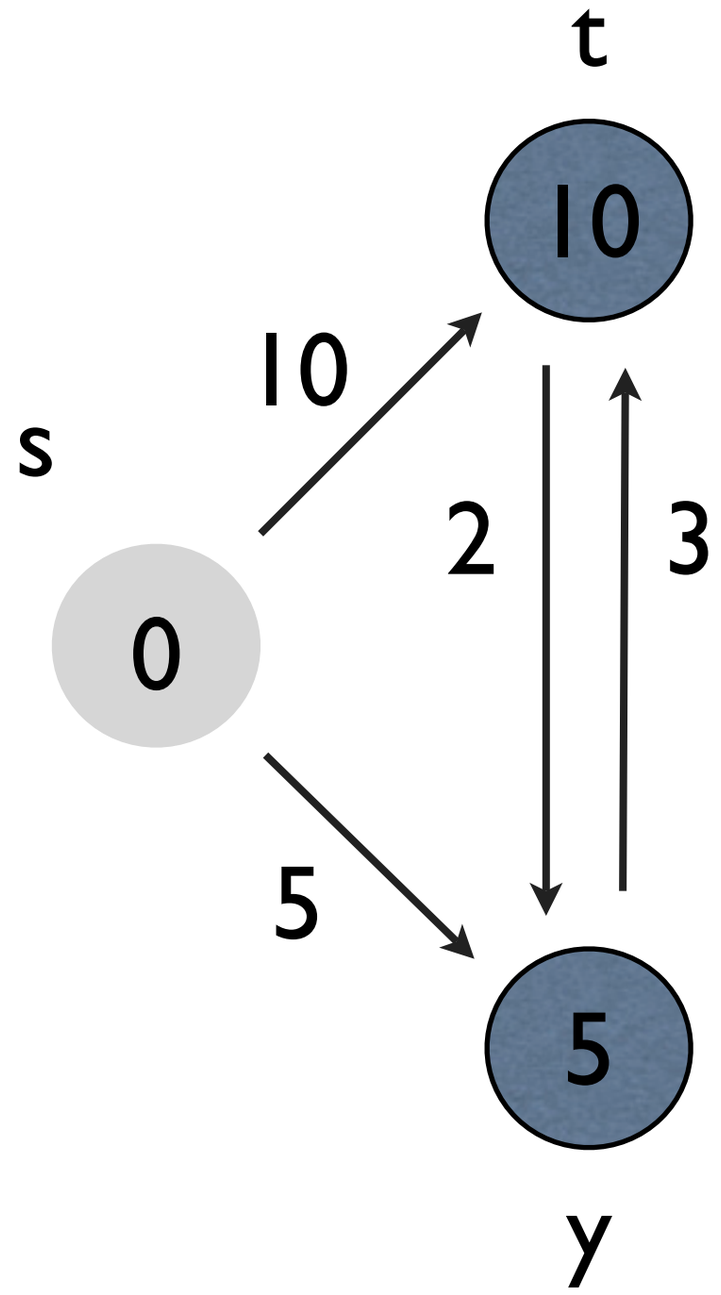


Relaxe |V|-1 vezes todas as arestas

Escolha aleatória das arestas

Devemos fazer todas as relaxações mais uma vez

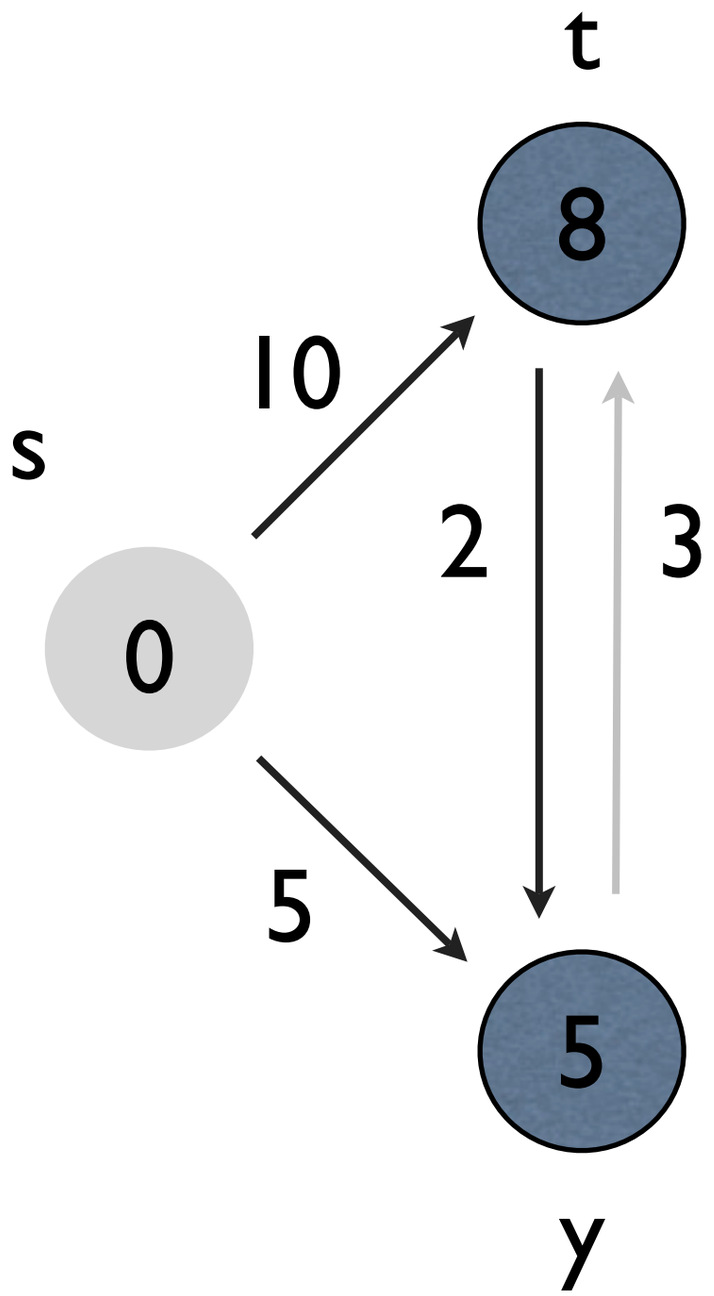
Bellman-Ford



Relaxe |V|-1 vezes todas as arestas

Escolha aleatória das arestas

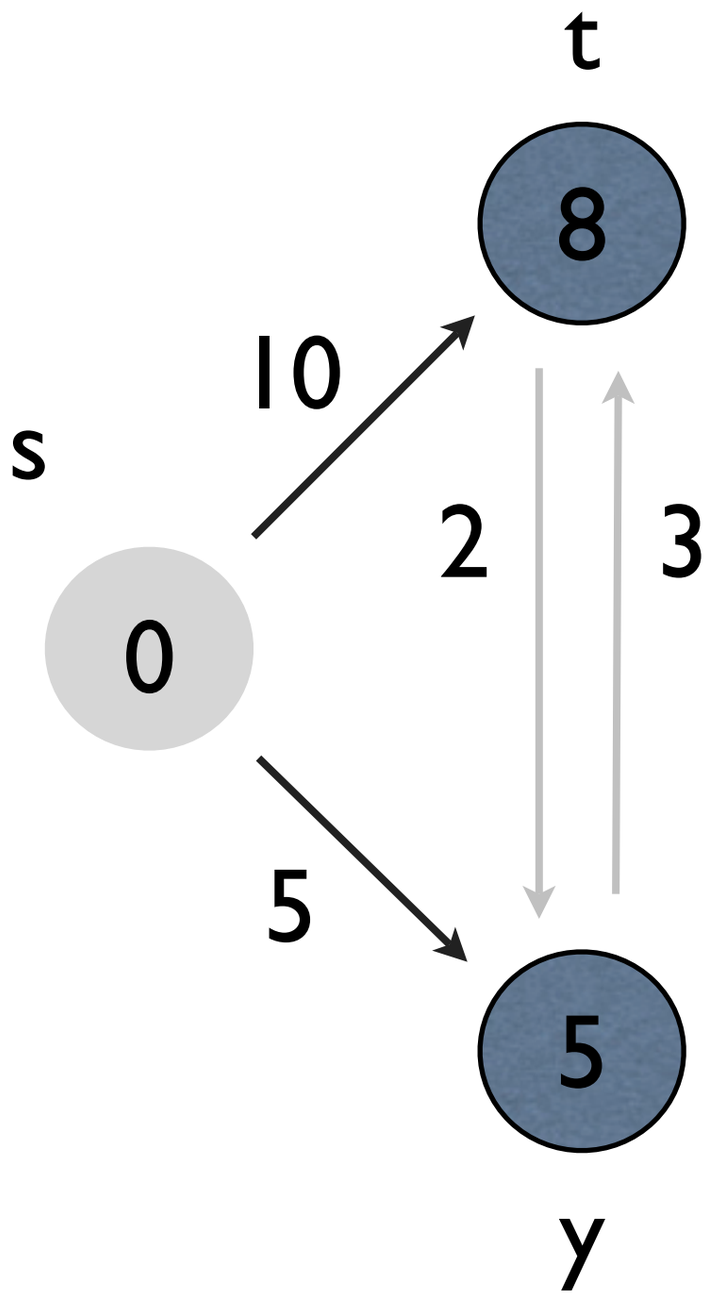
Bellman-Ford



Relaxe |V|-1 vezes todas as arestas

Escolha aleatória das arestas

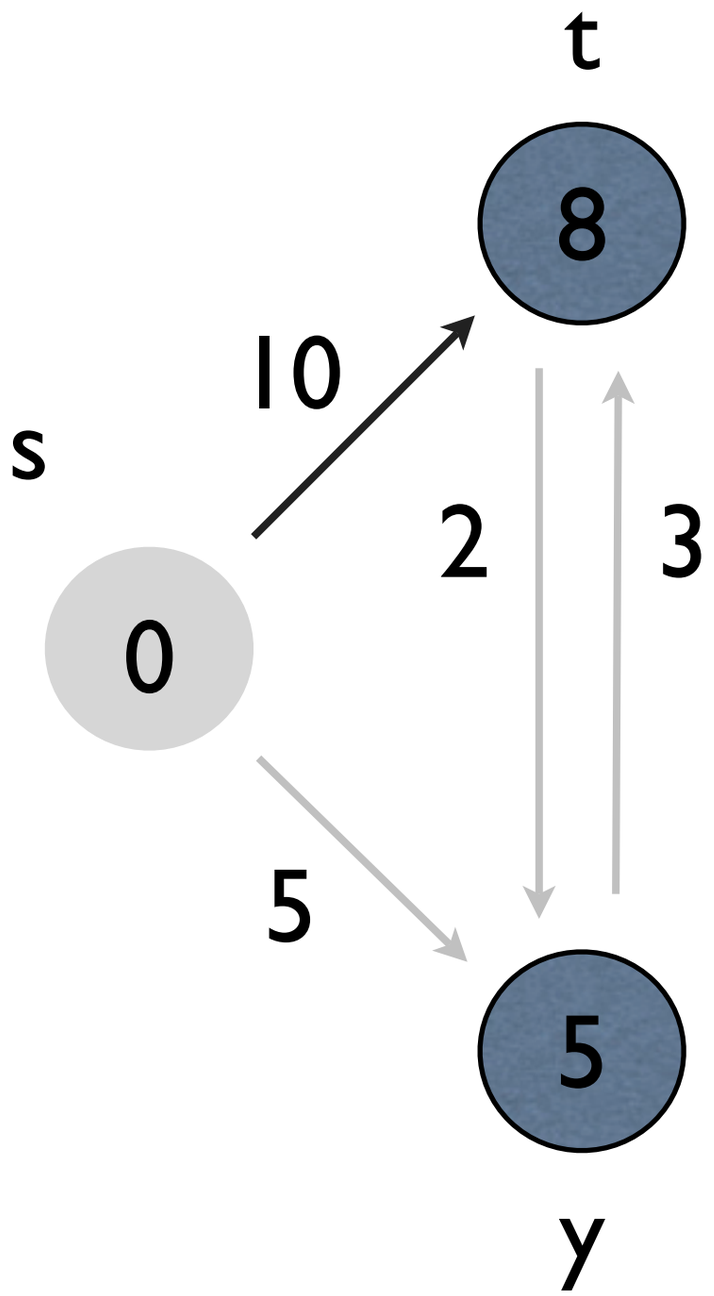
Bellman-Ford



Relaxe |V|-1 vezes todas as arestas

Escolha aleatória das arestas

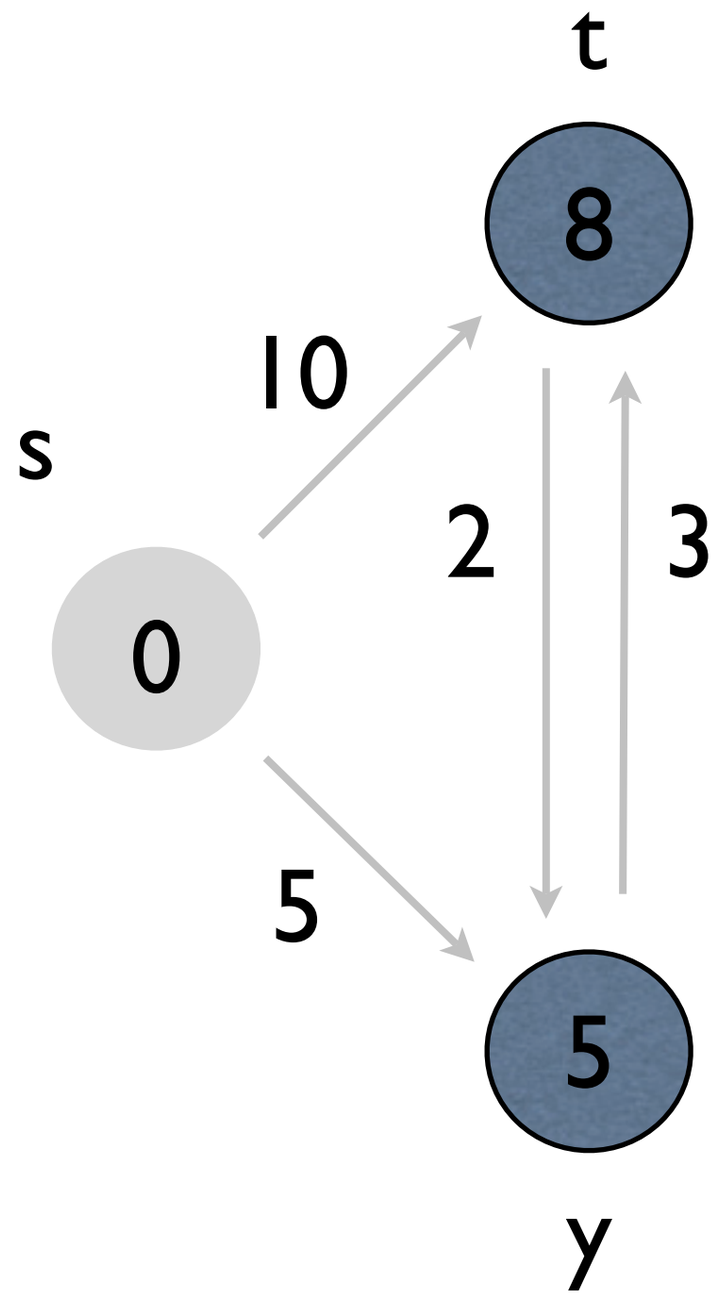
Bellman-Ford



Relaxe |V|-1 vezes todas as arestas

Escolha aleatória das arestas

Bellman-Ford



Relaxe |V|-1 vezes todas as arestas

Escolha aleatória das arestas

Agora temos as distâncias!

Bellman-Ford

```
Bellman-Ford(G,s)
    Inicializa(G,s)
    Para i=1 até |V|-1 Faça
        Para cada (u,v) e G.E Faça
            relaxa(u,v,w)
```

Bellman-Ford

```
Bellman-Ford(G,s)

    Inicializa(G,s)

    Para i=1 até |V|-1 Faça

        Para cada (u,v) e G.E Faça

            relaxa(u,v,w)


// verificando ciclos negativos (u -> v)

Para cada (u,v) e G.E Faça

    Se  u.dist + w(u,v) < v.dist Então

        imprimir "Ciclo Negativo"
```

Análise

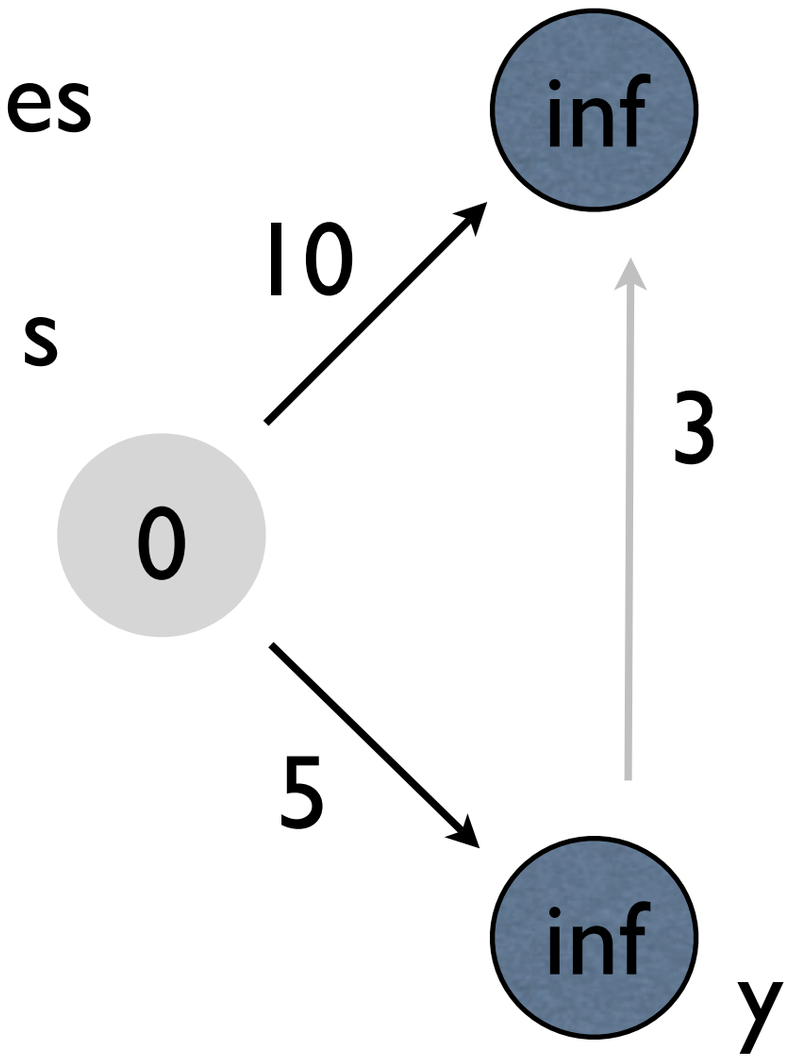
- Tempo O(VE)

Bellman-Ford (DAG)

- Achar caminho mínimo
  - Grafo Dirigido Acíclico
  - Aceita pesos negativos
  - Tempo O(V+E)

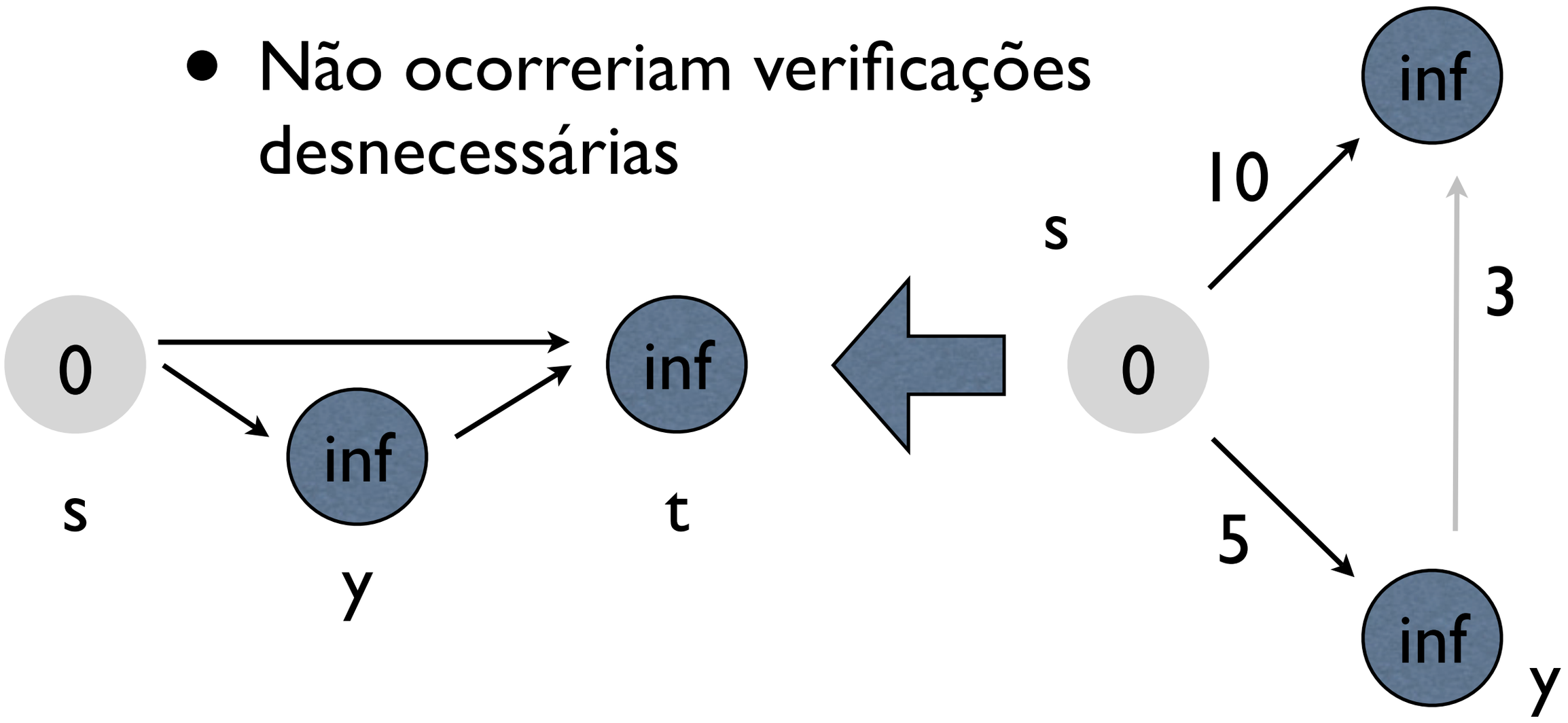
Bellman-Ford (DAG)

- Seria ótimo escolher a ordem de visita das arestas!
- Não ocorreriam verificações desnecessárias



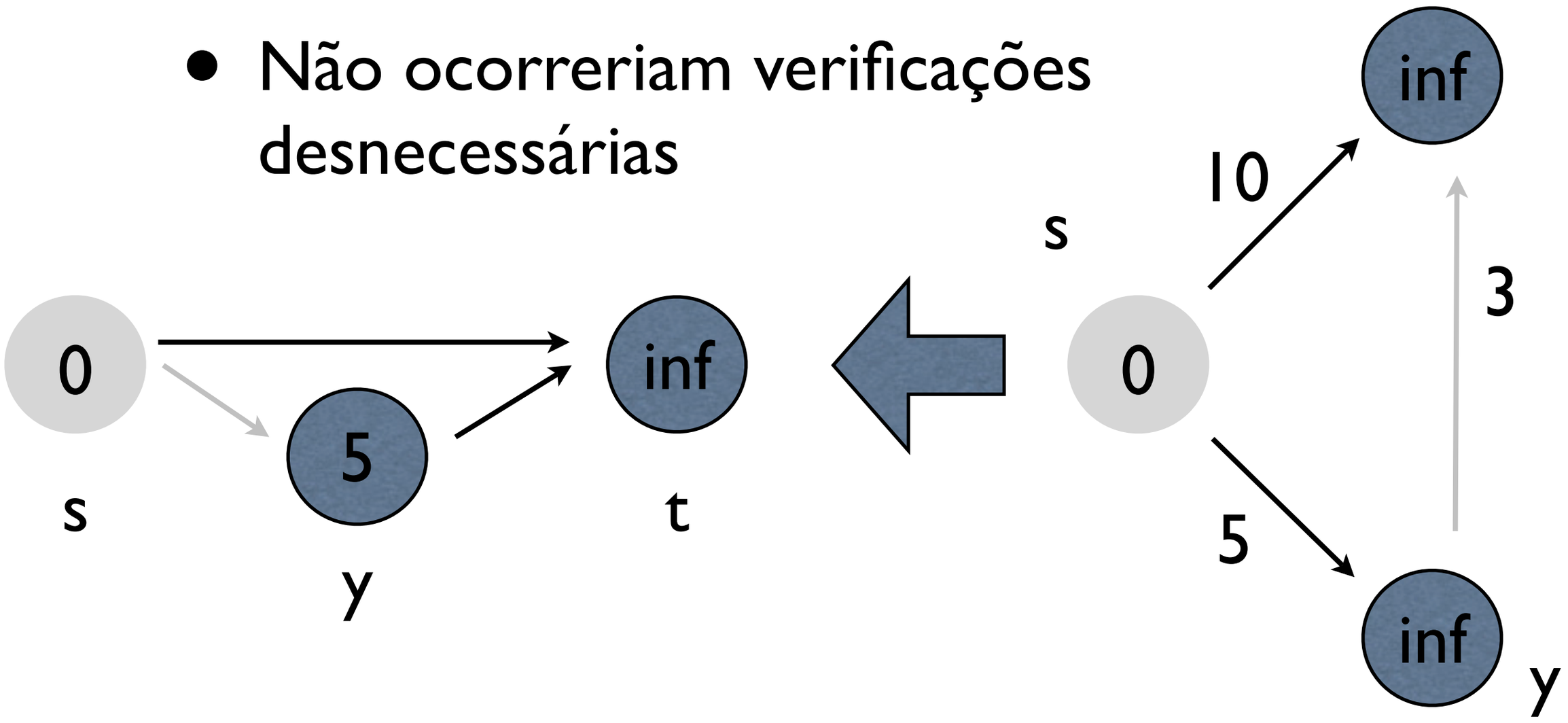
Bellman-Ford (DAG)

- Seria ótimo escolher a ordem de visita das arestas!
- Não ocorreriam verificações desnecessárias



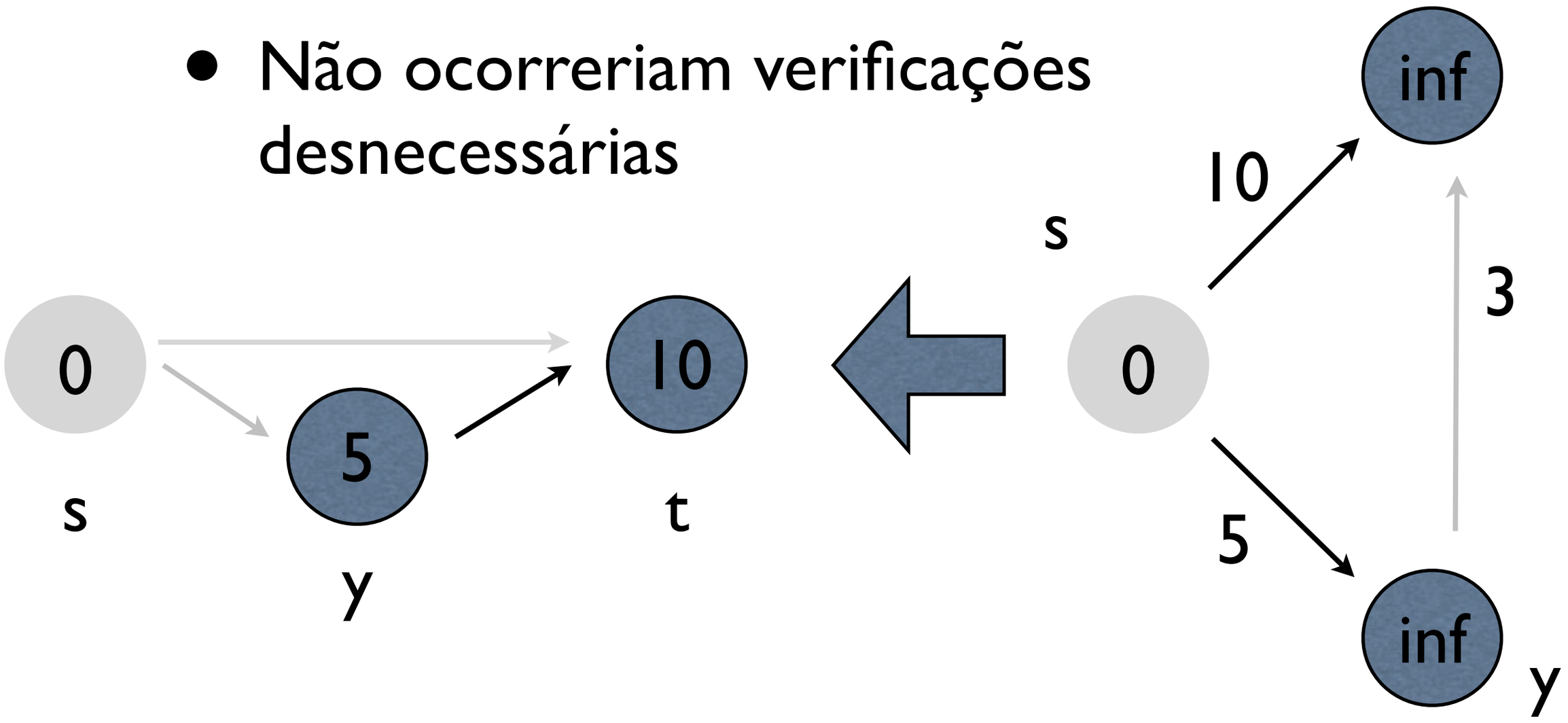
Bellman-Ford (DAG)

- Seria ótimo escolher a ordem de visita das arestas!
- Não ocorreriam verificações desnecessárias



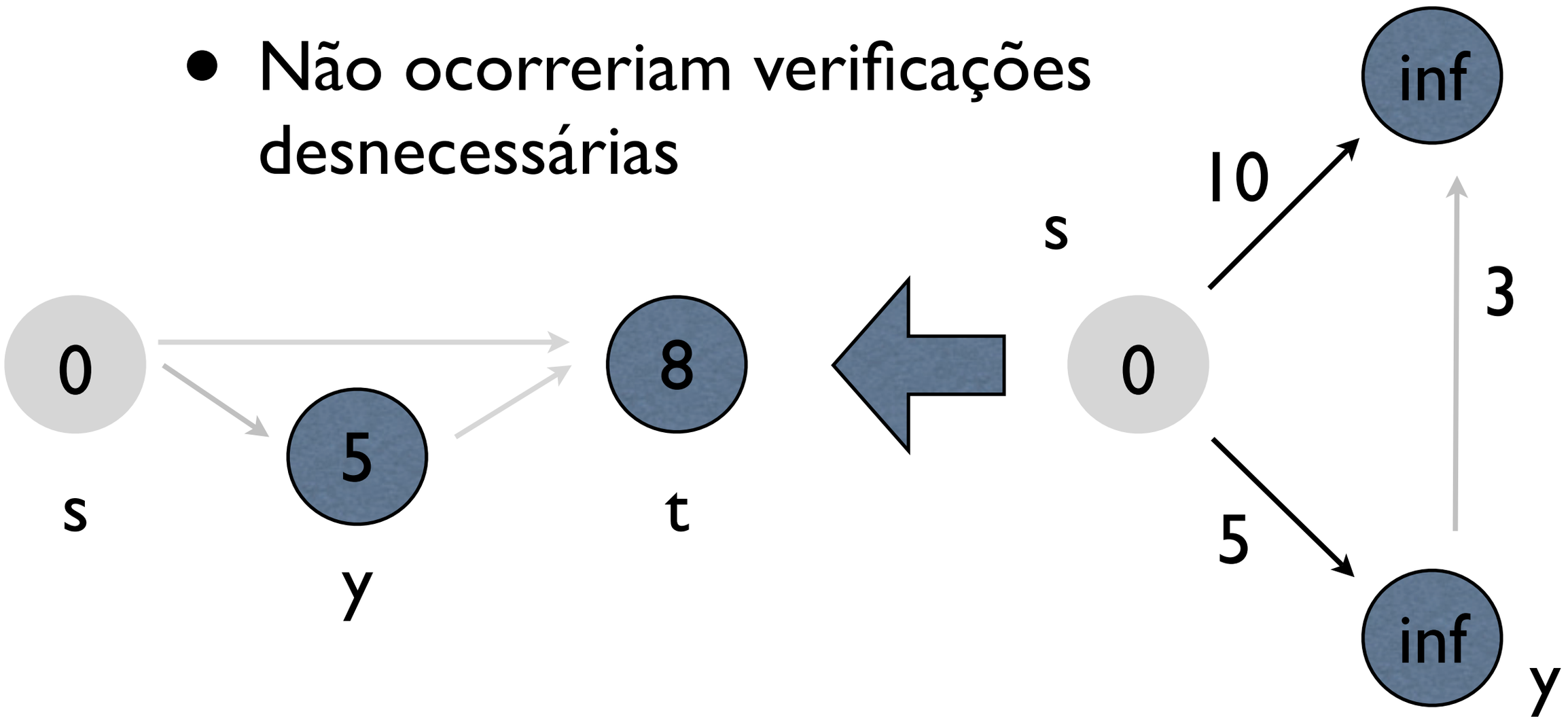
Bellman-Ford (DAG)

- Seria ótimo escolher a ordem de visita das arestas!
- Não ocorreriam verificações desnecessárias



Bellman-Ford (DAG)

- Seria ótimo escolher a ordem de visita das arestas!
- Não ocorreriam verificações desnecessárias



Bellman-Ford (DAG)

```
DAG_caminho_minimo(G,s)
    L = ordenacao_topologica(G)
    inicializa(G)
    Para cada v em L Faça
        Para cada u em vizinhos(v) Faça
            relaxa(u,v,w(u,v))
```

Visão Geral

- Caminho Mínimo de Única Fonte

Algoritmo	Grafo	Tempo
Busca em Largura	Somente arestas com pesos unitários	O(V+E)
Bellman-Ford-DAG	Sem ciclos, aceita pesos negativos	O(V+E)
Dijkstra	Somente arestas com pesos não-negativos	lista O(V^2), heap:O((V+E)log(V)), heap fib. O(Vlog(V) +E)
Bellman-Ford	Aceita arestas com pesos negativos	O(VE)