Grafos: caminhos mínimos
Parte 1

SCC0216 Modelagem Computacional em Grafos

Thiago A. S. Pardo
Maria Cristina F. Oliveira

O problema do menor caminho

- Um motorista deseja encontrar o caminho mais curto possível entre duas cidades do Brasil
- Ele consulta o Waze: a partir de um mapa rodoviário do Brasil, que apresenta as rodovias e distâncias entre cada par de cidades, como determinar uma rota mais curta entre as cidades desejadas?
- Uma maneira possível seria enumerar todas as rotas possíveis que levam de uma cidade à outra, calcular as distâncias, e então selecionar a mais curta

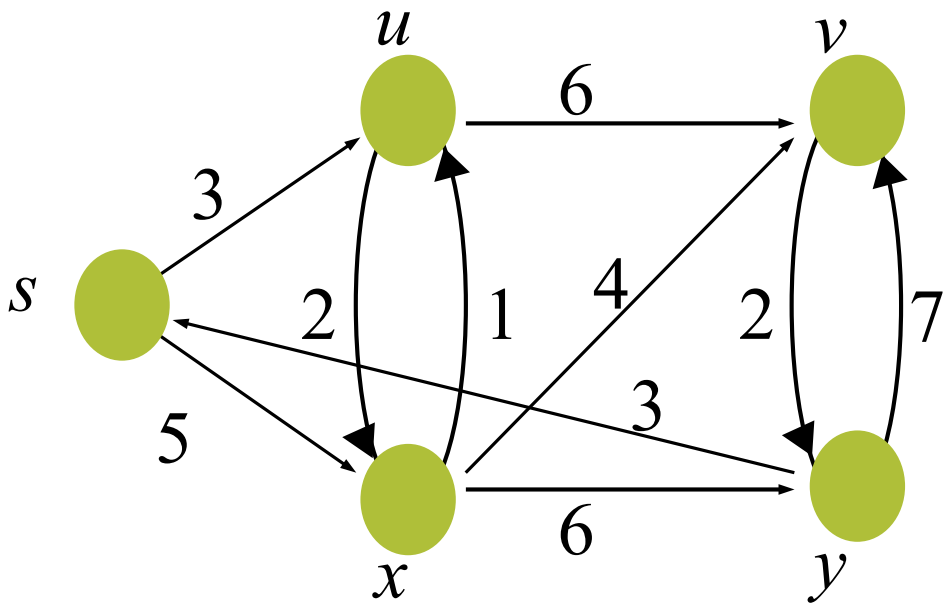
O problema do menor caminho

- Menor caminho = caminho de menor custo = melhor caminho
- Custo = distância ou qualquer outra métrica
- Cidades representadas como vértices de um grafo, rodovias entre elas representadas como relações de adjacência
  - Dois vértices estão conectados se as cidades que representam estão unidas por uma rodovia
  - Os pesos das arestas indicam o custo associado

Caminhos mínimos

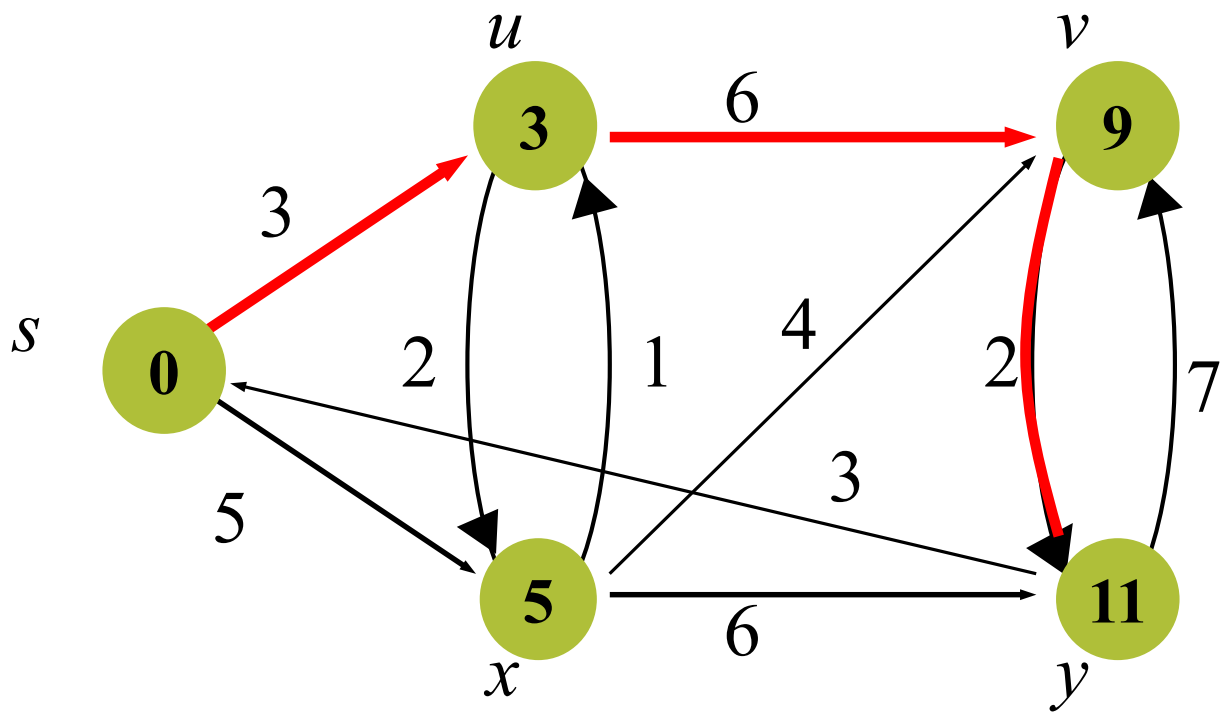
- O problema do caminho mínimo consiste em determinar um menor caminho entre um vértice de origem *u* e um vértice de destino *v*

- Qual o menor caminho entre *s* e *y*?



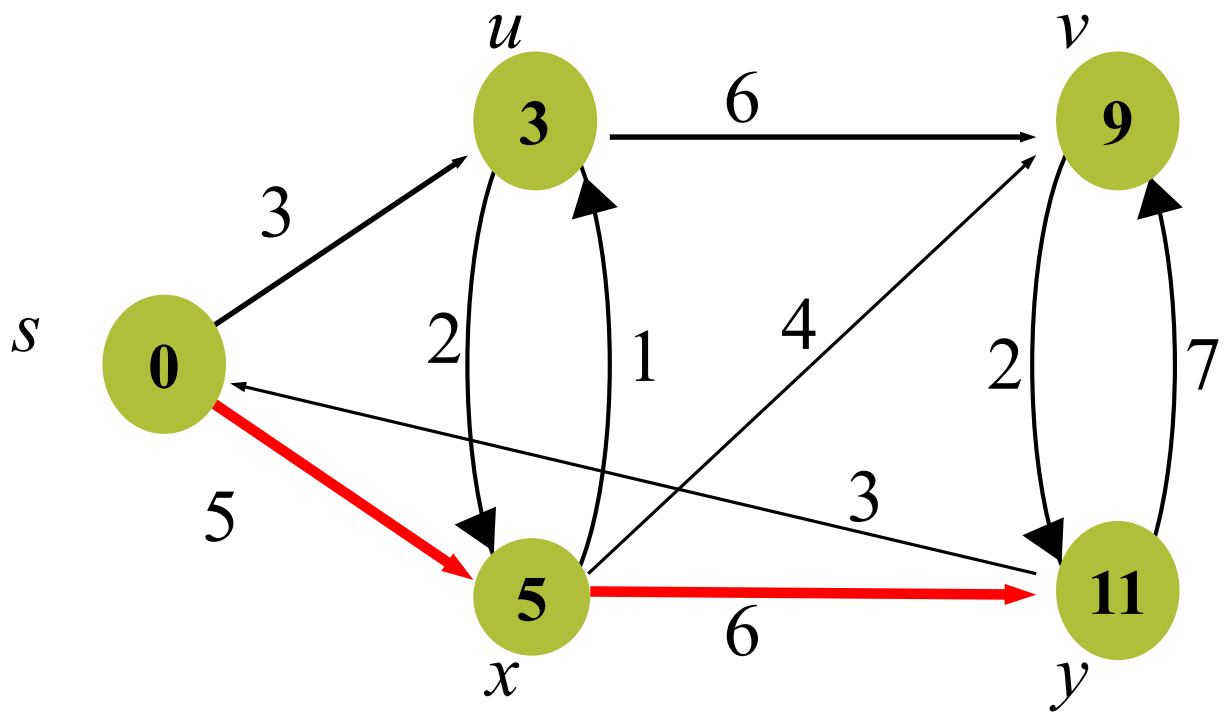
Caminhos mínimos

- Possíveis caminhos



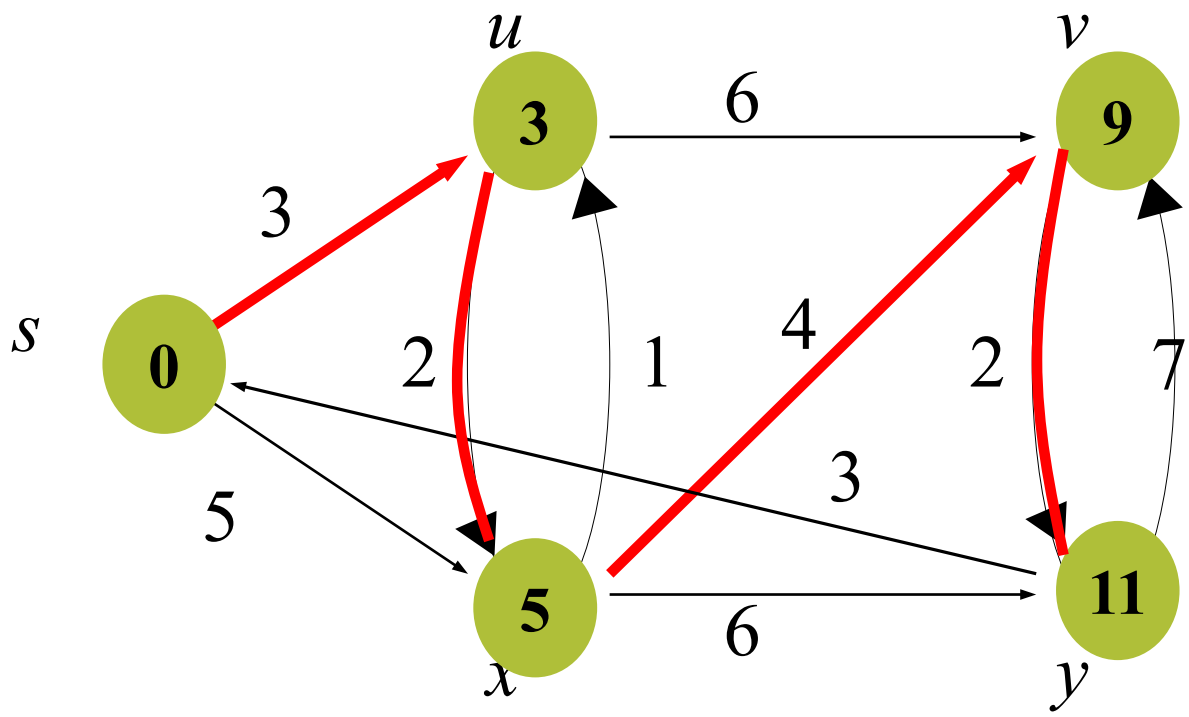
Caminhos mínimos

- Possíveis caminhos



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,A) com função peso *w*: A$\rightarrow\Re$ que associa pesos às arestas

- Peso (custo) do caminho p = <v_0, v_1, ..., v_k>

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

Caminho mínimo

- Grafo dirigido G(V,A) com função peso *w*: A$\rightarrow\Re$ que associa pesos às arestas
- 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

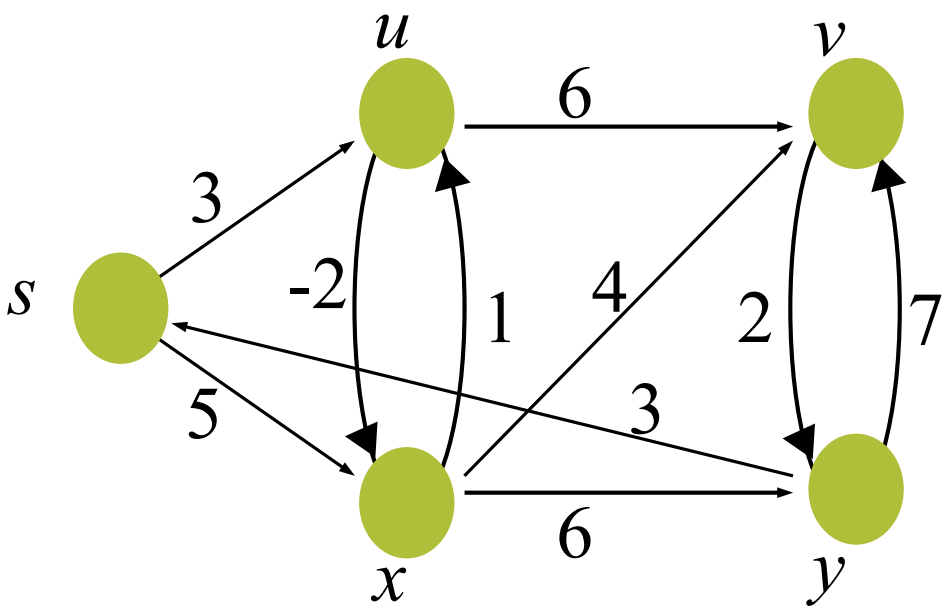
- Menor caminho entre os vértices *u* e *v* definido como qualquer rota **p** com um peso

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

- 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?

Exemplo

- Modelagem de transações financeiras, ou investimentos como um grafo
  - Vamos considerar um cenário em que você tem um portfólio de investimentos e deseja otimizar seus retornos considerando os custos associados à compra e venda de ativos

Exemplo

- Cada vértice representa um ativo diferente, como ações, títulos ou commodities. Por exemplo, os vértices podem ser rotulados como "Estoque A", "Título B" e "Commodity C“
- As arestas representam as transações ou negociações entre esses ativos
- Os pesos nas arestas representam os custos ou ganhos associados às transações
  - Pesos positivos podem representar custos de transação, enquanto pesos negativos podem representar ganhos ou lucros.
  - Por exemplo, se você vender uma ação por um preço mais alto do que pagou por ela, o peso da aresta correspondente seria negativo, indicando lucro
- Os caminhos no grafo representariam diferentes estratégias de investimento ou sequências de transações

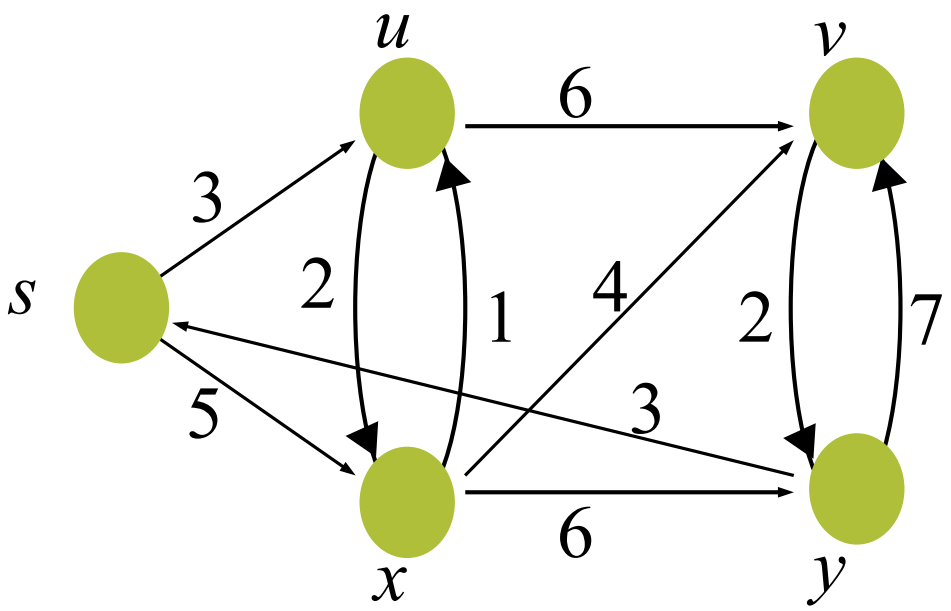
Exemplo

- O objetivo pode ser encontrar o caminho com o maior peso cumulativo, levando em consideração os pesos positivos e negativos: isso ajudaria a identificar a estratégia de investimento mais lucrativa considerando os custos envolvidos
- Ao permitir pesos negativos neste grafo, é possível modelar com precisão o aspecto financeiro dos investimentos, em que ganhos e perdas são fatores essenciais a serem considerados

- Algoritmos de menor caminho em grafos, como o algoritmo de Bellman-Ford ou Floyd-Warshall, permitem encontrar a sequencia de transações mais lucrativa ou otimizar uma estratégia de investimento considerando os ganhos e os custos de cada transação

Camino mínimo

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

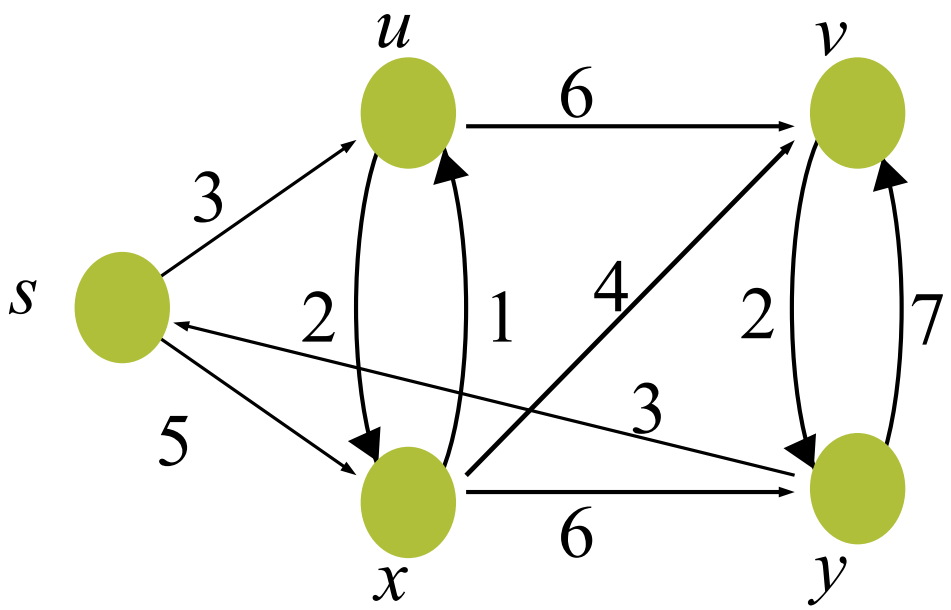


Caminhos mínimos

- Conceitos
  - Parte-se do vértice inicial *s*, associando-se a cada vértice um número d(*v*) que informa o valor (distância, custo) do menor caminho de *s* até *v*
  - d(*v*) mantém a estimativa do custo do menor caminho da origem até o vértice *v*

Caminhos mínimos

- Conceitos
  - Por exemplo, na figura abaixo, quando chegamos ao vértice *v*, d(*v*) será dado por min(d(*u*)+6, d(*x*)+4, d(*y*)+7)

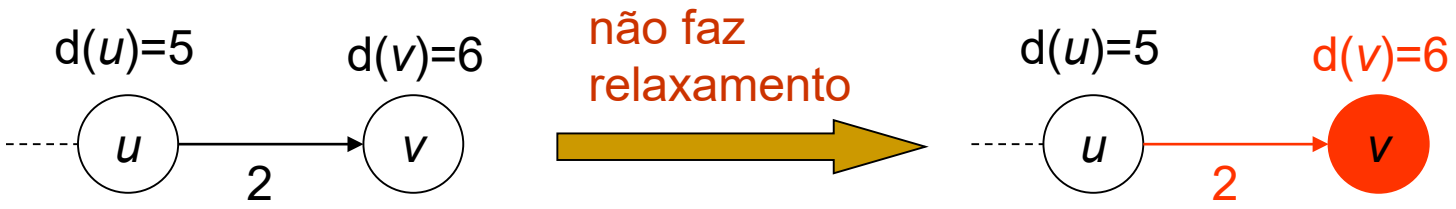


Caminhos mínimos

- Conceitos
  - *Relaxamento* de arestas
    - Faz-se uma estimativa pessimista para o valor do caminho mínimo até cada vértice: d(v) = ∞
    - O processo de 'relaxar' uma aresta consiste em verificar se é possível melhorar essa estimativa pessimista, fazendo um caminho que passa pelo vértice u e pela aresta (u,v)

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 essa estimativa, fazendo um caminho que passa pelo vértice u e pela aresta (u,v)



Caminhos mínimos

- Sub-rotina para relaxamento de arestas

relax(*u*, *v*, *w*) // (*u*,*v*) é a aresta, *w* é o seu peso

início

se d[*v*] > d[*u*] + *w*(*u*,*v*) **então**

d[*v*] = d[*u*] + *w*(*u*,*v*)

antecessor[*v*]=*u*

fim

Caminhos mínimos

- Sub-rotina para relaxamento de arestas

relax(*u*, *v*, *w*) // (*u*,*v*) é a aresta, *w* é o seu peso

início

se d[*v*] > d[*u*] + *w*(*u*,*v*) **então**

d[*v*] = d[*u*] + *w*(*u*,*v*)

antecessor[*v*]=*u*

fim

Caminhos mínimos

- Sub-rotina para relaxamento de arestas

relax(*u*, *v*, *w*) // (*u*,*v*) é a aresta, *w* é o seu peso

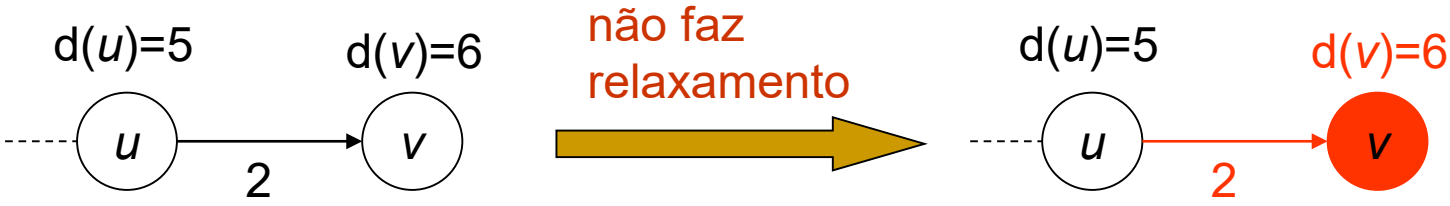
início

se d[*v*] > d[*u*] + *w*(*u*,*v*) **então**

d[*v*] = d[*u*] + *w*(*u*,*v*)

antecessor[*v*]=*u*

fim



Caminho mínimo

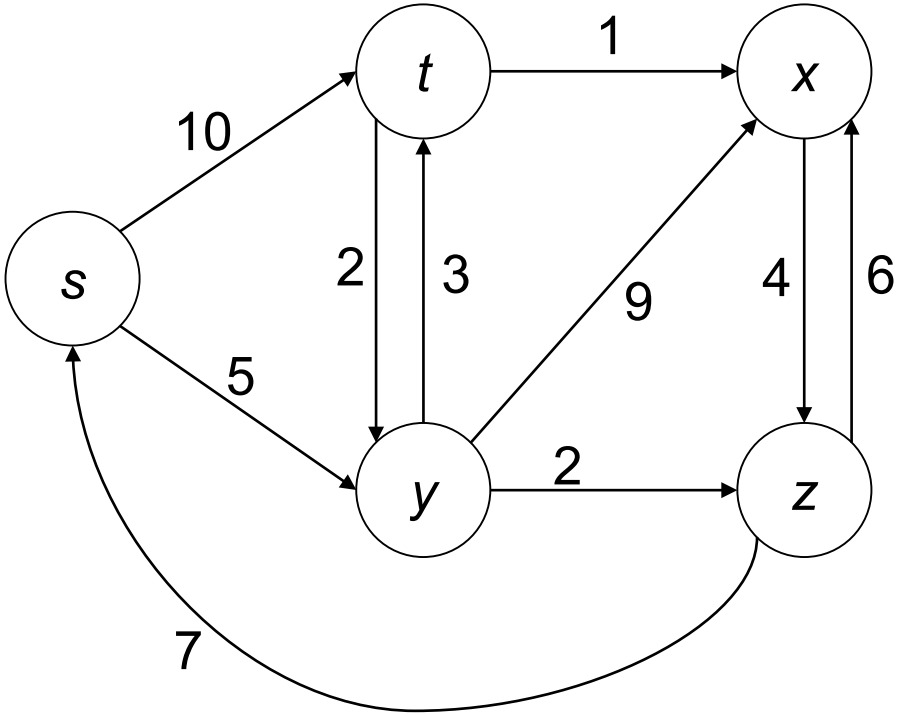
- Várias possibilidades de caminhos
  - Caminhos mais curtos de origem única
  - Caminhos mais curtos de destino único
  - Caminho mais curto de par único
  - Caminhos mais curtos de todos os pares

Algoritmo de Dijkstra

- Características
  - Caminho mais curto de origem única
  - Admite ciclos
  - Só admite pesos positivos
- Método
  - Inicia-se com estimativas pessimistas para cada vértice
  - A cada passo, adiciona a um conjunto S um vértice *u* de menor estimativa de caminho mínimo
    - Ao final, S vai conter os vértices do caminho mínimo
    - A cada passo, S contém os vértices cujo caminho mínimo desde a origem já foi definido
  - Relaxam-se as arestas adjacentes a *u*

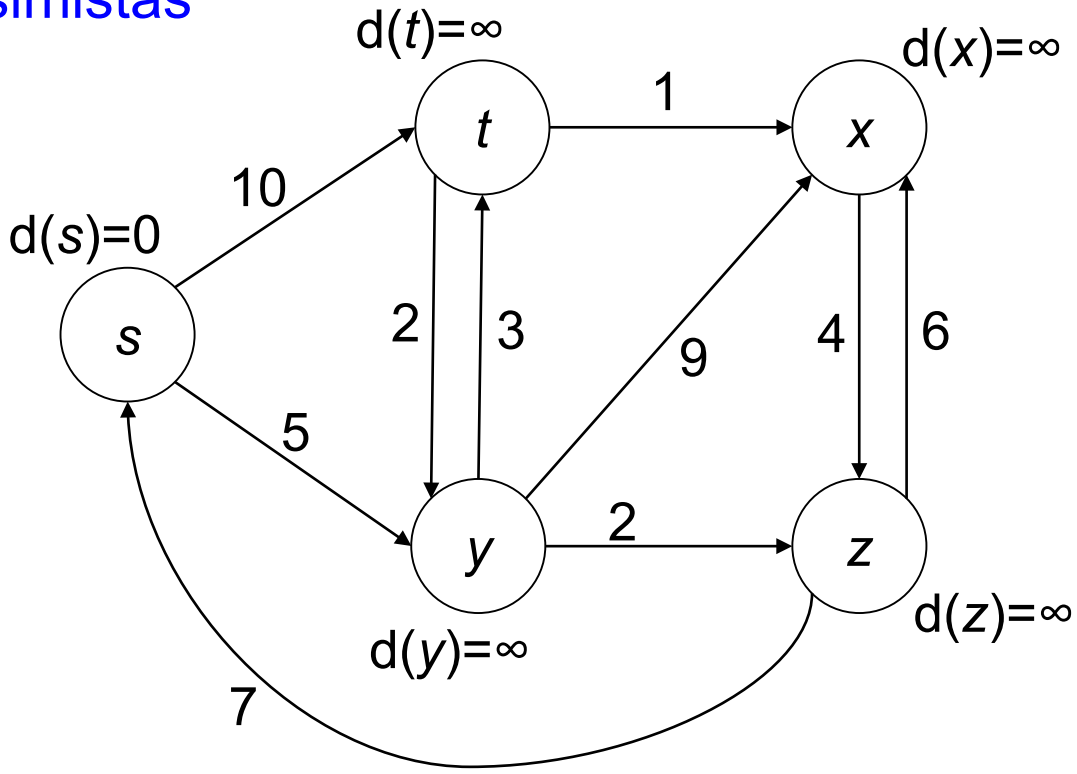
Algoritmo de Dijkstra

- Exemplo (a partir de *s*)



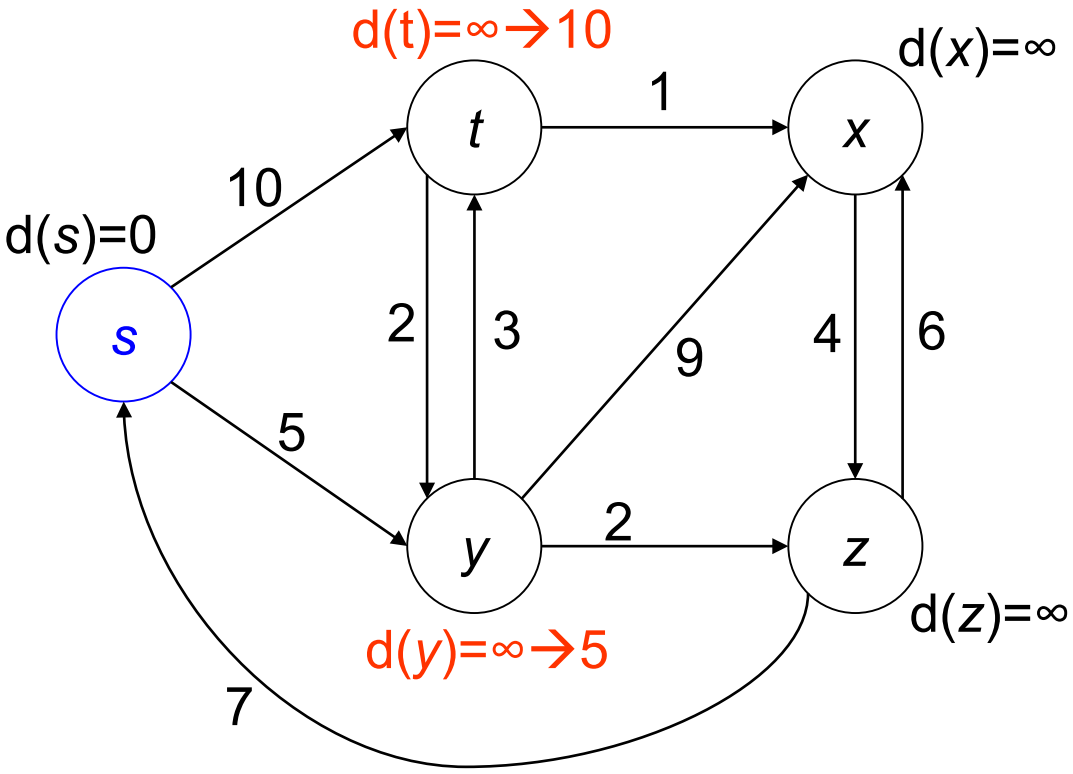
Algoritmo de Dijkstra

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



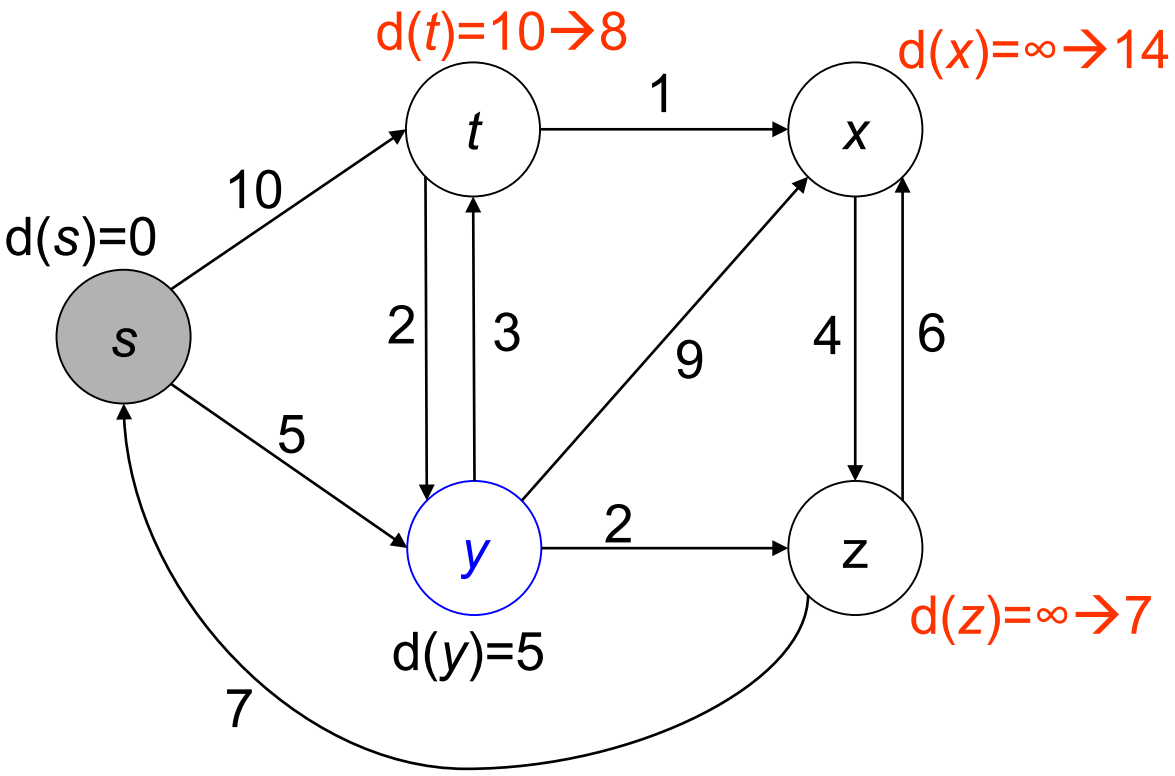
Algoritmo de Dijkstra

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



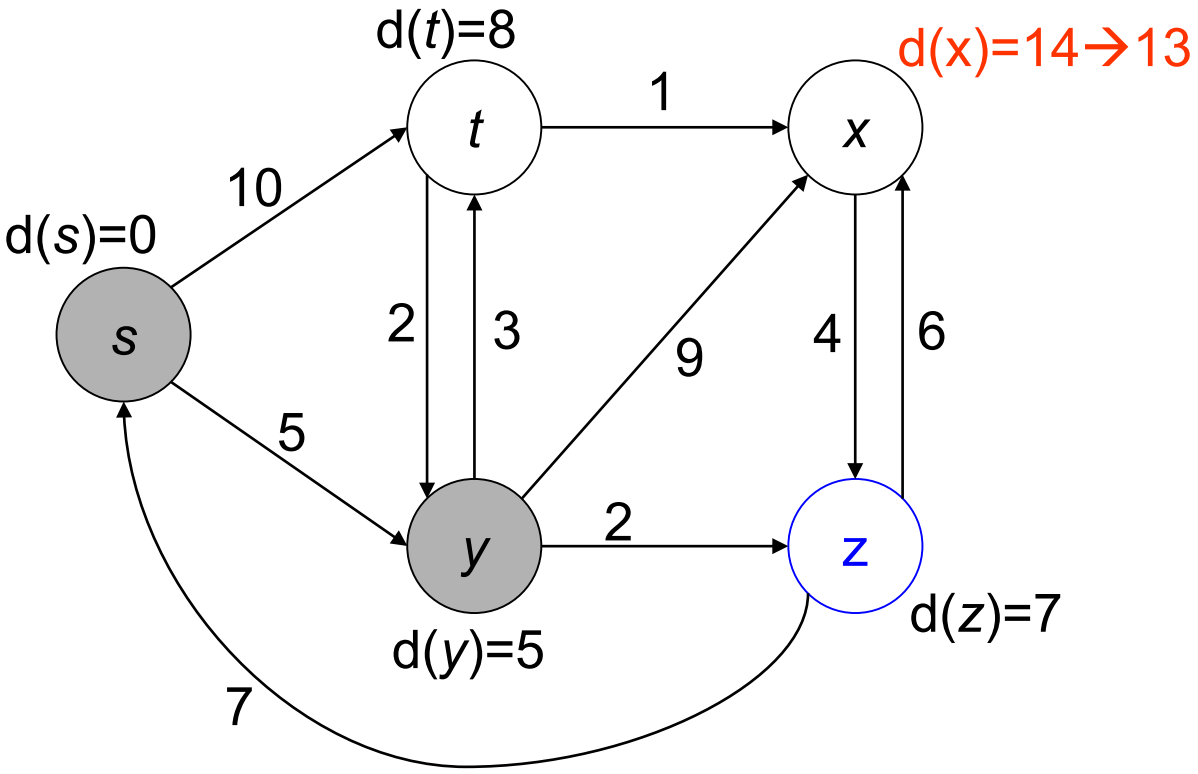
Algoritmo de Dijkstra

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



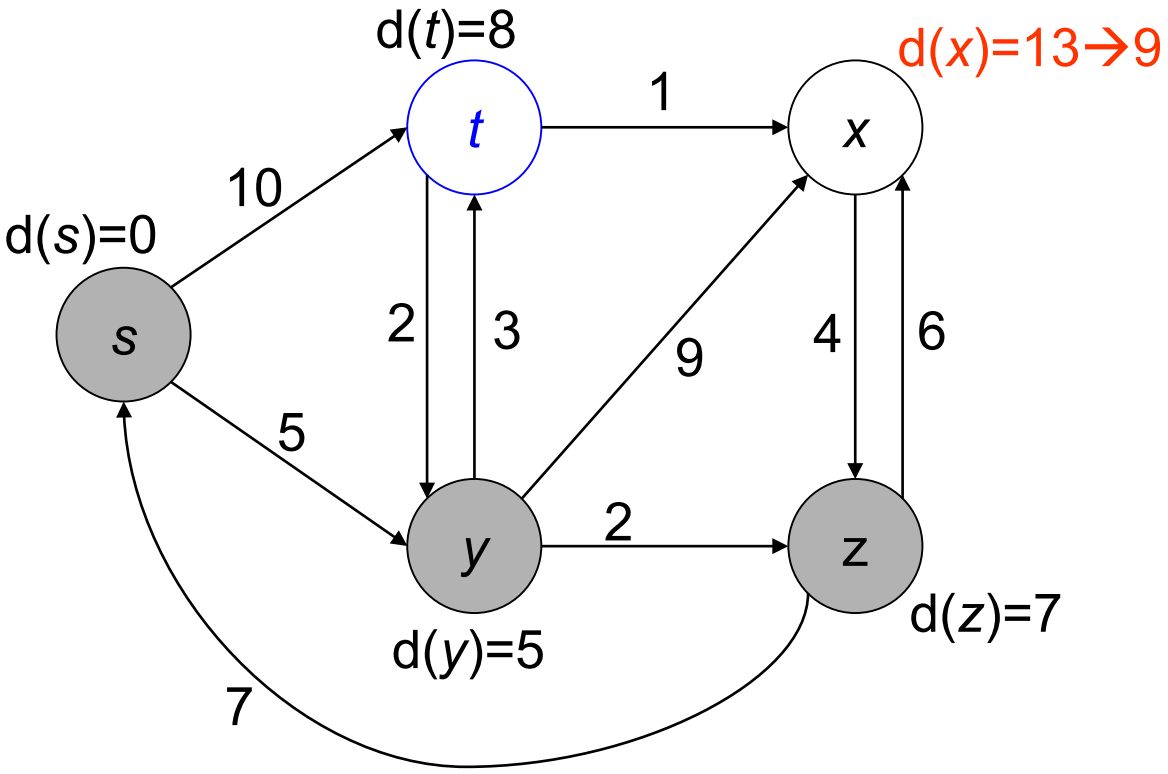
Algoritmo de Dijkstra

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



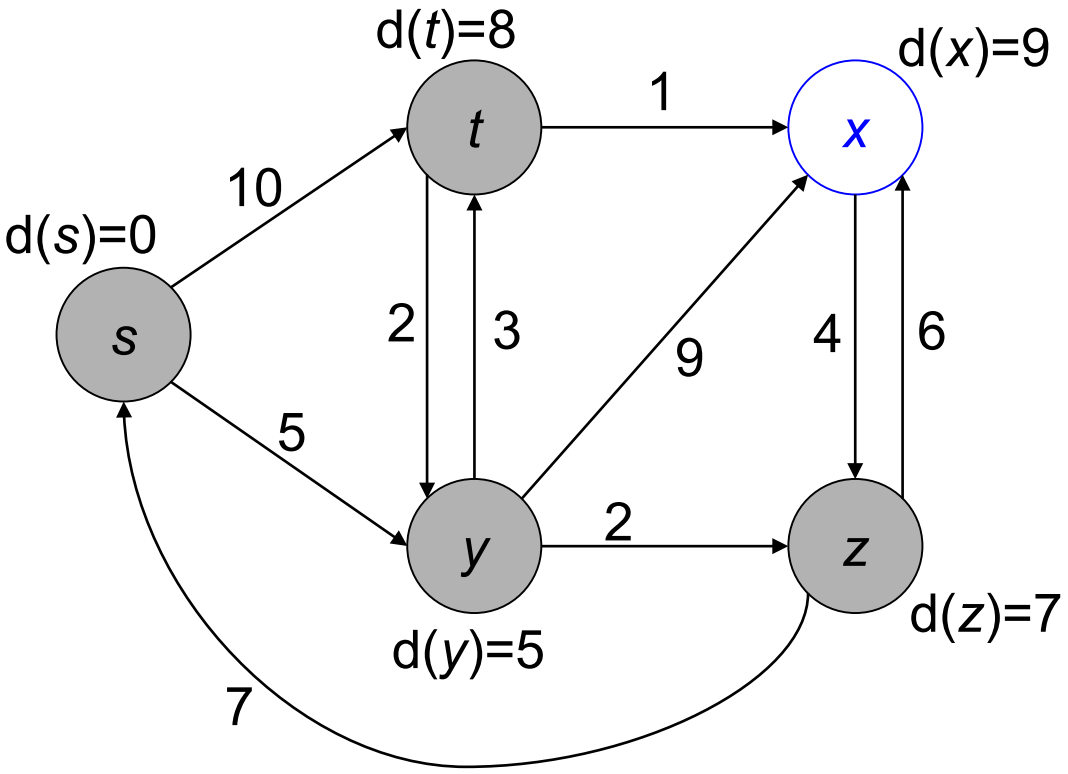
Algoritmo de Dijkstra

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



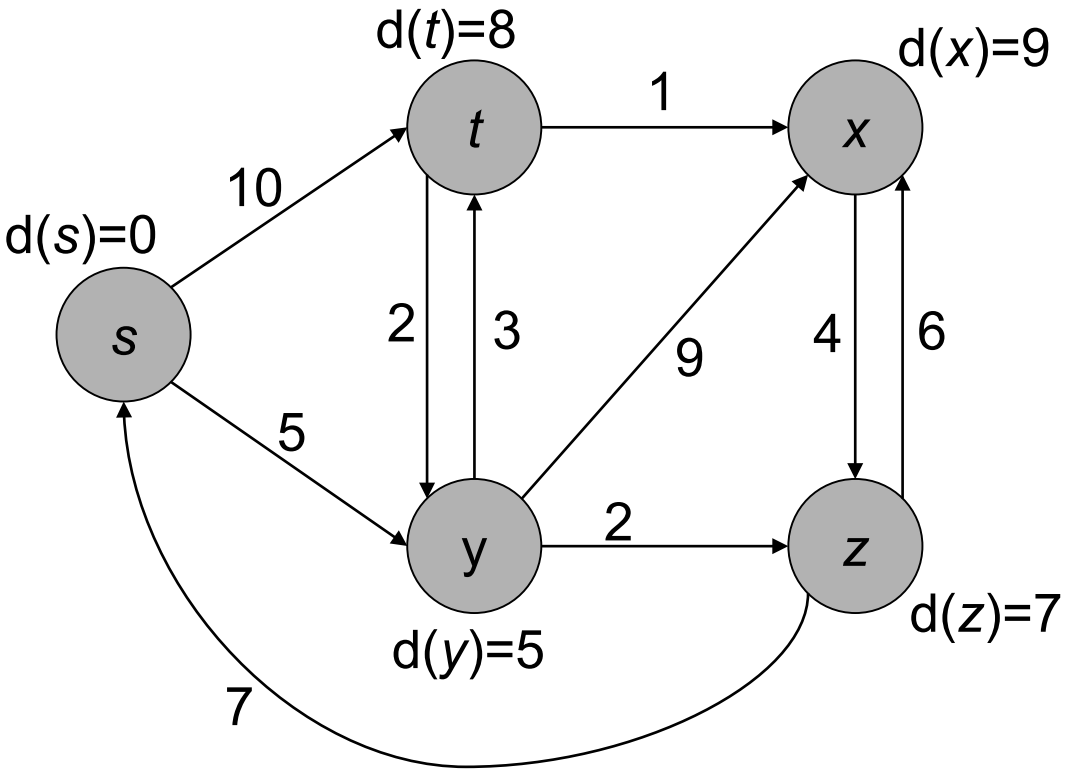
Algoritmo de Dijkstra

- 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

- Por que funciona? Solução ótima?

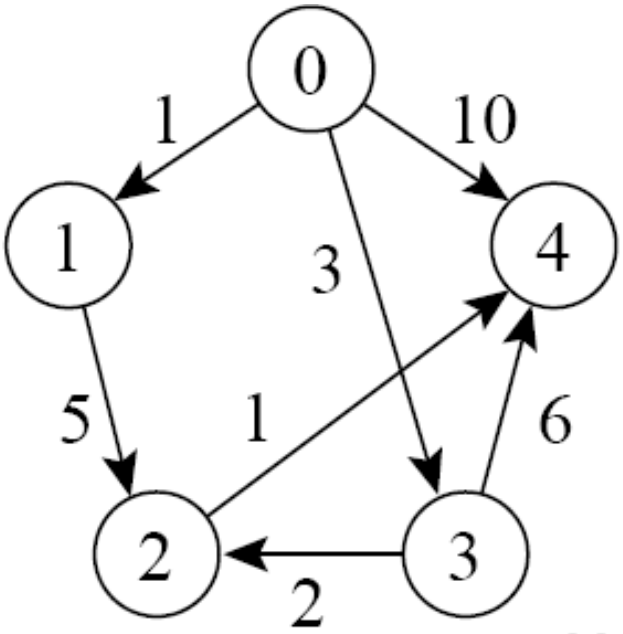
Algoritmo de Dijkstra

- Por que funciona? Solução ótima?
  - Estratégia gulosa
  - Solução ótima, pois sempre busca pelo vértice mais leve
    - Como o algoritmo examina os vértices na ordem de suas distâncias da origem, uma vez que um vértice é processado, sua distância final à origem está determinada e não será reconsiderada

Exercícios

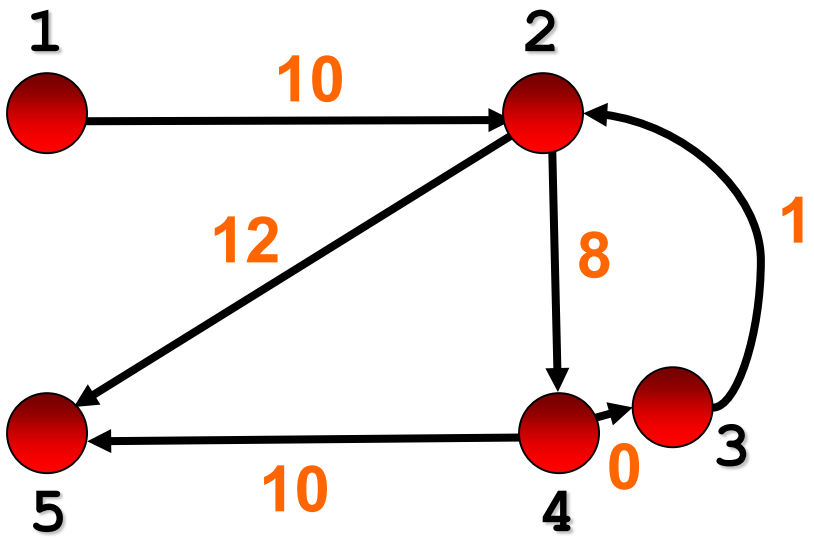
Exercício

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



Exercício

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



Algoritmo de Dijkstra

- Como implementar?

Algoritmo de Dijkstra

- Como implementar?
  - Como identificar qual é o vértice com menor estimativa?

Algoritmo de Dijkstra

- Implementação do algoritmo de Dijkstra
  - Insere os vértices em uma **fila de prioridades**, organizada em função da atual estimativa do custo do 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  // custo do caminho, até o momento
    S= Ø
    cria fila de prioridade F com os vértices do grafo
    //insere vértice u em S e faz relaxamento a partir das arestas
    adjacentes a u
    enquanto F ≠ Ø faça
            u= retira vértice de F
            S= S + {u}
            para cada vértice v adjacente a u faça
                    relax(u,v,w) //atualizando fila de prioridade, se necessário
    fim
```

Algoritmo de Dijkstra

- Complexidade de tempo
  - Depende da estrutura de dados utilizada para manter a fila de prioridade!
  - Implementações costumam usar um *heap* binária
  - (ou uma Fibonacci *heap*)
  - Heap Visualization (usfca.edu)

Algoritmo de Dijkstra

```
DIJKSTRA(G, w, s)        // |V| vértices e |A| arestas
    início
    //inicializa variáveis
    para cada vértice v faça  // custo deste laço é O(|V|)
            d[v]= ∞
            antecessor[v]= -1
    d[s]= 0  // s vértice inicial
    S= Ø
    cria fila de prioridade F com os |V| vértices de G  // custo é O(|V|log|V|)
    //insere vértice u em S e faz relaxamento a partir das arestas
    adjacentes a u
    enquanto F ≠ Ø faça     // custo deste laço é O(|V|)
            u= retira vértice de F  // custo desta operação é O(log(|V|)
            S= S + {u}
            para cada vértice v adjacente a u faça // custo total é O(|A|)
                    relax(u,v,w) //atualiza fila de prioridade, se necessário,
                                                custo de cada operação é O(log(|V|)
    fim
```

Algoritmo de Dijkstra

- Complexidade de tempo (se fila de prioridade implementada em uma *heap* binária):
  - O(|V|) para inicializar variáveis
  - O(|V|*log|V|) para criar a fila de prioridade com todos os vértices na *heap binária*
  - O(|V|) para processar cada vértice. Isso requer atualizar a fila de prioridades após cada remoção, que é O(log|V|)
    - O(|V|*log|V|), portanto
  - O(|A|) para processar todas as arestas. Mas, para cada relaxamento, precisa atualizar a fila de prioridades, que é O(log|V|)
    - O(|A|*log|V|), portanto
  - Total: O(|V|) + O(|V|*log|V|) + O(|V|*log|V|) + O(|A|*log|V|)

Algoritmo de Dijkstra

- Complexidade de tempo (se fila de prioridade implementada em uma *heap* binária):

- $O(|V|)+O(|V|*\log|V|)+O(|V|*\log|V|)+O(|A|*\log|V|)$

  ou seja

- $O((|V| + |A|)*\log|V|)$

  ou seja

- $O(|A|*\log|V|)$, no pior caso, se $|A|>>|V|$

Algoritmo de Dijkstra

- Complexidade de tempo (se fila de prioridade implementada em uma *heap* binária):
  - O(|V|) para inicializar variáveis
  - O(|V|) para criar a fila de prioridade com otodos os vértices na *heap binária*
  - O(|V|) para processar cada vértice. Mas isso inclui atualizar a sua posição na fila de prioridades, que é O(log|V|)
    - O(|V| log|V|), portanto
  - O(|A|) para processar todas as arestas. Mas, para cada relaxamento, precisa atualizar a fila! Custo de cada operação é O(log|V|)
    - O(|A| log|V|), portanto
- O(|V|) + O(|V|) + O(|V|log|V|) + O(|A|log|V|)

Algoritmo de Dijkstra

- Complexidade de tempo
  - **Com *heap* binário**: O((|V| + |A|)*log (|V|)).
  - O algoritmo itera |V| vezes (uma vez para cada vértice) e executa operações de remoção no heap binário, que levam tempo O(log(|V|)). Além disso, cada aresta é relaxada no máximo uma vez, resultando em uma complexidade total de O(|A|*log|V|)

- O(|A| log|V|), no pior caso, considerando |A|>>|V|

Algoritmo de Dijkstra

- Complexidade de tempo
  - Se um heap de Fibonacci for usado como fila de prioridade, a complexidade de tempo pode ser reduzida para O(|V|log(|V|) + |A|).
  - As operações de remover e de atualizar na *heap* de Fibonacci têm uma complexidade de tempo amortizada de O(log(|V|)), resultando em uma complexidade geral aprimorada.

Algoritmo de Dijkstra

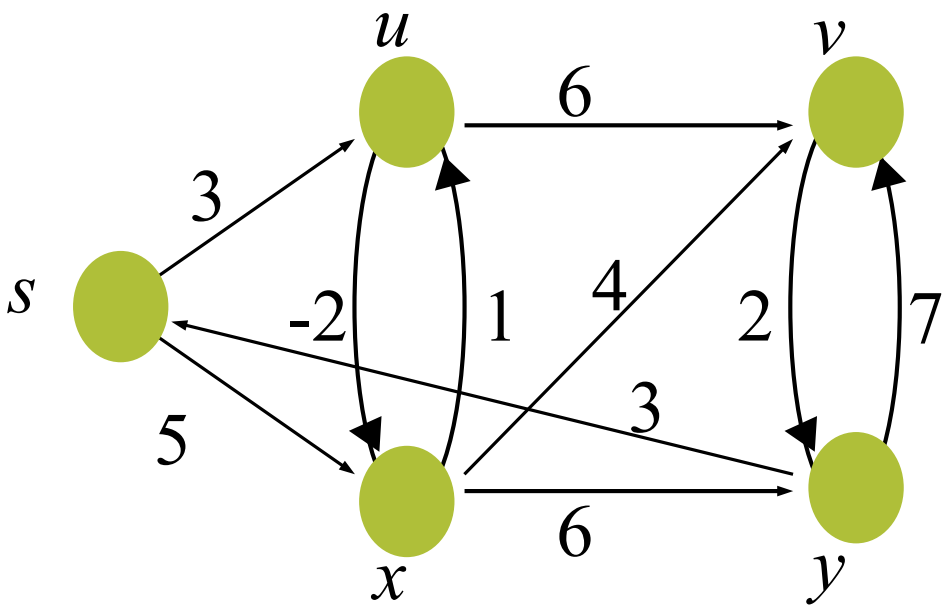
- Complexidade de tempo
  - Na prática, a escolha da estrutura de dados depende das características do grafo e dos requisitos específicos da aplicação
  - Os *heaps* binários são mais simples de implementar e podem ter fatores constantes mais baixos, enquanto os *heaps* de Fibonacci podem oferecer melhor complexidade de tempo, mas com maior sobrecarga na prática
  - Se o grafo for esparso ($|A| << |V|^2$), o número real de arestas consideradas pelo algoritmo pode ser significativamente menor, resultando potencialmente em melhor desempenho

Questão

- Como lidar com os demais casos abaixo?
  - Caminhos mais curtos de origem única → **ok**
  - Caminhos mais curtos de destino único?
  - Caminho mais curto de par único?
  - Caminhos mais curtos de todos os pares?

Exercício

- Propor algoritmo de caminho mínimo para grafo com ciclo e peso negativo



Questão

- Para pensar
  - Como a busca pelo caminho mínimo pode se beneficiar da ordenação topológica?