Caminho Mínimo de Fonte Única em Grafos sem Pesos Negativos

Letícia Rodrigues Bueno

UFABC

Problema 2: Menor caminho entre duas cidades

- Dado um mapa de cidades, contendo as distâncias entre cidades, qual o menor caminho entre quaisquer cidades *A* e *B*?

Problema 2: Menor caminho entre duas cidades

- Dado um mapa de cidades, contendo as distâncias entre cidades, qual o menor caminho entre quaisquer cidades *A* e *B*?
- Este problema pode ser modelado através de um grafo:
  - **Cidade:** vértices;
  - **Estradas entre cidades:** arestas ponderadas com peso que indicam distância entre cidades.
  - **Arestas ponderadas:** podem indicar tempo, custo, penalidades, perdas, etc;
- Esse problema é conhecido como **Problema do Caminho Mínimo.**

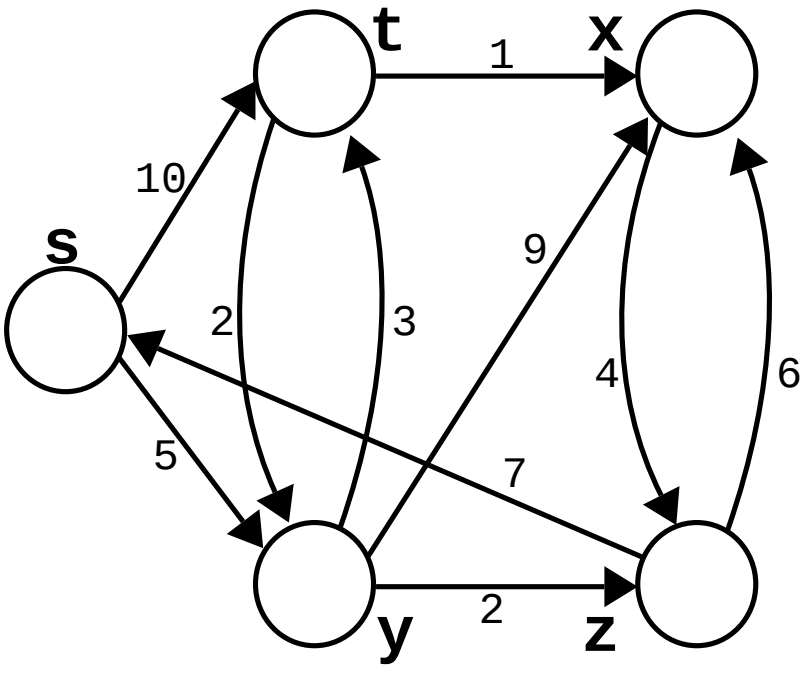
Abordagem

- Problema do caminho mínimo é semelhante a problema do número de Erdõs;

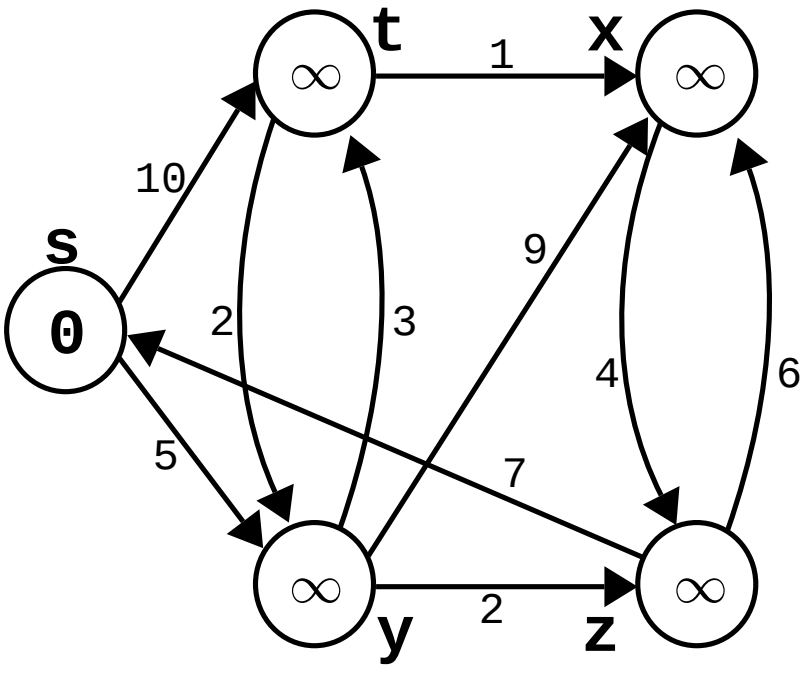
Abordagem

- Problema do caminho mínimo é semelhante a problema do número de Erdõs;
- Podemos utilizar uma idéia semelhante a do algoritmo de **Busca em Largura**;
- Levar em consideração:
  - O problema tem subestrutura ótima: um caminho mínimo entre dois vértices contém outros caminhos mínimos em seu interior.

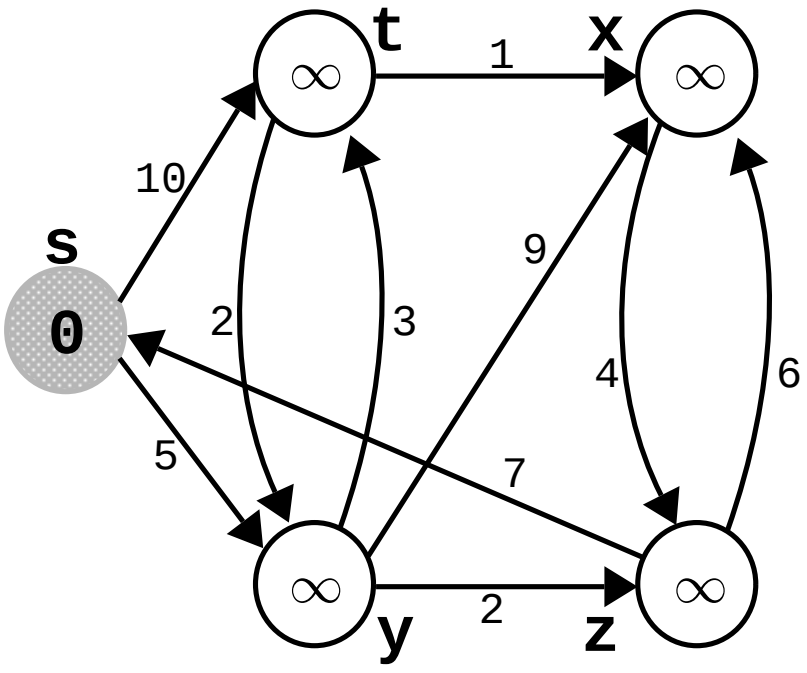
Caminho mínimo de fonte única: algoritmo de Dijkstra



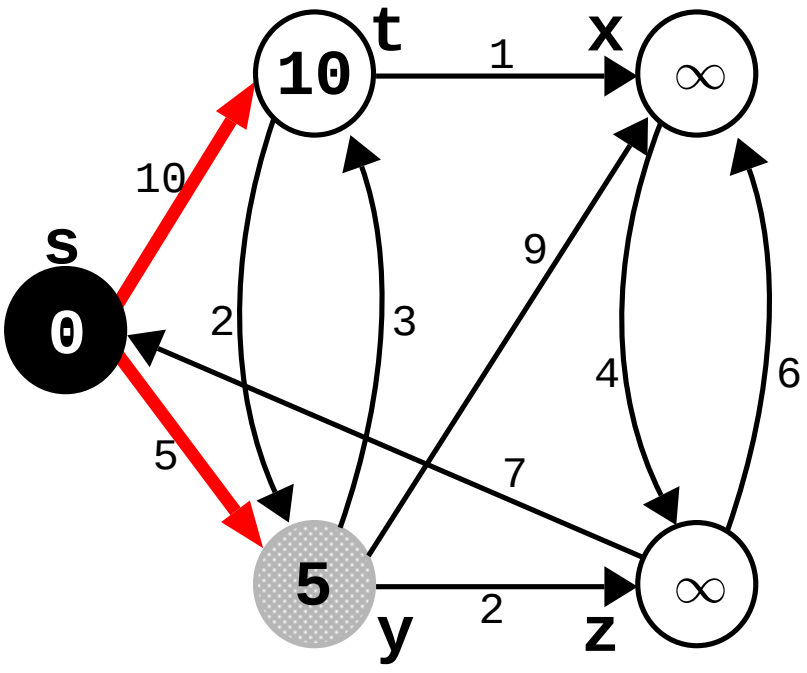
Caminho mínimo de fonte única: algoritmo de Dijkstra



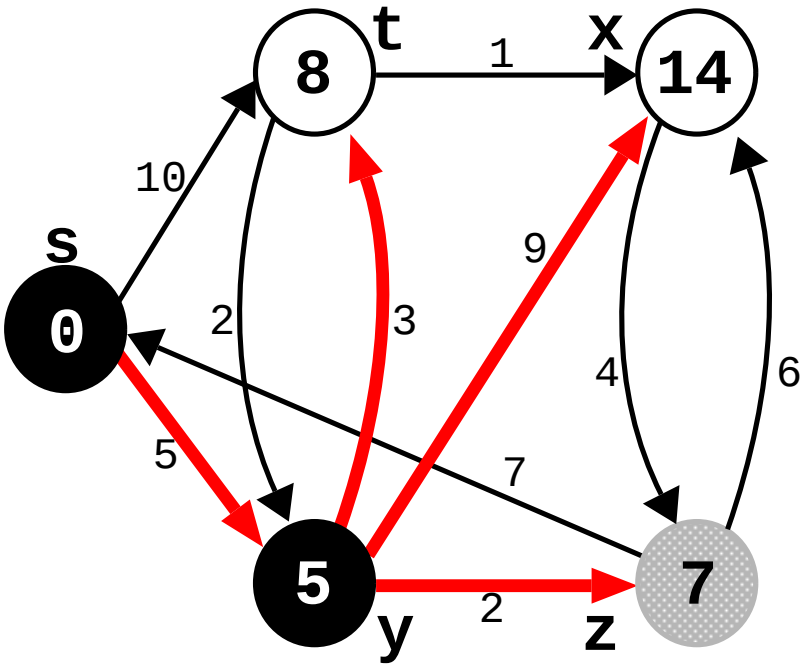
Caminho mínimo de fonte única: algoritmo de Dijkstra



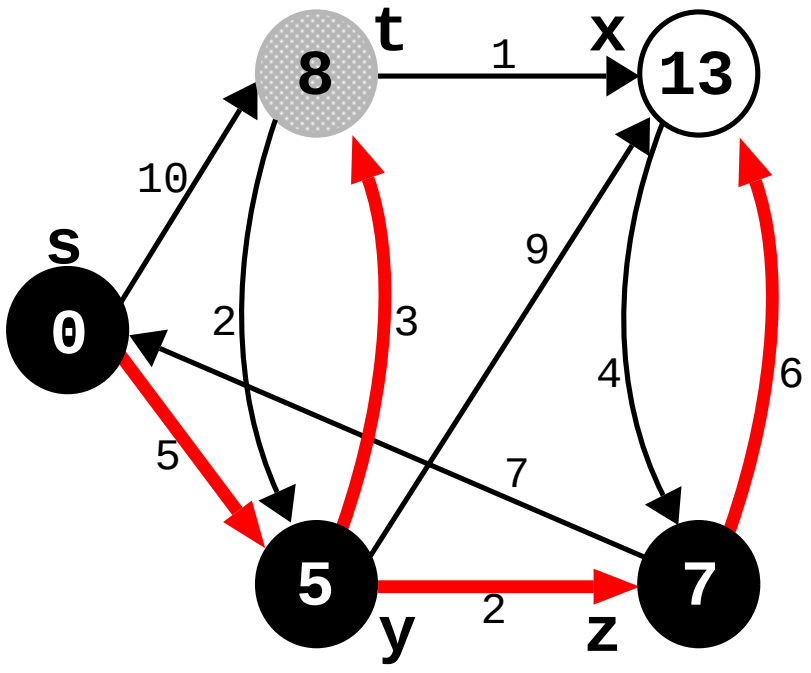
Caminho mínimo de fonte única: algoritmo de Dijkstra



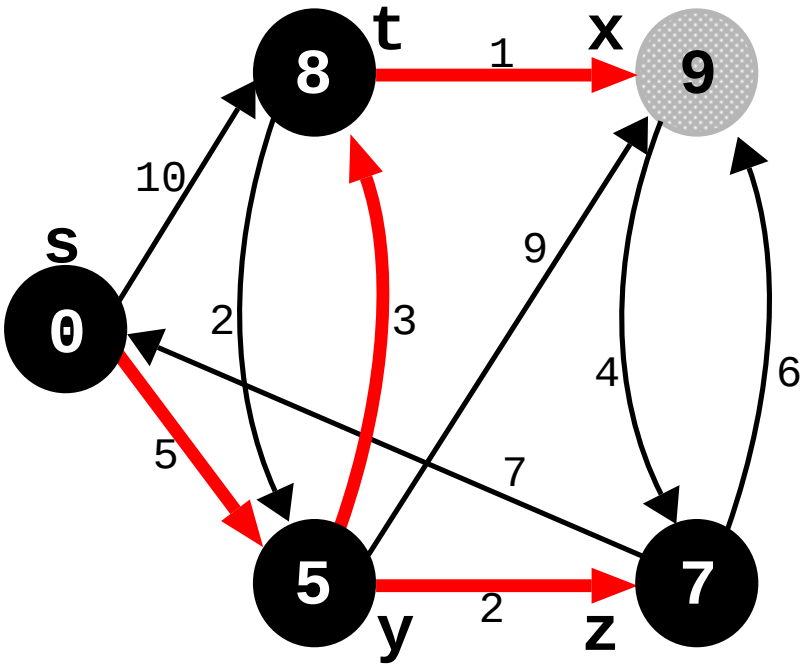
Caminho mínimo de fonte única: algoritmo de Dijkstra



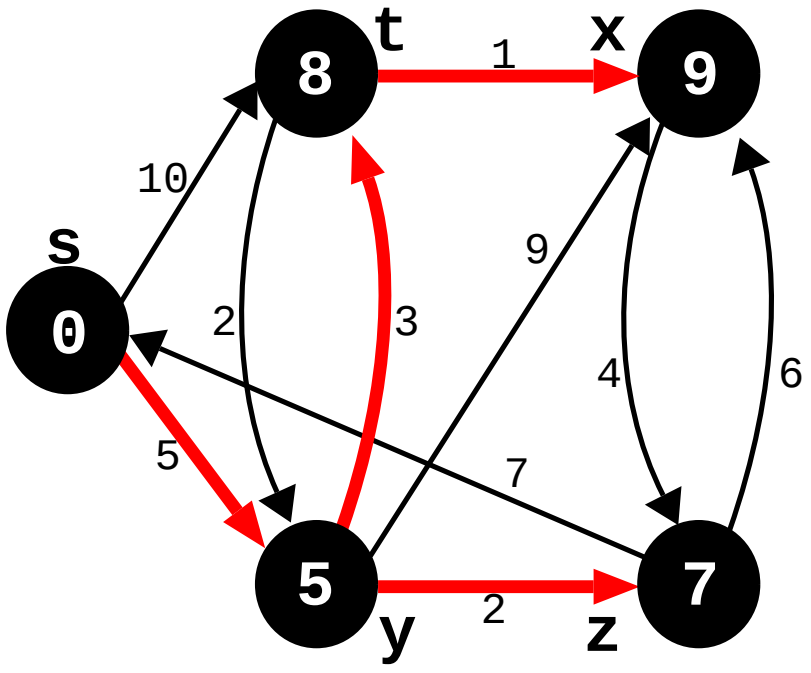
Caminho mínimo de fonte única: algoritmo de Dijkstra



Caminho mínimo de fonte única: algoritmo de Dijkstra

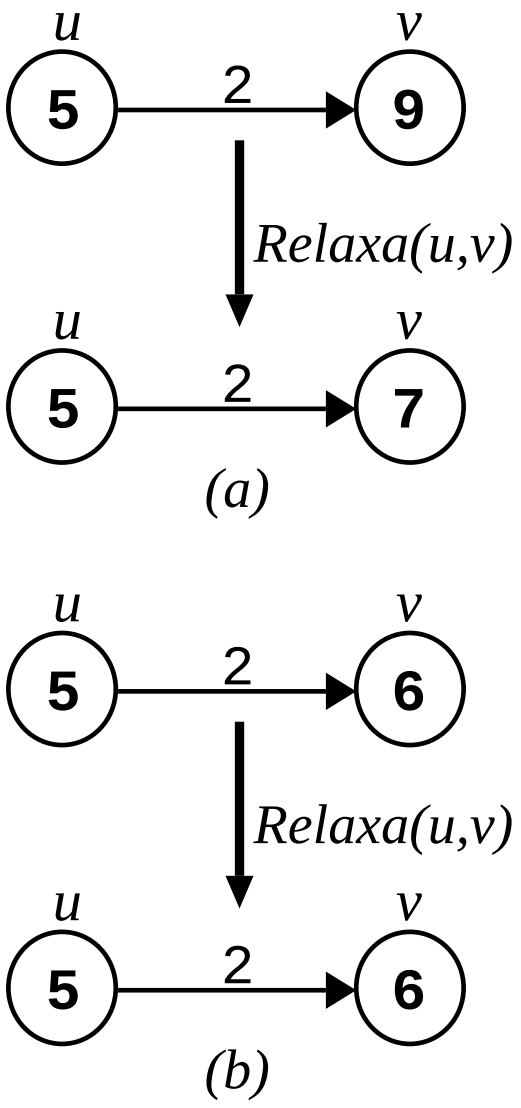


Caminho mínimo de fonte única: algoritmo de Dijkstra



Caminho mínimo de fonte única: algoritmo de Dijkstra

```
1   relaxa(u, v):
2       se v.d > u.d + p(u, v) então
3           v.d = u.d + p(u, v)
4           v.p = u
```



Caminho mínimo de fonte única: algoritmo de Dijkstra

```
 1  dijkstra(G, s):
 2      para u em V(G) faça
 3          u.d = ∞
 4          u.p = None
 5      s.d = 0
 6      s.p = s
 7      A = Heap(V(G)) \\ com base em d
 8      S = [ ]
 9      enquanto tamanho(A) > 1 faça
10          u = retira_min(A)
11          S = S + u
12          para v em adj(u) faça
13              relaxa(u, v)
14              refaz_heap(A)
```

Caminho mínimo de fonte única: algoritmo de Dijkstra

Análise da complexidade:

```
1   dijkstra(G, s):
2       para u em V(G) faça
3           u.d = ∞
4           u.p = None
5       s.d = 0
6       s.p = s
7       A = Heap(V(G)) \\ com base em d
8       S = [ ]
9       enquanto tamanho(A) > 1 faça
10          u = retira_min(A)
11          S = S + u
12          para v em adj(u) faça
13              relaxa(u, v)
14              refaz_heap(A)
```

Caminho mínimo de fonte única: algoritmo de Dijkstra

```
 1  dijkstra(G, s):
 2      para u em V(G) faça
 3          u.d = ∞
 4          u.p = None
 5      s.d = 0
 6      s.p = s
 7      A = Heap(V(G)) \\ com base em d
 8      S = [ ]
 9      enquanto tamanho(A) > 1 faça
10          u = retira_min(A)
11          S = S + u
12          para v em adj(u) faça
13              relaxa(u, v)
14              refaz_heap(A)
```

Análise da complexidade:

- Construção do *heap* (linha 6): $O(n)$

Caminho mínimo de fonte única: algoritmo de Dijkstra

```
1   dijkstra(G, s):
2     para u em V(G) faça
3         u.d = ∞
4         u.p = None
5     s.d = 0
6     s.p = s
7     A = Heap(V(G)) \\ com base em d
8     S = [ ]
9     enquanto tamanho(A) > 1 faça
10        u = retira_min(A)
11        S = S + u
12        para v em adj(u) faça
13            relaxa(u, v)
14            refaz_heap(A)
```

Análise da complexidade:

- Construção do *heap* (linha 6): $O(n)$
- retira_min (linha 9) tem complexidade $(\log n)$ e é executado n vezes: $O(n \log n)$

Caminho mínimo de fonte única: algoritmo de Dijkstra

```
 1  dijkstra(G, s):
 2      para u em V(G) faça
 3          u.d = ∞
 4          u.p = None
 5      s.d = 0
 6      s.p = s
 7      A = Heap(V(G)) \\ com base em d
 8      S = [ ]
 9      enquanto tamanho(A) > 1 faça
10          u = retira_min(A)
11          S = S + u
12          para v em adj(u) faça
13              relaxa(u, v)
14              refaz_heap(A)
```

Análise da complexidade:

- Construção do *heap* (linha 6): $O(n)$
- retira_min (linha 9) tem complexidade $(\log n)$ e é executado n vezes: $O(n \log n)$
- refaz_heap (linha 13) tem complexidade $(\log n)$ e é executado m vezes: $O(m \log n)$

Caminho mínimo de fonte única: algoritmo de Dijkstra

```
 1  dijkstra(G, s):
 2      para u em V(G) faça
 3          u.d = ∞
 4          u.p = None
 5      s.d = 0
 6      s.p = s
 7      A = Heap(V(G)) \\ com base em d
 8      S = [ ]
 9      enquanto tamanho(A) > 1 faça
10          u = retira_min(A)
11          S = S + u
12          para v em adj(u) faça
13              relaxa(u, v)
14              refaz_heap(A)
```

Análise da complexidade:

- Construção do *heap* (linha 6): $O(n)$
- retira_min (linha 9) tem complexidade $(\log n)$ e é executado n vezes: $O(n \log n)$
- refaz_heap (linha 13) tem complexidade $(\log n)$ e é executado m vezes: $O(m \log n)$
- Complexidade total: $O((n+m) \log n)$

Problemas do Caminho Mínimo

- **Caminho mínimo de fonte única:** algoritmo de Dijsktra;

Problemas do Caminho Mínimo

- **Caminho mínimo de fonte única:** algoritmo de Dijsktra;
- **Caminho mínimo de destino único:** inverta a direção das arestas e aplique algoritmo de Dijstra;

Problemas do Caminho Mínimo

- **Caminho mínimo de fonte única:** algoritmo de Dijsktra;
- **Caminho mínimo de destino único:** inverta a direção das arestas e aplique algoritmo de Dijstra;
- **Caminho mínimo entre quaisquer vértices *u* e *v*:** algoritmo de Dijsktra;

Problemas do Caminho Mínimo

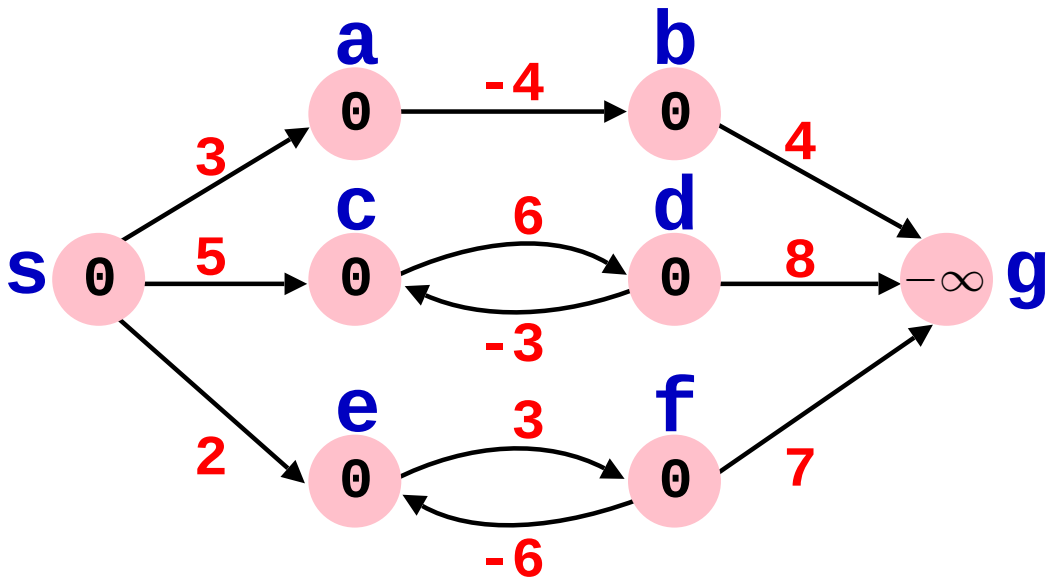
- **Caminho mínimo de fonte única:** algoritmo de Dijsktra;
- **Caminho mínimo de destino único:** inverta a direção das arestas e aplique algoritmo de Dijstra;
- **Caminho mínimo entre quaisquer vértices u e v:** algoritmo de Dijsktra;
- **Caminho mínimo de todos os vértices para todos os vértices:** algoritmo de Floyd-Warshall em tempo $O(n^3)$.

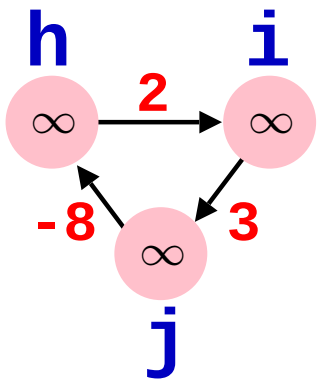
Limitações do algoritmo de Dijkstra

- Não funciona para grafos que contém ciclos com pesos negativos acessíveis a partir da origem;

Limitações do algoritmo de Dijkstra

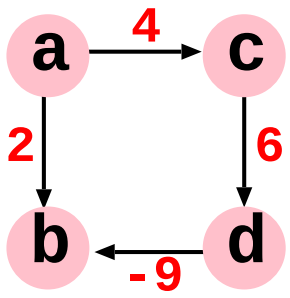
- Não funciona para grafos que contém ciclos com pesos negativos acessíveis a partir da origem;
- **Alternativa:** Algoritmo de Bellman-Ford de complexidade $O(n \cdot m)$;





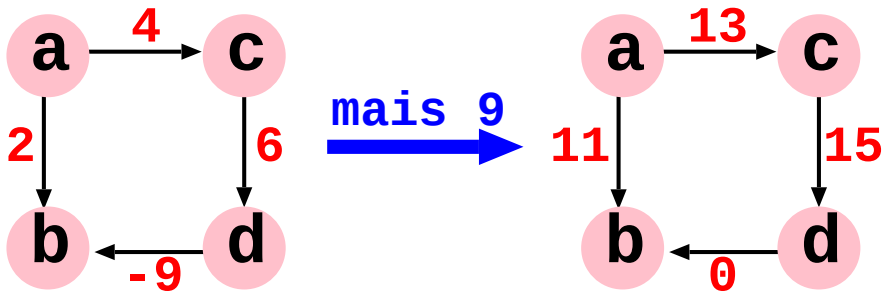
Caminho Mínimo em Grafos com Pesos Negativos

- E se adicionarmos uma constante ao peso de cada aresta?



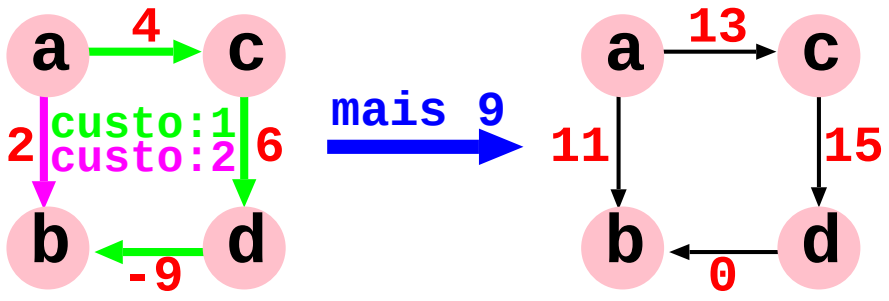
Caminho Mínimo em Grafos com Pesos Negativos

- E se adicionarmos uma constante ao peso de cada aresta?



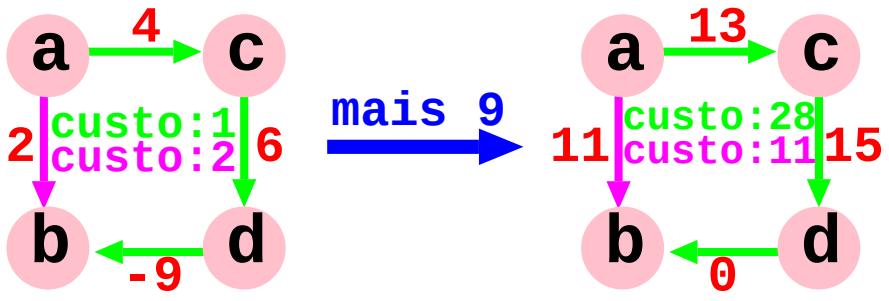
Caminho Mínimo em Grafos com Pesos Negativos

- E se adicionarmos uma constante ao peso de cada aresta?



Caminho Mínimo em Grafos com Pesos Negativos

- E se adicionarmos uma constante ao peso de cada aresta?



Bibliografia Utilizada

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. e STEIN, C. *Introduction to Algorithms*, 3ª edição, MIT Press, 2009.