Caminhos mínimos de única origem

Marco A L Barbosa

malbarbo.pro.br

Departamento de Informática
Universidade Estadual de Maringá

Este trabalho está licenciado com uma Licença Creative Commons - Atribuição-CompartilhaIgual 4.0 Internacional.
http://github.com/malbarbo/na-grafos

Introdução

Introdução

- Como encontrar o caminho mínimo entre duas cidades?
- Vamos estudar este tipo de problema, que é conhecido como **problema de caminho mínimo**
- Entrada
  - Um grafo orientado $G = (V, E)$
  - Uma função peso $w : E \to \mathbb{R}$

- O **peso do caminho** $p = \langle v_0, v_1, \ldots, v_k \rangle$ é a soma dos pesos das arestas no caminho
  - $w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$

- O **peso do caminho mínimo** $\delta(u, v)$ deste u até v é

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

- Um **caminho mínimo** do vértice u até o vértice v é qualquer caminho p com peso $w(p) = \delta(u, v)$

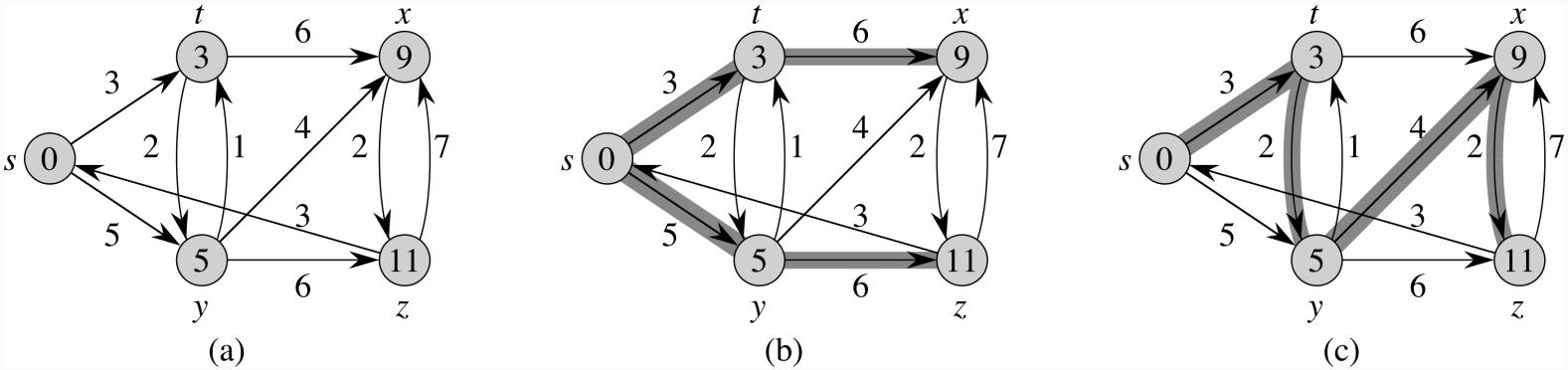
- Os pesos das arestas podem representar outras métricas além da distância, como o tempo, custo, ou outra quantidade que acumule linearmente ao longo de um caminho e que se deseja minimizar
- O algoritmo de busca em largura é um algoritmo de caminhos mínimos que funciona para grafos não valorados, isto é, as arestas tem peso unitário

Tipos de problemas de caminho mínimo

- **Origem única**: Encontrar um caminho mínimo a partir de uma dada origem $s \in V$ até todo vértice $v \in V$
- **Destino único**: Encontrar um caminho mínimo até um determinado vértice de destino t a partir de cada vértice v
- **Par único**: Encontrar o caminho mínimo de u até v
- **Todos os pares**: Encontrar um caminho mínimo deste u até v para todo par de vértices u e v

Exemplo

- Exemplo de caminhos mínimos de única origem



- Observe que
  - O caminho mínimo pode não ser único
  - Os caminhos mínimos de uma origem para todos os outros vértices formam uma árvore

Caminhos mínimos

- Veremos algumas características dos caminho mínimos

Subestrutura ótima

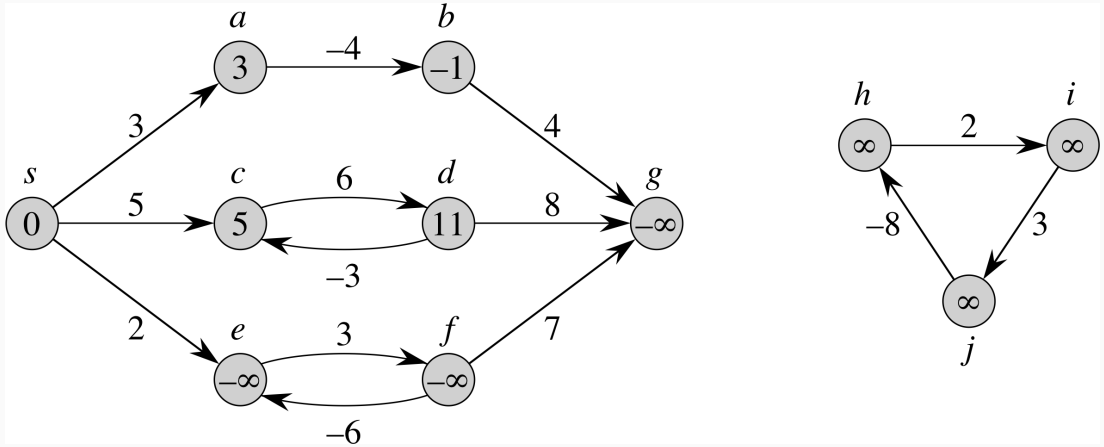
Em geral os algoritmos de caminhos mínimos se baseiam na seguinte propriedade

Lema 24.1
Qualquer subcaminho de um caminho mínimo é um caminho mínimo

Prova
Ideia da prova: se um subcaminho não for mínimo podemos trocá-lo por um subcaminho mínimo é obter um caminho menor, o que é uma contradição pois o caminho é mínimo!

Arestas com pesos negativos

- Não apresentam problemas se nenhum ciclo com peso negativo é alcançável a partir da origem
- Nenhum caminho da origem até um vértice em um ciclo negativo pode ser mínimo
- Se existe um ciclo de peso negativo em algum caminho de s até v, definimos $\delta(s, v) = -\infty$



Ciclos

- Caminhos mínimos podem conter ciclos? Não
  - Peso negativo, acabamos de descartar
  - Peso positivo, podemos obter um caminho mínimo eliminando o ciclo
  - Peso nulo, não existe razão para usar tal ciclo
- Qualquer caminho acíclico em um grafo $G = (V, E)$ contém no máximo $|V|$ vértices distintos e no máximo $|V| - 1$ arestas
  - Desta forma, vamos restringir a nossa atenção para ciclos com no máximo $|V| - 1$ arestas

Representação

- Representamos os caminhos mínimos de forma semelhante as árvores produzidas pelo BFS

Representação

- Para cada vértice $v \in V$, a saída dos algoritmos consiste em
  - $v.d = \delta(s, v)$
    - Inicialmente $v.d = \infty$
    - Diminui conforme o algoritmo progride, mas sempre mantém a propriedade $v.d \geq \delta(s, v)$
    - Vamos chamar $v.d$ de **estimativa do caminho mínimo**
  - $v.\pi =$ predecessor de v no caminho mínimo a partir de s
    - Se não existe predecessor, então $v.\pi = \text{NIL}$
    - π induz uma árvore, a árvore de caminhos mínimos

Relaxamento

INITIALIZE-SINGLE-SOURCE(G, s)

1 **for** $v \in G.V$

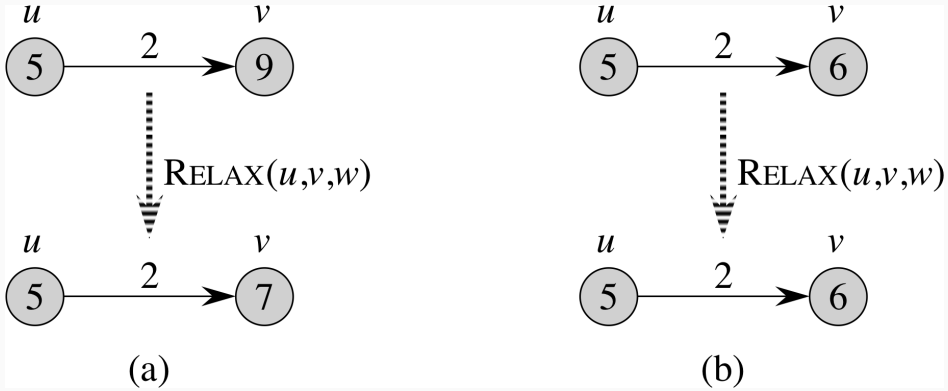
2 $\quad v.d = \infty$

3 $\quad v.\pi = \text{NIL}$

4 $s.d = 0$

Relaxamento

Sendo os vértices inicializados com INITIALIZE-SINGLE-SOURCE, podemos melhorar a estimativa do caminho mínimo para v, indo através de u e seguindo (u, v)?



RELAX(u, v, w)

1 **if** $v.d > u.d + w(u, v)$
2 $\quad v.d = u.d + w(u, v)$
3 $\quad v.\pi = u$

Propriedades

- Desigualdade de triângulos (Lema 24.10)
  - Para toda $(u, v) \in E$, temos que $\delta(s, v) \leq \delta(s, u) + w(u, v)$
- Para as próximas propriedades supomos que
  - O grafo é inicializado com uma chamada a INITIALIZE-SINGLE-SOURCE
  - O único modo de modificar $v.d$ e $v.\pi$ (para qualquer vértice) e pela chamada de RELAX

Propriedades

- Propriedade do limite superior (Lema 24.11)
  - Sempre temos $v.d \geq \delta(s, v)$ para todo v. Uma vez que $v.d = \delta(s, v)$, ele nunca muda
- Propriedade de nenhum caminho (Corolário 24.12)
  - Se $\delta(s, v) = \infty$, então sempre $v.d = \infty$

Propriedades

- Propriedade de convergência (Lema 24.14)
  - Se $s \rightsquigarrow u \rightarrow v$ é um caminho mínimo, $u.d = \delta(s, u)$ e RELAX(u, v, w) é chamado, então, em todos os momentos após a chamada, temos $v.d = \delta(s, v)$
- Propriedade de relaxamento de caminho (Lema 24.15)
  - Seja $p = \langle v_0, v_1, \ldots, v_k \rangle$ o caminho mínimo de $s = v_0$ até v_k, se a função RELAX for chamada na ordem $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, mesmo que intercalada com outros relaxamentos, então $v_k.d = \delta(s, v_k)$

Propriedades

- Propriedade do subgrafo-predecessor
  - Uma vez que $v.d = \delta(s, v)$ for todo $v \in V$, o subgrafo predecessor é uma árvore de caminhos mínimos enraizada em s

Ideia dos algoritmos

- Os algoritmos que veremos usam a mesma ideia
  - inicializar os atributos $v.d$ e $v.\pi$
  - relaxar as arestas
- Eles diferem na ordem e na quantidade de vezes que cada aresta é relaxada

Algoritmo de Bellman-Ford

Algoritmo de Bellman-Ford

- Resolve o problema para o caso geral, as arestas podem ter pesos negativos
    - Se ciclos negativos acessíveis a partir da origem forem encontrados o algoritmo devolve FALSE, caso contrário, devolve TRUE
    - Calcula $v.d$ e $v.\pi$ para todo $v \in V$
- Ideia
    - Relaxar todas as arestas, $|V| - 1$ vezes

Algoritmo de Bellman-Ford

BELLMAN-FORD(G, w, s)

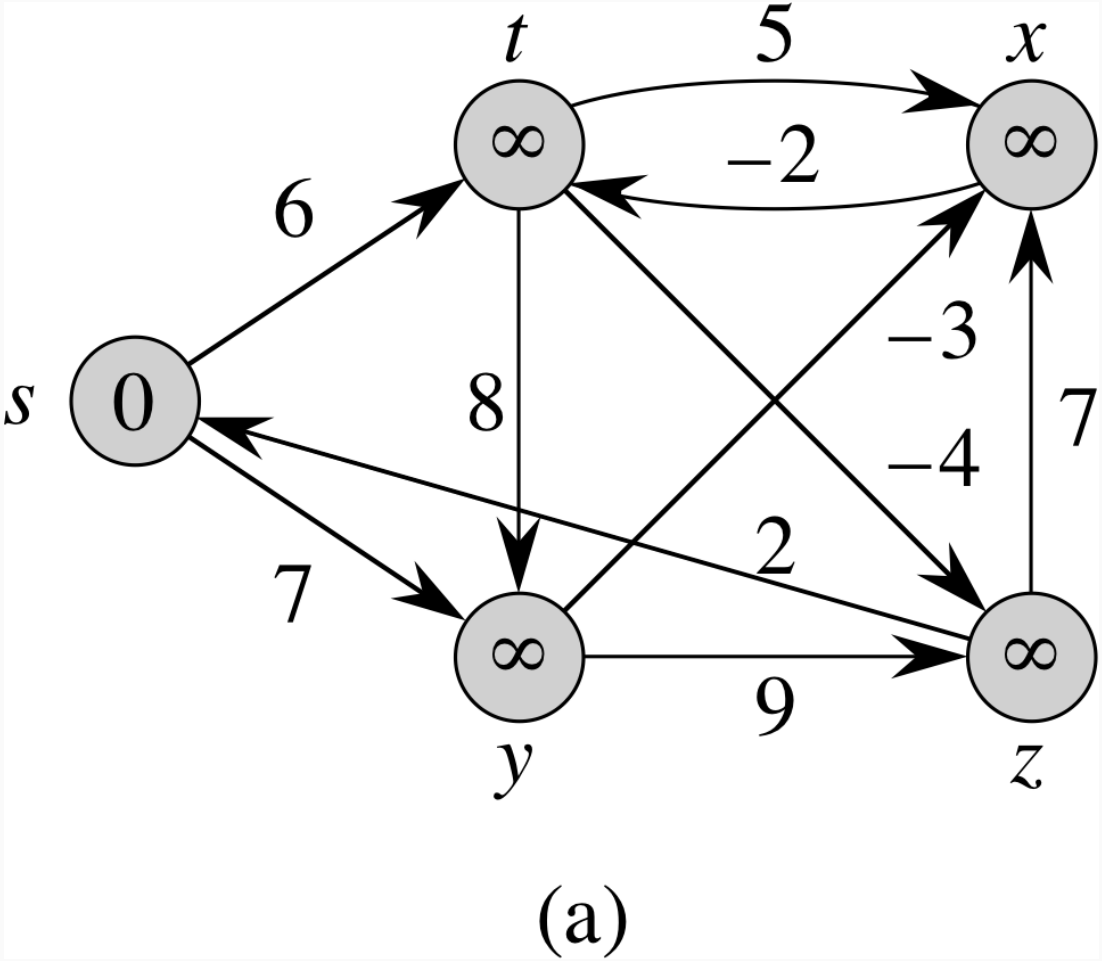
1 INITIALIZE-SINGLE-SOURCE(G, s)
2 **for** $i = 1$ **to** $|G.V| - 1$
3 **for** $(u, v) \in G.E$
4 RELAX(u, v, w)
5 **for** $(u, v) \in G.E$
6 **if** $v.d > u.d + w(u, v)$
7 **return** FALSE
8 **return** TRUE

Análise do tempo de execução

- A inicialização na linha 1 demora $\Theta(V)$
- Cada uma das $|V| - 1$ passagens das linha 2 a 4 demora o tempo $\Theta(E)$, totalizando $O(V \cdot E)$
- O laço das linha 5 a 7 demora $O(E)$
- Tempo de execução do algoritmo $\Theta(V \cdot E)$

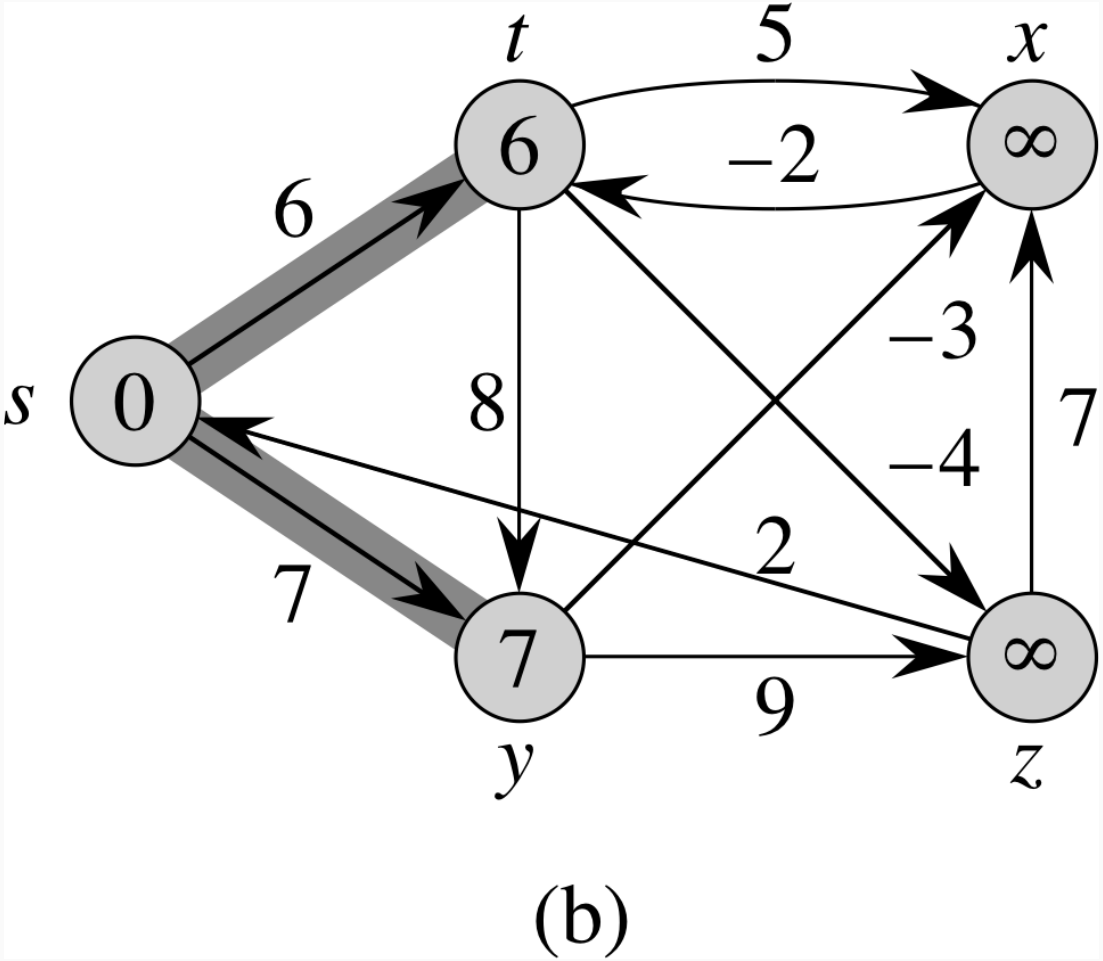
Relaxação das arestas na ordem

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



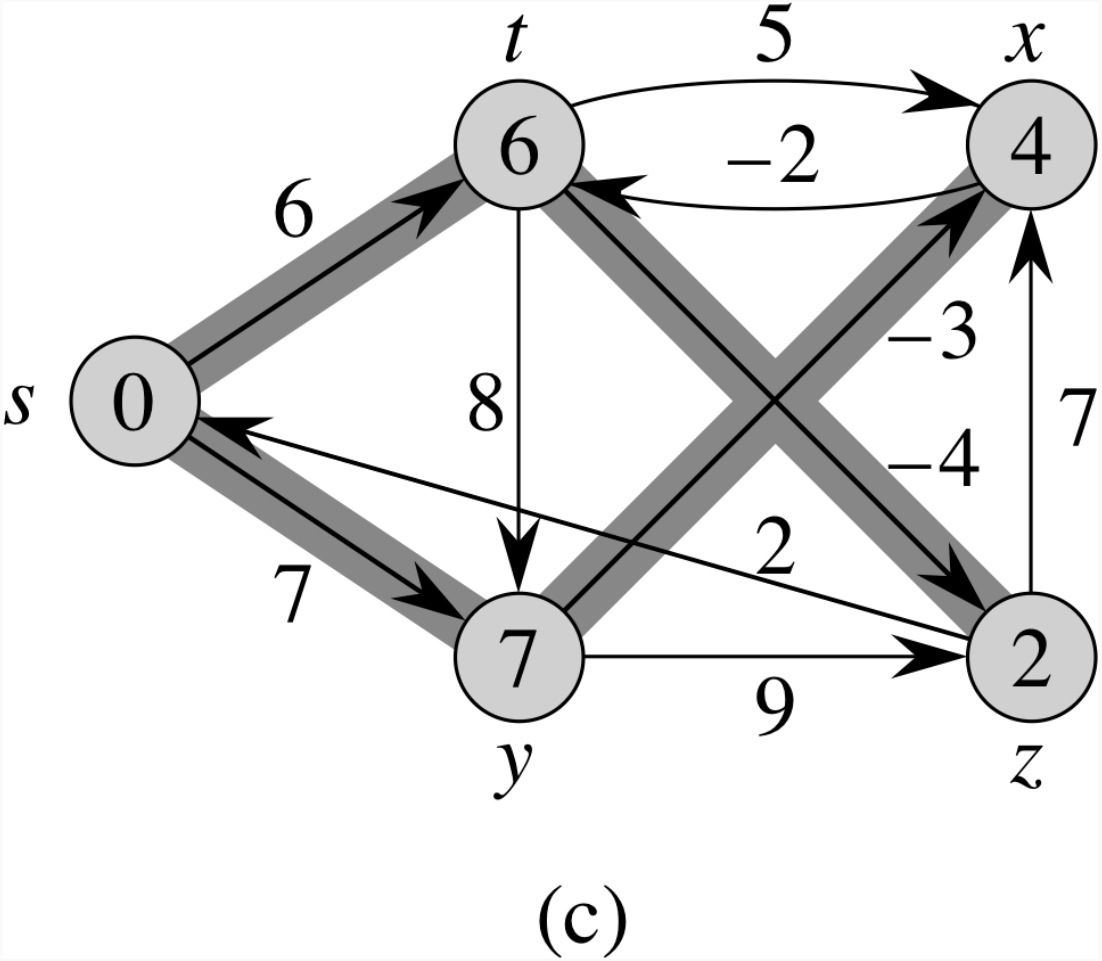
Relaxação das arestas na ordem

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



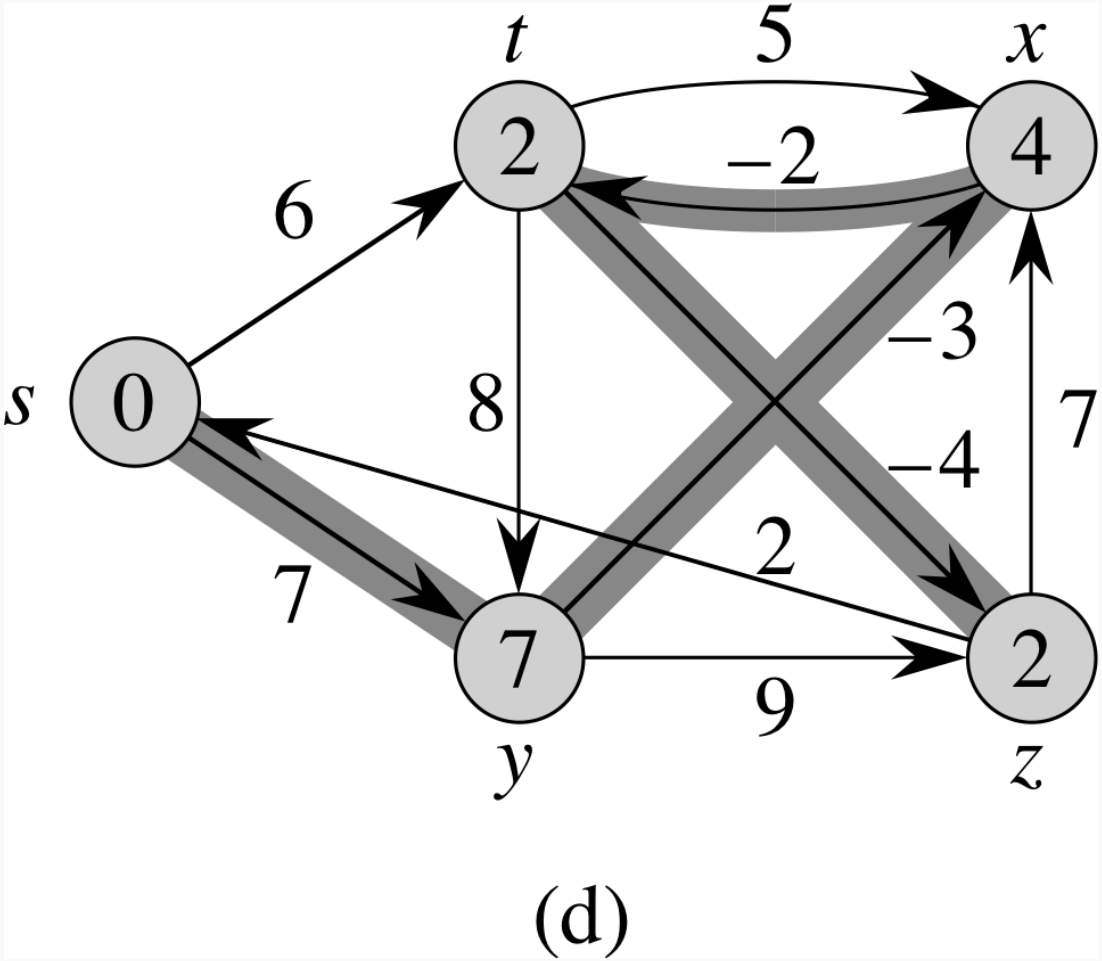
Relaxação das arestas na ordem

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



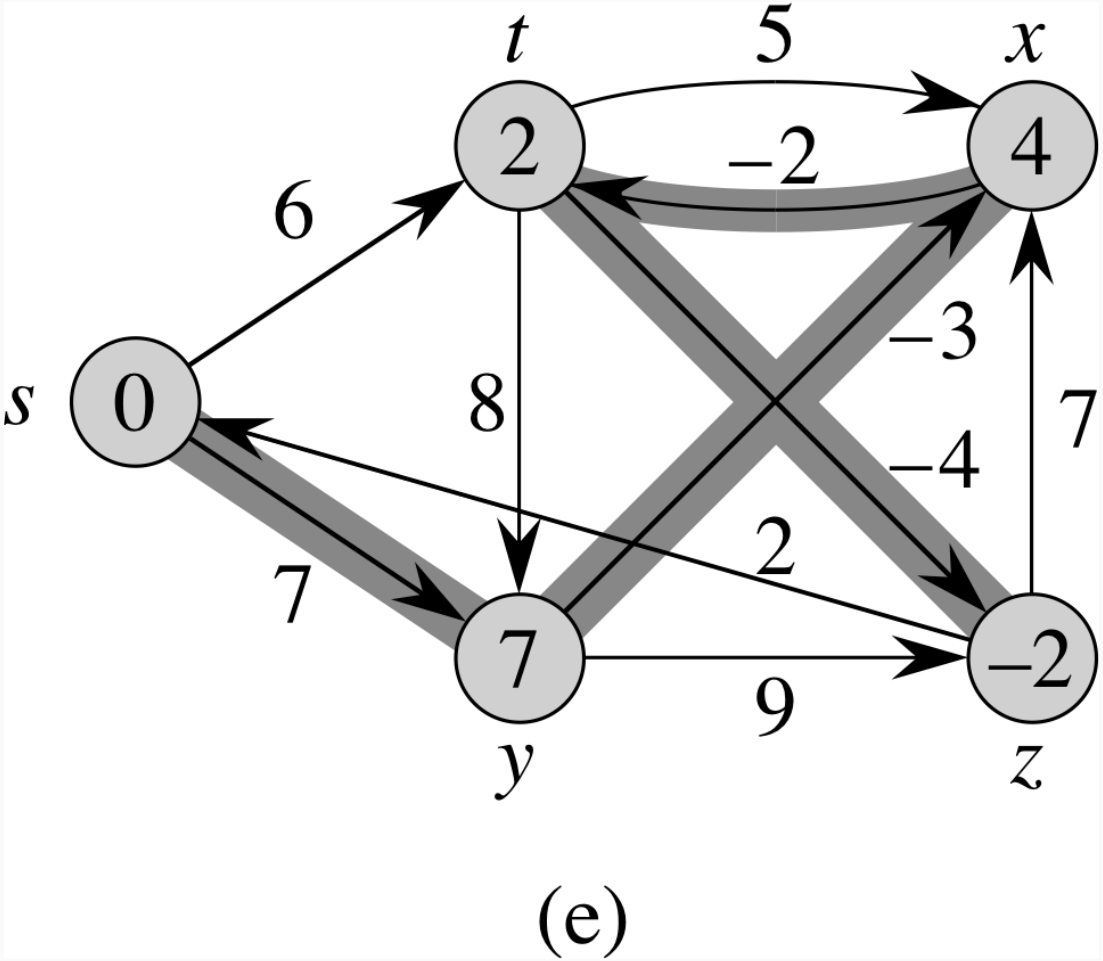
Relaxação das arestas na ordem

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



Relaxação das arestas na ordem

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



Algoritmo de Bellman-Ford

- Por que este algoritmo funciona?
  - Propriedade de relaxamento de caminho
  - Seja $p = \langle v_0, v_1, \ldots, v_k \rangle$ um caminho mínimo acíclico entre $s = v_0$ e $v = v_k$. O caminho p tem no máximo $|V| - 1$ arestas, e portanto $k \leq |V| - 1$
  - Cada iteração do laço da linha 2 relaxa todas as arestas
    - A primeira iteração relaxa (v_0, v_1)
    - A segunda iteração relaxa (v_1, v_2)
    - ...
    - A k-ésima iteração relaxa (v_{k-1}, v_k)
  - Pela propriedade de relaxamento de caminho $v.d = v_k.d = \delta(s, v_k) = \delta(s, v)$

Grafo orientado acíclico

Algoritmo para gaos

- Em grafo acíclico orientado (gao), os caminhos mínimos são sempre bem definidos (não existem ciclos de peso negativo)
- Ideia
    - Relaxar as arestas em uma ordem topológica dos vértices

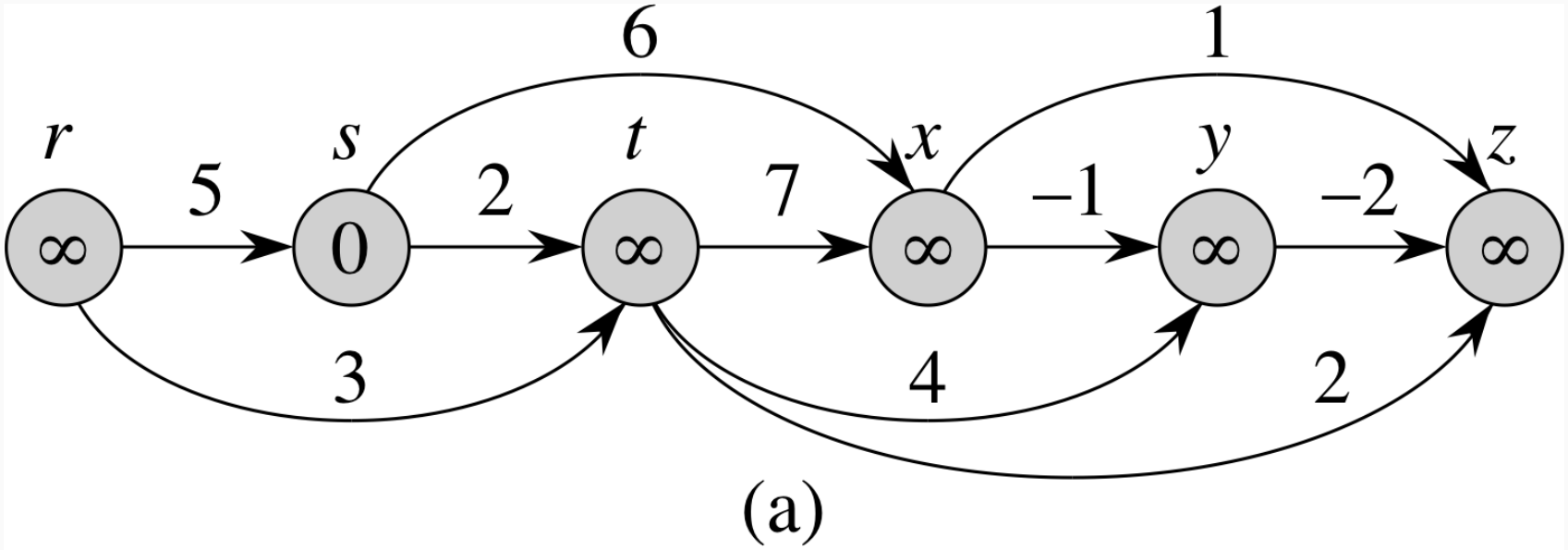
Caminhos mínimos de única origem em gaos

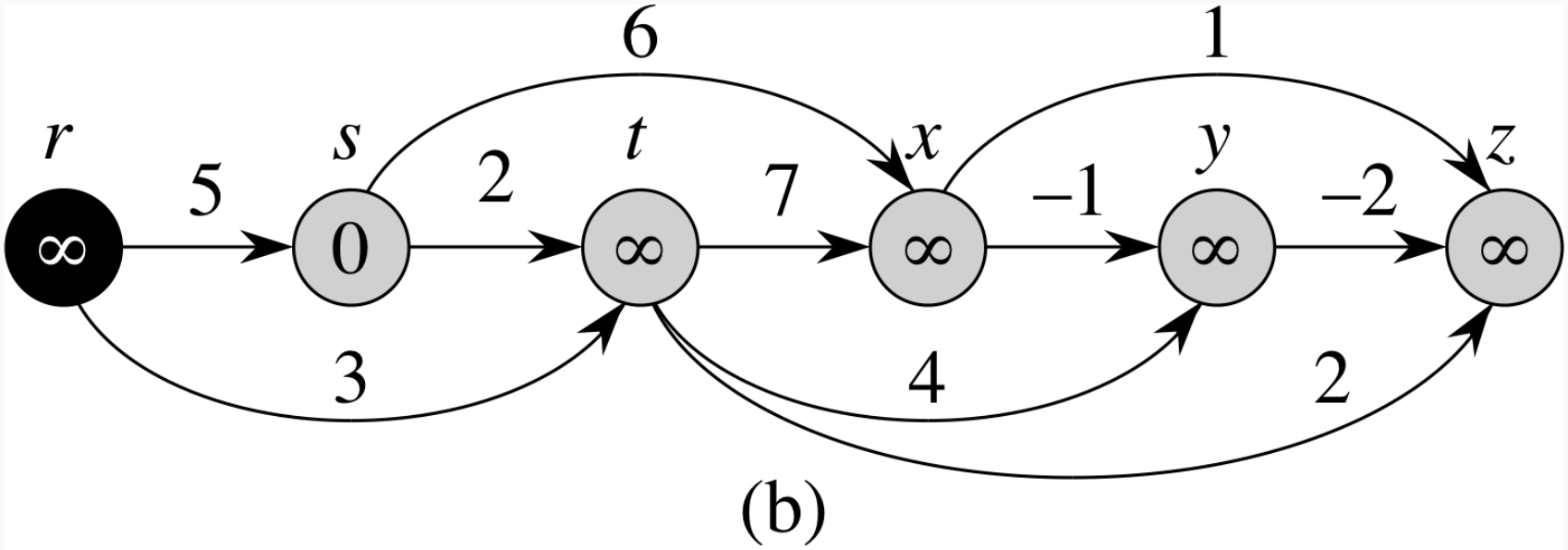
DAG-SHORTEST-PATHS(G, w, s)

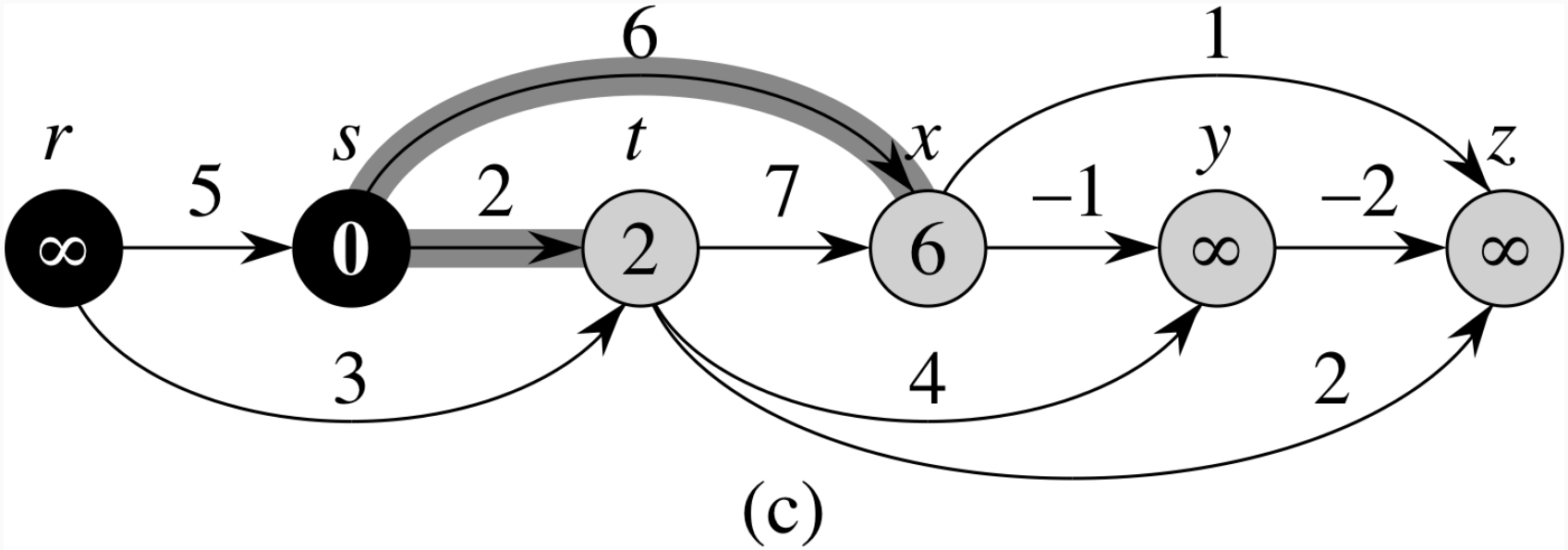
1 ordenar topologicamente $G.V$
2 INITIALIZE-SINGLE-SOURCE(G, s)
3 **for** vértice u em ordem topológica
4 **for** $v \in G.adj[u]$
5 RELAX(u, v, w)

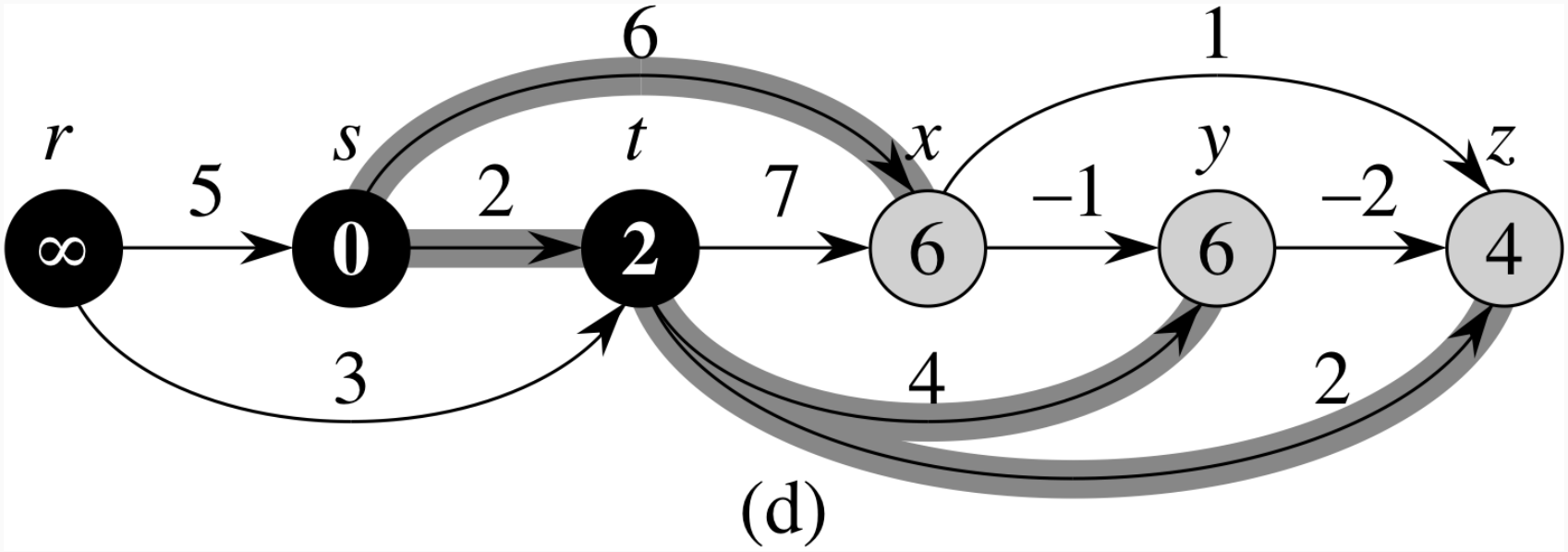
Análise do tempo de execução

- A ordenação topológica da linha 1 tem tempo $\Theta(V + E)$
- INITIALIZE-SINGLE-SOURCE na linha 2 tem tempo $\Theta(V)$
- Nos laços das linhas 2 e 3 a lista de adjacências de cada vértices é visitada apenas uma vez, totalizando $V + E$ (análise agregada), como o relaxamento de cada aresta custa $O(1)$, o tempo total é $\Theta(E)$
- Portanto, o tempo de execução do algoritmo é $\Theta(V + E)$

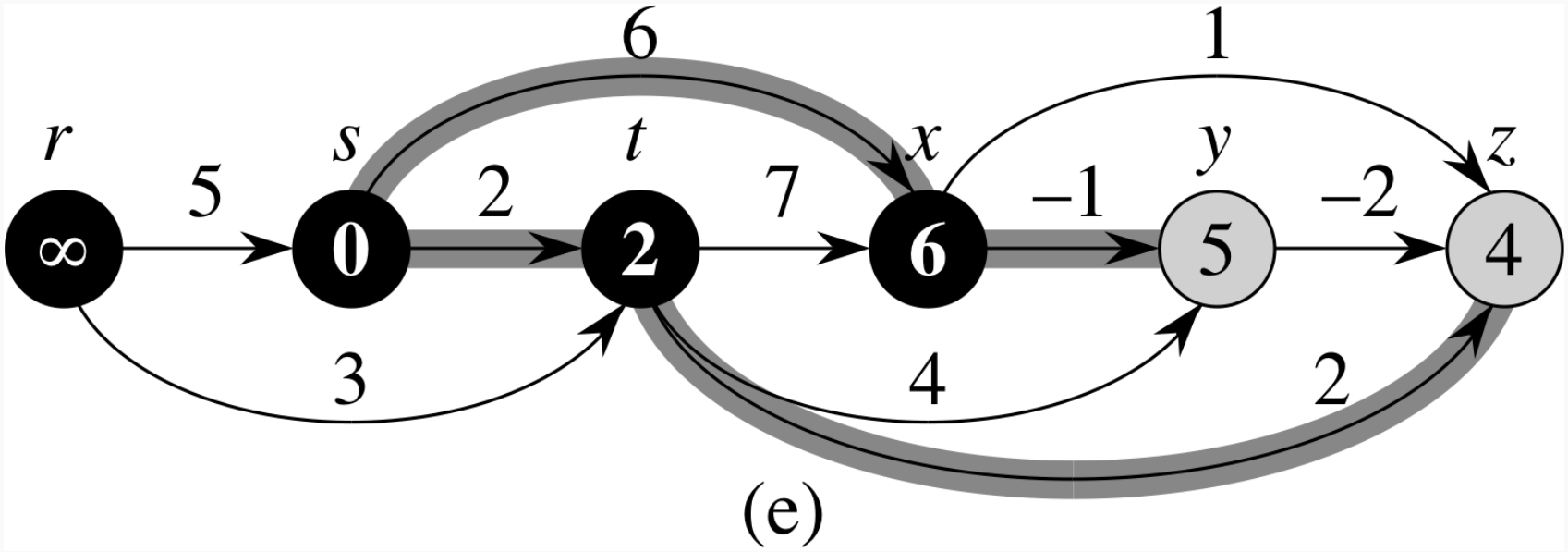

r
s
t
x
y
z
∞
0
∞
∞
∞
∞
6
1
5
2
7
−1
−2
3
4
2
(a)

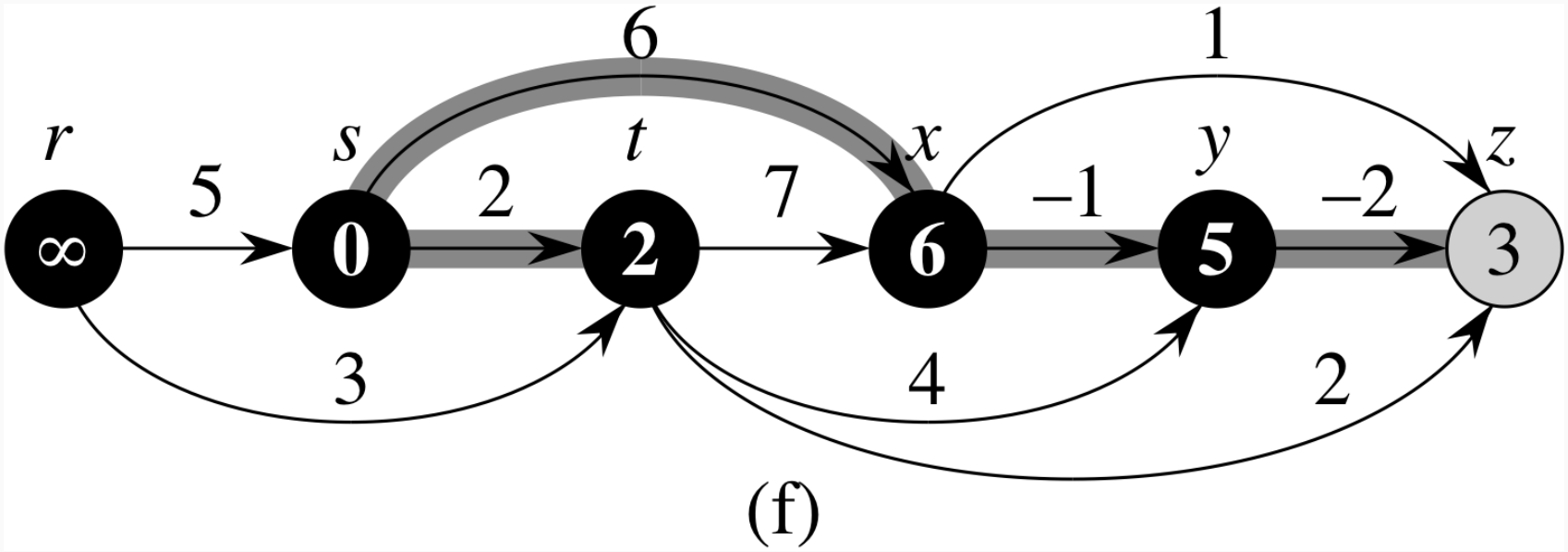

r
s
t
x
y
z
∞
0
∞
∞
∞
∞
6
1
5
2
7
−1
−2
3
4
2
(b)


r
s
t
x
y
z
∞
0
2
6
∞
∞
5
2
7
−1
−2
6
1
3
4
2
(c)

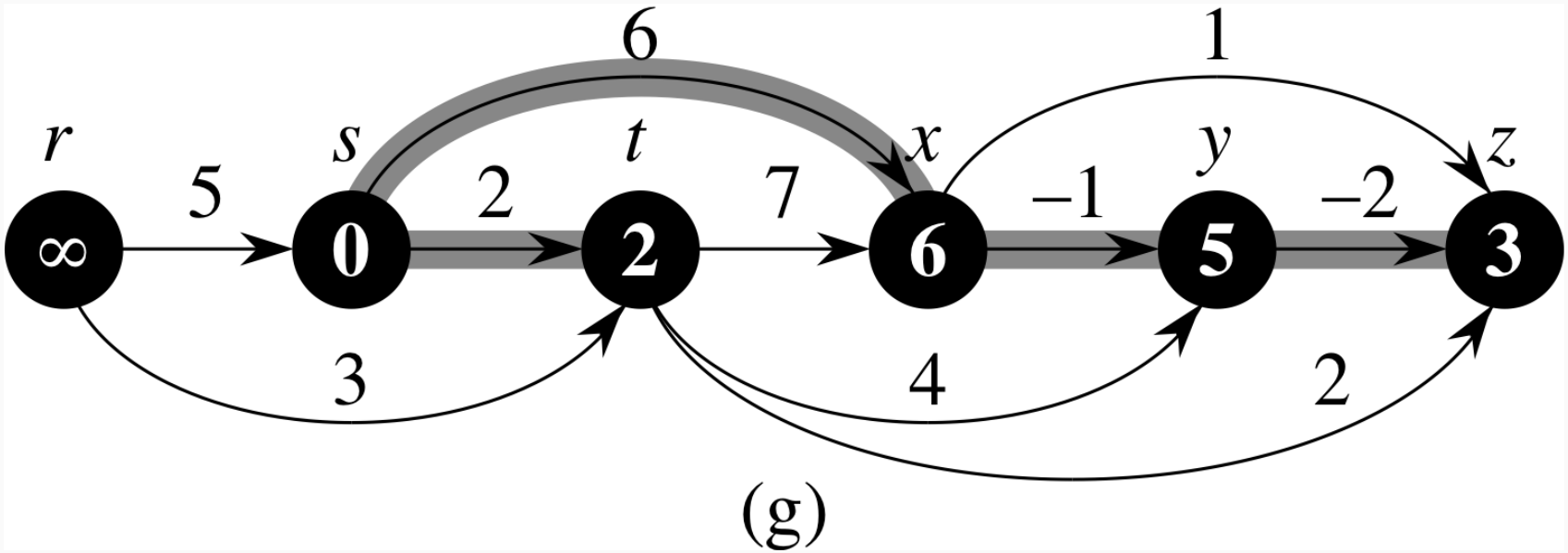

r
s
t
x
y
z
∞
0
2
6
6
4
5
2
7
−1
−2
6
1
3
4
2
(d)

Caminhos mínimos de única origem em gaos




r
s
t
x
y
z
∞
0
2
6
5
3
5
2
7
−1
−2
6
1
3
4
2
(f)

Caminhos mínimos de única origem em gaos



- Por que este algoritmo funciona?
  - Como os vértices são processados em ordem topológica, as arestas de qualquer caminho são relaxadas na ordem que aparecem no caminho
  - Pela propriedade de relaxamento de caminho, o algoritmo funciona corretamente

Aplicação

- Caminhos críticos na análise de diagramas PERT (*program evaluation and review technique*)
- As arestas representam serviços a serem executados
- Os pesos de arestas representam os tempos necessários para execução de determinados serviços
- (u, v), v, (v, x): serviço (u, v) deve ser executado antes do serviço (v, x)

Aplicação

- Um caminho através desse gao: sequência de serviços
- Caminho crítico: é um caminho mais longo pelo gao
    - Tempo mais longo para execução de uma sequência ordenada
- O peso de um caminho crítico é um limite inferior sobre o tempo total para execução de todos os serviços

Aplicação

- Podemos encontrar um caminho crítico de duas maneiras:
  - Tornando negativos os pesos das arestas e executando DAG-SHORTEST-PATHS; ou
  - Executando DAG-SHORTEST-PATHS, substituindo "∞" por "$-\infty$" na linha 2 de INITIALIZE-SINGLE-SOURCE e "$>$" por "$<$" no procedimento RELAX

Algoritmo de Dijkstra

Algoritmo de Dijkstra

- Caminho mínimo de única origem em um grafo orientado ponderado
- Todos os pesos de arestas são não negativos, ou seja $w(u, v) \geq 0$ para cada aresta $(u, v) \in E$

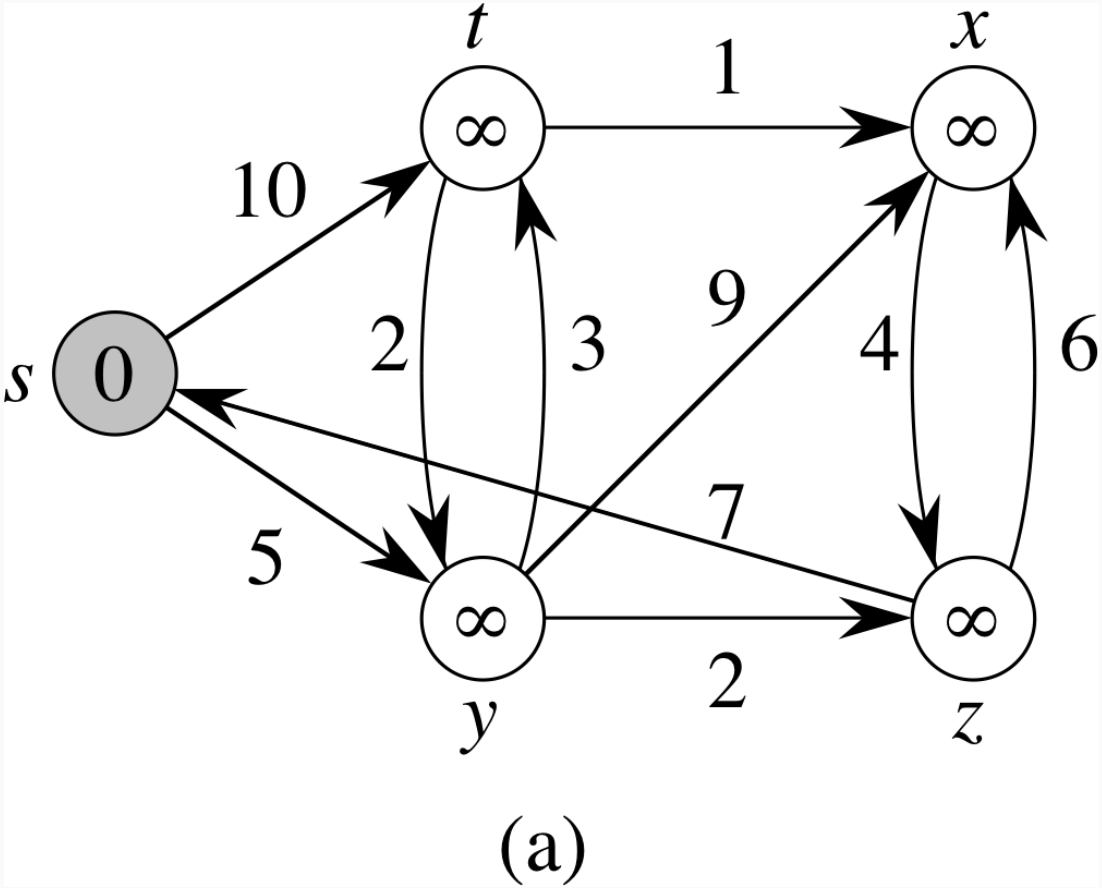
Algoritmo de Dijkstra

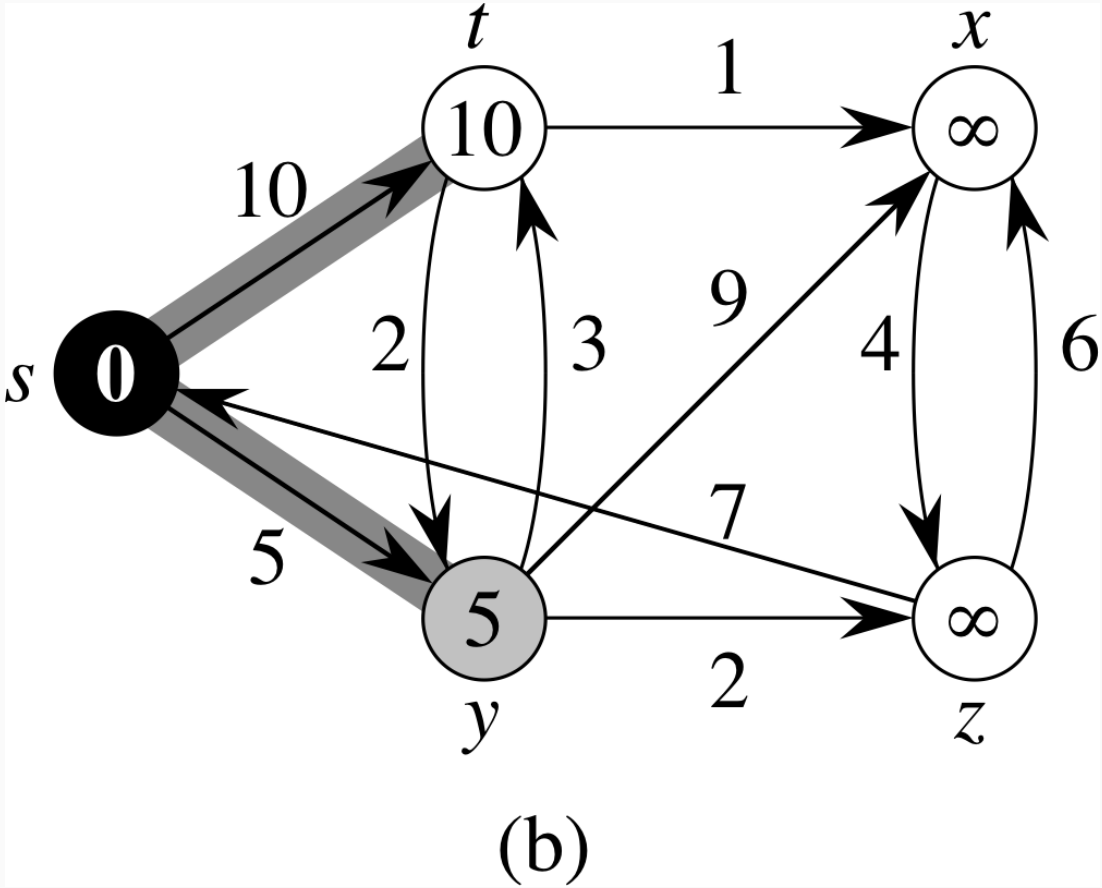
- Ideia
  - Essencialmente uma versão ponderada da busca em largura
    - Ao invés de uma fila FIFO, usa uma fila de prioridades
    - As chaves são os valores $v.d$
  - Mantém dois conjuntos de vértices
    - S: vértices cujo caminho mínimo desde a origem já foram determinados
    - $Q = V - S$: fila de prioridades
  - O algoritmo seleciona repetidamente o vértice $u \in Q$ com a mínima estimativa de peso do caminho mínimo, adiciona u a S e relaxa todas as arestas que saem de u

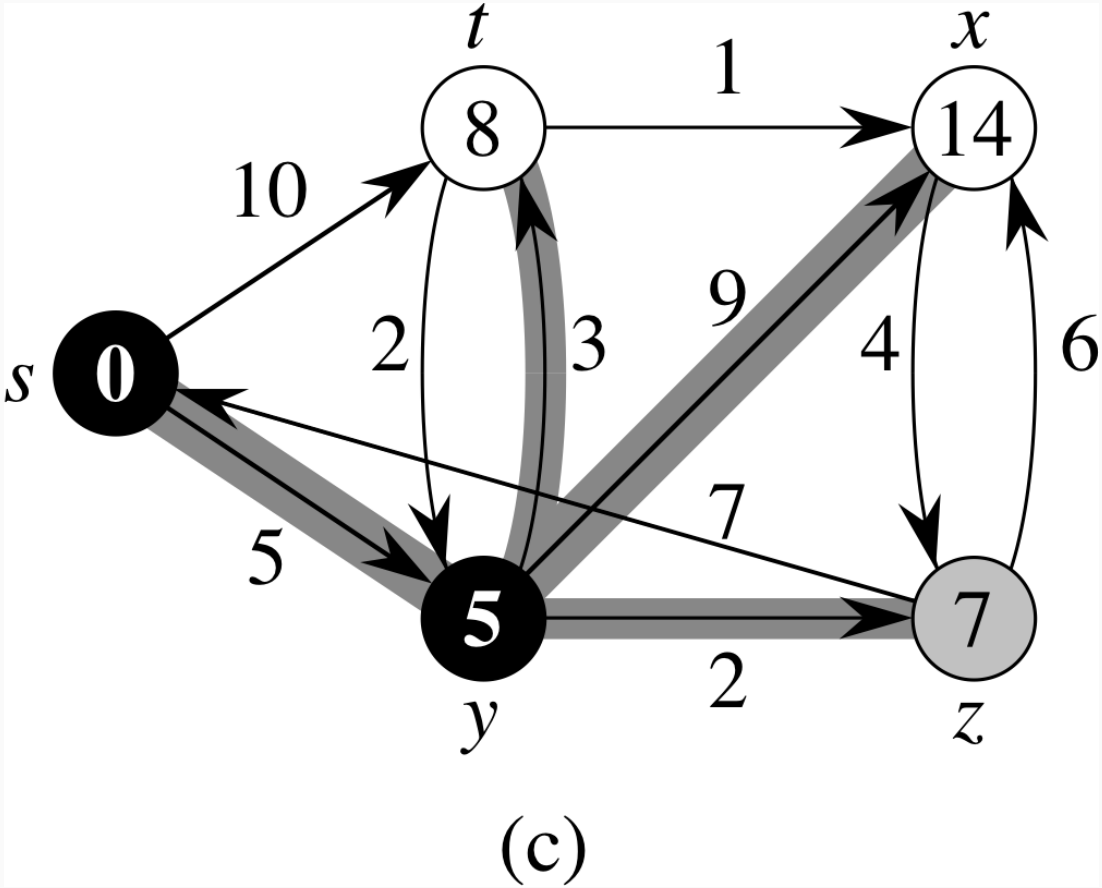
Algoritmo de Dijkstra

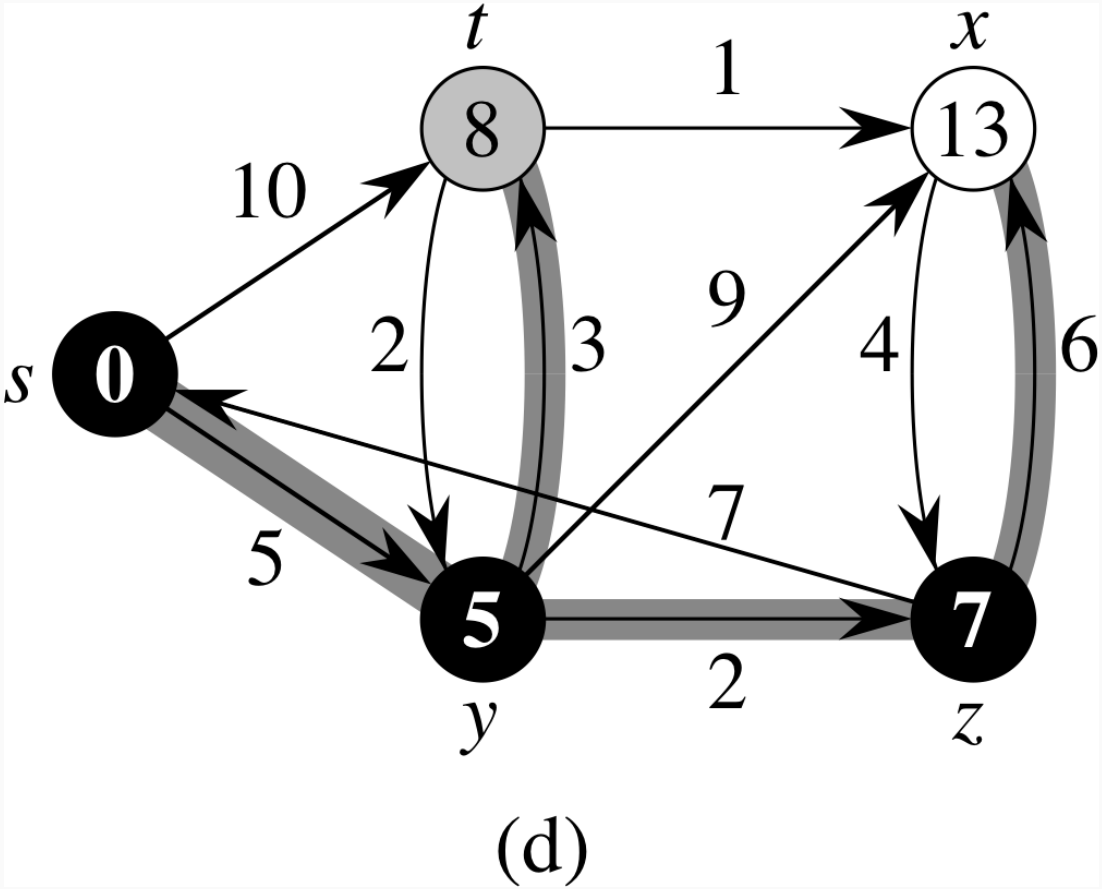
DIJKSTRA(G, w, s)

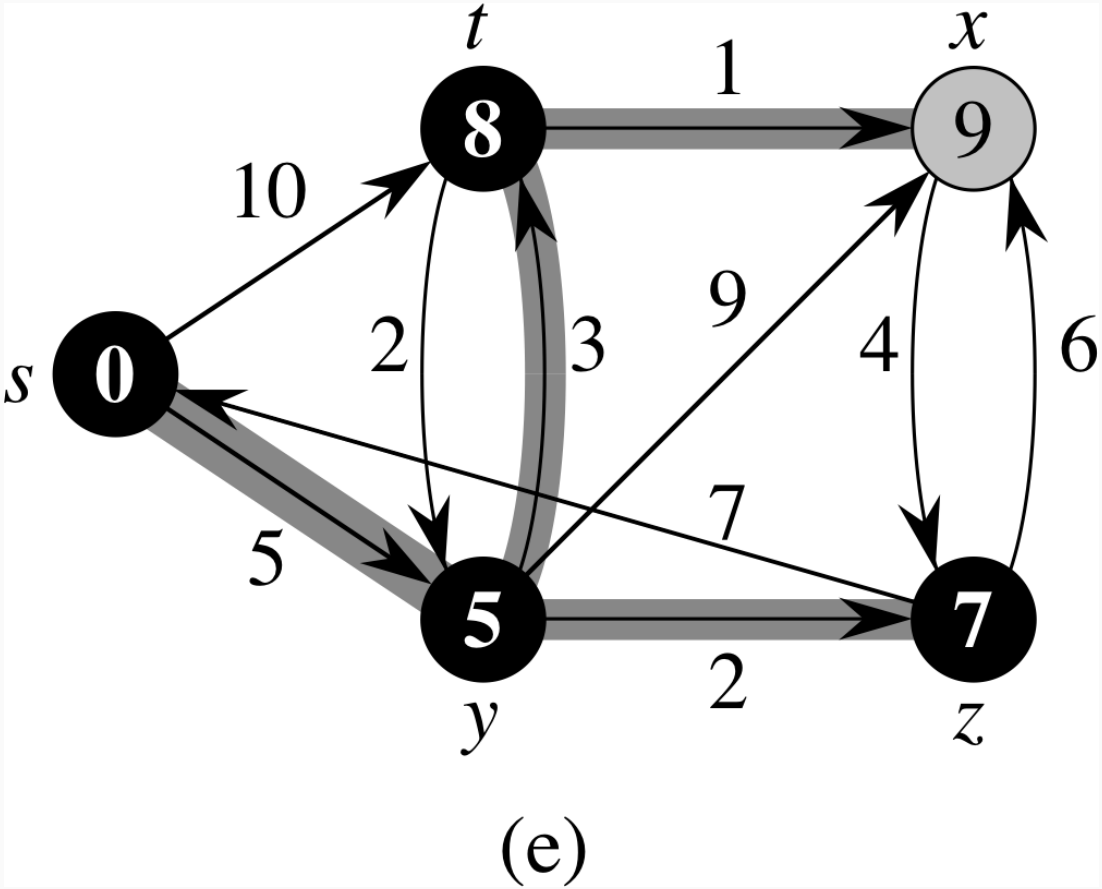
1 INITIALIZE-SINGLE-SOURCE(G, s)
2 $S = \emptyset$
3 $Q = G.V$
4 **while** $Q \neq \emptyset$
5 $\quad u =$ EXTRACT-MIN(Q)
6 $\quad S = S \cup \{u\}$
7 $\quad$ **for** $v \in G.adj[u]$
8 $\quad\quad$ RELAX(u, v, w)

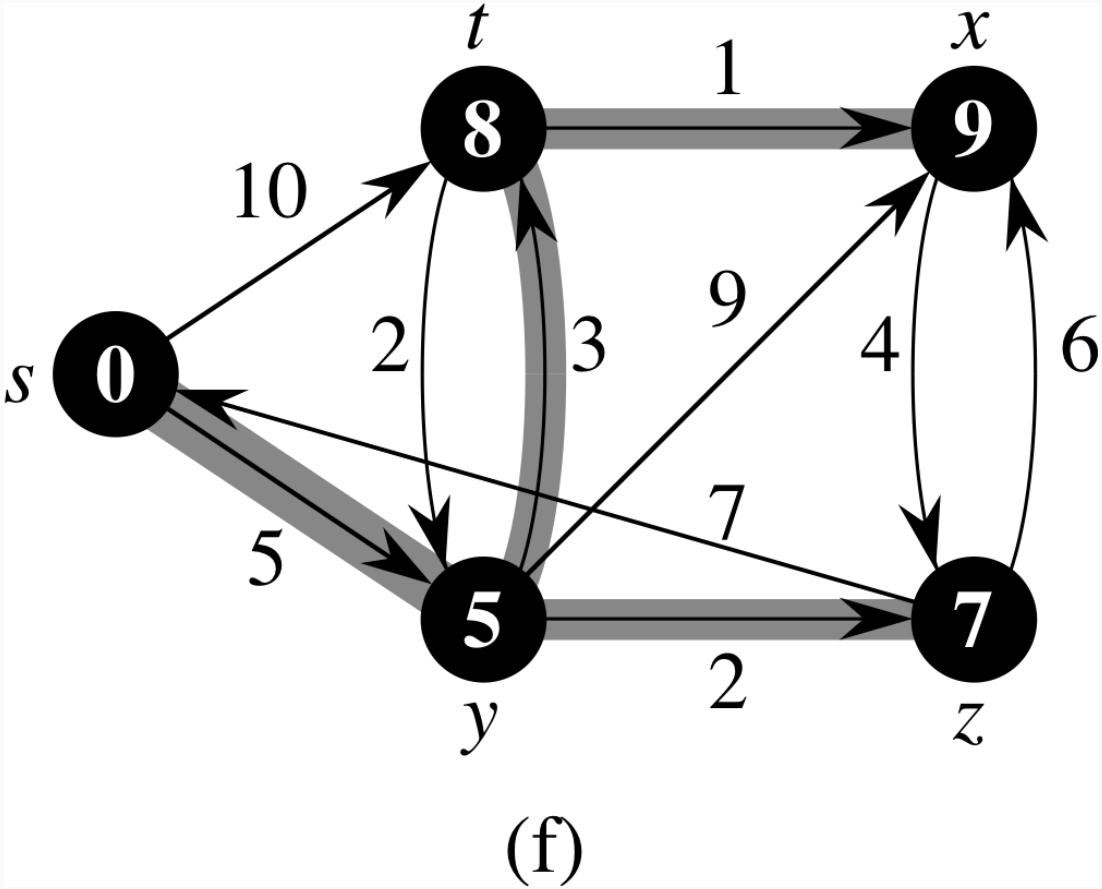

t
x
1
∞
∞
10
9
2
3
4
6
s
0
7
5
∞
∞
2
y
z
(a)


t
x
10
∞
1
10
9
s
0
2
3
4
6
7
5
5
∞
2
y
z
(b)


t
x
1
8
14
10
9
2
3
4
6
s
0
7
5
5
7
2
y
z
(c)


t
x
1
8
13
10
9
s
0
2
3
4
6
7
5
5
7
2
y
z
(d)


t
x
1
8
9
10
9
2
3
4
6
s
0
7
5
5
7
2
y
z
(e)



(f)

Algoritmo de Dijkstra

- Análise do tempo de execução
  - Linha 1 $\Theta(V)$
  - Linhas 3 a 8 $O(V + E)$ (sem contar as operações com fila)
  - Operações de fila
    - INSERT implícita na linha 3 (executado uma vez para cada vértice)
    - EXTRACT-MIN na linha 5 (executado uma vez para cada vértice)
    - DECREASE-KEY implícita em RELAX (executado no máximo de $|E|$ vezes, uma vez para cada aresta relaxada)
  - Depende da implementação da fila de prioridade

Algoritmo de Dijkstra

- Análise do tempo de execução
  - Arranjos simples
    - Como os vértices são enumerados de 1 a $|V|$, armazenamos o valor $v.d$ na v-ésima entrada de um arranjo
    - Cada operação INSERT e DECREASE-KEY demora $O(1)$
    - Cada operação EXTRACT-MIN demora $O(V)$ (pesquisa linear)
    - Tempo total de $O(V^2 + E) = O(V^2)$

- Análise do tempo de execução
  - Heap
    - Se o grafo é esparso, em particular, $E = o(V^2 / \lg V)$ é prático utilizar um heap binário
    - O tempo para construir um heap é $O(V)$
    - Cada operação de EXTRACT-MIN e DECREASE-KEY demora $O(\lg V)$
    - Tempo total de $O((V + E) \lg V + V)$, que é $O(E \lg V)$ se todos os vértices são acessíveis a partir da origem

Algoritmo de Dijkstra

- Análise do tempo de execução
  - Heap de Fibonacci
    - Cada operação EXTRACT-MIN demora $O(\lg V)$
    - Cada operação DECREASE-KEY demora o tempo amortizado de $O(1)$
    - Tempo total de $O(V \lg V + E)$

Algoritmo de Dijkstra

- Porque este algoritmo funciona?
  - Invariante de laço: no início de cada iteração do laço **while** , $v.d = \delta(s, v)$ para todos $v \in S$
  - Inicialização: $S = \emptyset$, então é verdadeiro
  - Término: No final, $Q = \emptyset \Rightarrow S = V \Rightarrow v.d = \delta(s, v)$, para todo $v \in V$
  - Manutenção: precisamos mostrar que $u.d = \delta(s, u)$ quando u é adicionado a S em cada iteração (Comentado em sala, veja o livro para a prova completa)
  - Feito em sala

Referências

- Thomas H. Cormen et al. Introduction to Algorithms. 3rd edition. Capítulo 24.