Governo Federal

Ministério da Educação

deINF departamentodeInformática

Universidade Federal do Maranhão

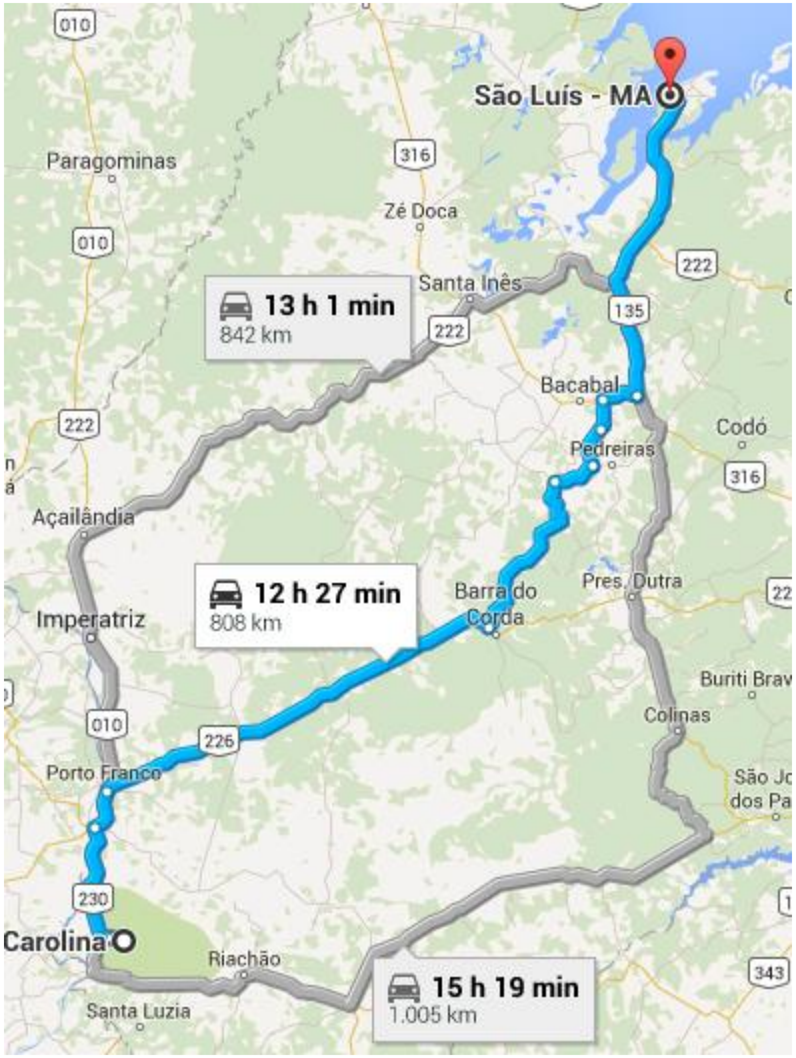
A Universidade que Cresce com Inovação e Inclusão Social

Estrutura de Dados II

Caminhos mínimos
Algoritmo de Dijkstra

Introdução

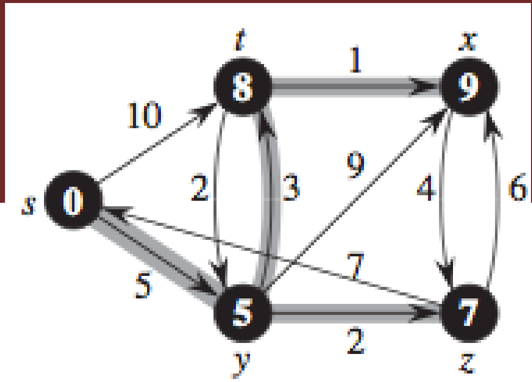
- Caminho mais curto de origem única.
- problema em questão terá diversas possibilidades! Sendo que, a maioria, não vale a pena considerar.



Aplicações

- **Quando o GPS determina a rota mais rápida entre a sua localização atual e um destino**
  - O GPS provavelmente usa um algoritmo que encontra todos os caminhos mínimos que partem de uma fonte única, ma o GPS só retorna o caminho mínimo.

Introdução



- Para o problema do caminho mais curto, temos um grafo orientado e ponderado
  - G(V, E)
- com uma função de peso
  - w: E -> R (números reais),
    - mapeando arestas em valores reais.
- Para o caminho p = $<v_0, v_1, v_2, \ldots, v_k>$
  - o seu peso é o *somatório dos pesos de suas arestas*:

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

caminho entre s e x:
p=<s, y, t, x>

peso do caminho entre s e x:
w(p)=w(s,y)+ w(y,t) + w(t,x)
= 5 + 3 + 1
= 9

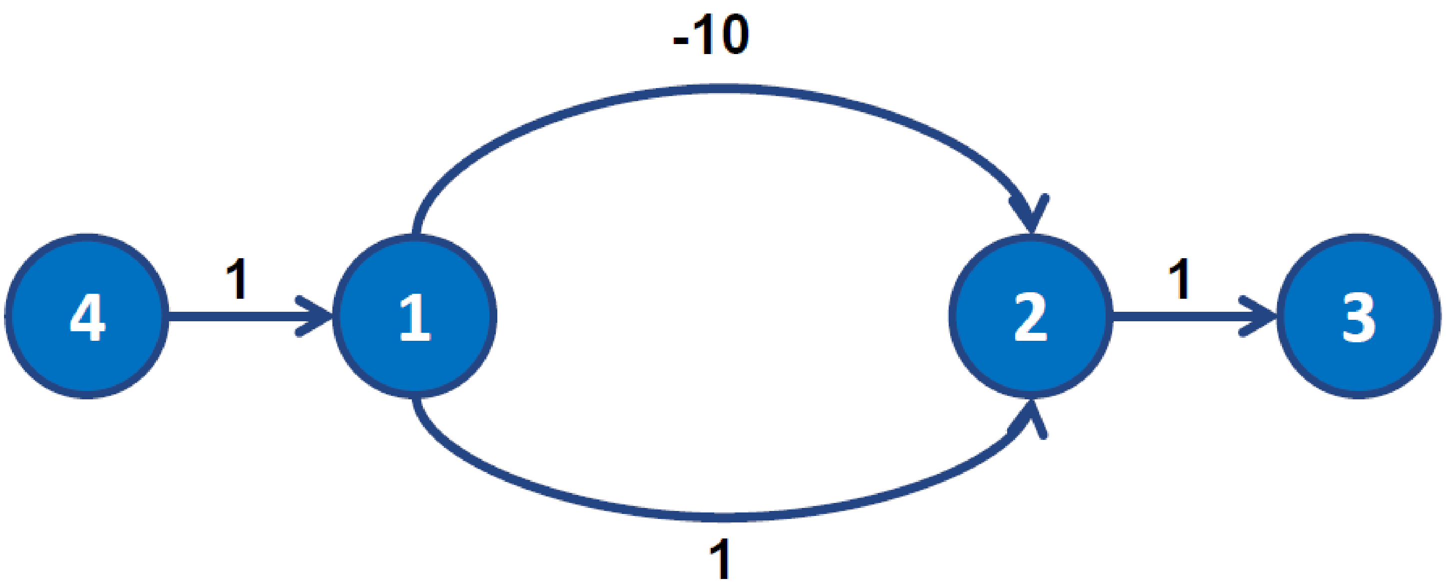
FONTE: http://professor.unisinos.br/pjaques/material/est2/13-grafos-caminhos-minimos.pdf

Caminho mínimo - variantes

- **Variações do problema:**
  - Caminhos mais curtos de origem única;
    - Encontrar o caminho desde um vértice de origem s ϵ V até todo vértice v ϵ V
  - Caminhos mais curtos de destino único;
    - Caminha mais curto até um vértice t a partir de cada vértice v
  - Caminho mais curto de único par;
    - Caminho mais curto de desde u até v, para determinados u e v
  - Caminho mais curtos de todos para todos;
    - Caminho mais curto desde u até v para todo par de vértices u e v

Caminho mínimo – Pesos negativos

- **Em algumas instâncias do problema de caminhos mínimos com fonte única, podem existir arestas com pesos negativos**
- **Se o grafo G = (V, E) não tiver ciclo com peso negativo alcançável a partir da fonte s, então (s, v) permanece bem definido para todo v ∈ V, mesmo se o grafo contiver algum ciclo negativo**



Caminho mínimo – Pesos negativos

- **Alguns algoritmos, como o algoritmo de Dijkstra, assumem que todos os pesos são não-negativos**

- **Algoritmos como Bellman-Ford e Floyd-Warshall, entretanto, podem operar com arestas de peso negativo, desde que não existam ciclos de peso negativo**

- **Tipicamente, estes algoritmos detectam a presença de ciclos negativos**

Caminho mínimo – Pesos negativos

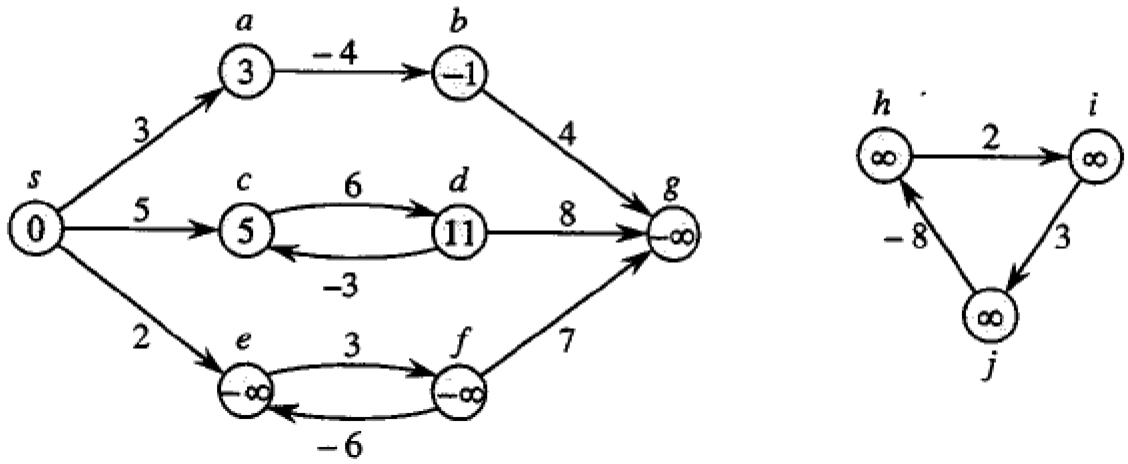


FIGURA 24.1 Pesos de arestas negativos em um grafo orientado. Mostramos dentro de cada vértice o peso de seu caminho mais curto a partir da origem *s*. Como os vértices *e* e *f* formam um ciclo de peso negativo acessível a partir de *s*, eles têm pesos de caminhos mais curtos iguais a −∞. Como o vértice *g* é acessível a partir de um vértice cujo peso de caminho mais curto é −∞, ele também tem um peso de caminho mais curto igual a −∞. Vértices como *h*, *i* e *j* não são acessíveis a partir de *s*, e assim os pesos de seus caminhos mais curtos são iguais a ∞, embora eles residam em um ciclo de peso negativo

Caminhos Mínimos - Representação

- Estamos interessados não apenas na distância do caminho mais curto, mas também no caminho em si

- A representação do caminho mais curto é similar àquela usada na busca em largura (BFS)

- Dado um grafo G = (V, E), mantemos para cada vértice v $\in$ V o seu predecessor □[v] que é outro vértice ou *nil*

- Os algoritmos de caminhos mínimos definem os atributos □ de maneira que a cadeia de predecessores originada em v, de trás para frente, nos dá o caminho mais curto de s para v

Árvore de Caminho mais curto

- **Uma árvore de caminhos mais curtos com raiz em $u \in V$ é um subgrafo direcionado G = (V , A), onde $V \subseteq V$ e $A \subseteq A$, tal que:**
  - 1. V é o conjunto de vértices alcançáveis a partir de $s \in G$,
  - 2. G forma uma árvore de raiz s,
  - 3. para todos os vértices $v \in V$, o caminho simples de s até v é um caminho mais curto de s até v em G.

Algoritmo de Dijkstra

- **Resolve o problema do caminho mais curto de origem única em:**
  - Um grafo ponderado G(V, E);
  - Sendo que todos os pesos das arestas NÃO SÃO NEGATIVOS;
  - O grafo pode conter ciclos.

- **É o algoritmo de caminho mais curto mais aplicado ao problema rodoviário porque aceita ciclos e não possui arestas de peso negativo.**

Algoritmo de Dijkstra

- **Mantém um conjunto S de vértices cujos caminhos mais curtos até um vértice origem já são conhecidos.**

- **Produz uma árvore de caminhos mais curtos de um vértice origem s para todos os vértices que são alcançáveis a partir de s.**

Relaxamento

- **Utiliza a técnica de relaxamento:**
  - Para cada vértice v ∈ V o atributo p[v] é um limite superior do peso de um caminho mais curto do vértice origem s até v.
  - O vetor p[v] contém uma estimativa de um caminho mais curto.

- **Quando chamamos o método para relaxar (RELAX(u,v)) estamos determinando se podemos melhorar o caminho mínimo atual de s a v tomando (u,v) como a última aresta.**

Algoritmo

- **O primeiro passo do algoritmo é inicializar os antecessores e as estimativas de caminhos mais curtos:**
  - antecessor[v] = null para todo vértice $v \in V$,
  - $p[u] = 0$, para o vértice origem s, e
  - $p[v] = \infty$ para $v \in V - \{s\}$.
    - P[] é a distância. Distância da origem u até v

Algoritmo

1. Iguale *shortest*[v] a ∞ para cada vértice v exceto s, iguale *shortest*[s] a 0 e iguale *pred*[v] a NULL para cada vértice v.
2. Ajuste Q para conter todos os vértices.
3. Enquanto Q não estiver vazio, faça o seguinte:
   - **a.** Ache o vértice u no conjunto Q que tem o valor *shortest* mais baixo e remova-o de Q.
   - **b.** Para cada vértice v adjacente a u:
     - i. Chame RELAX(u,v).

Algoritmo

- **Inicializa**

INITIALIZE-SINGLE-SOURCE(G, s)
1 **for** cada vértice $v \in V[G]$
2 **do** $d[v] \leftarrow \infty$
3 $\pi[v] \leftarrow$ NIL
4 $d[s] \leftarrow 0$

```
INICIALIZA (G(V,E), s)
    for cada v ∈ V
        v.dist = ∞
        v.pai = NULL
    end-for
    s.dist = 0
```

d[v] = distancia estimada
Pi[v] = antecessor

Relaxamento de Aresta

- **O relaxamento de uma aresta (u, v) consiste em verificar se é possível melhorar o melhor caminho até v obtido até o momento se passarmos por u.**

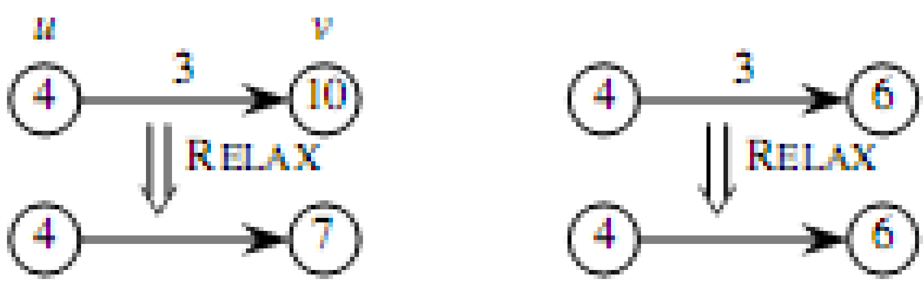
- **Comparamos o peso do caminho mínimo atual até v.**

- **Se isto acontecer, p[v] e antecessor[v] devem ser atualizados.**

Relaxamento de Arestas

```
RELAX(u, v, w)
if d[v] > d[u] + w(u, v)
   then d[v] ← d[u] + w(u, v)
        π[v] ← u
```

W(u,v) = peso da aresta



Algoritmo

1. Iguale *shortest*[*v*] a ∞ para cada vértice *v* exceto *s*, iguale *shortest*[*s*] a 0 e iguale *pred*[*v*] a NULL para cada vértice *v*.
2. Faça (*Q*, *v*) uma fila de prioridade vazia.
3. Para cada vértice *v*:
   a. Chame INSERT (*Q*, *v*).
4. Enquanto *Q* não estiver vazio, faça o seguinte:
   a. Chame EXTRACT-MIN (*Q*) e determine que *u* contenha o vértice retornado.
   b. Para cada vértice *v* adjacente a *u*:
      i. Chame RELAX(*u*, *v*).
      ii. Se a chamada a RELAX(*u*, *v*) diminuir o valor de *shortest*[*v*], chame DECREASE-KEY(*Q*, *v*).

Algoritmo

$\text{DIJKSTRA}(V, E, w, s)$

$\text{INIT-SINGLE-SOURCE}(V, s)$

$S \leftarrow \emptyset$

$Q \leftarrow V$ $\quad \triangleright$ i.e., insert all vertices into Q

while $Q \neq \emptyset$

do $u \leftarrow \text{EXTRACT-MIN}(Q)$

$S \leftarrow S \cup \{u\}$

for each vertex $v \in Adj[u]$

do $\text{RELAX}(u, v, w)$

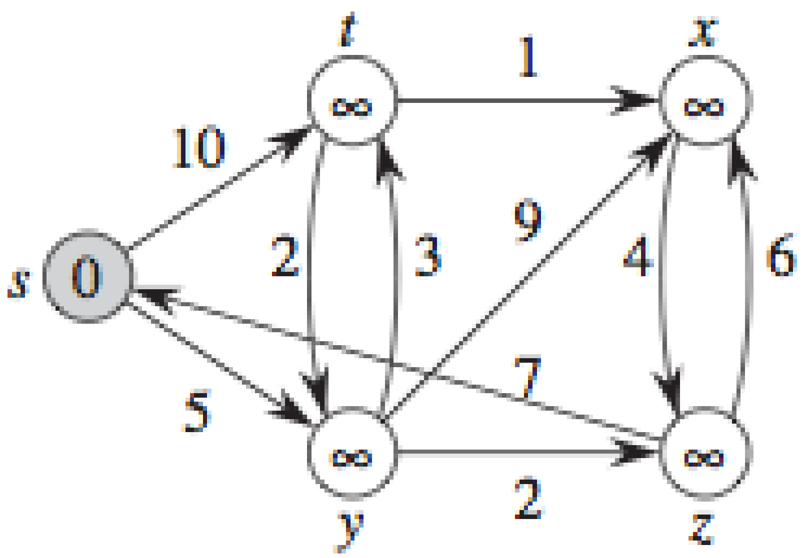
```
DIJKSTRA(G(V,E), w, s)
  INICIALIZA(G, s)
  S ← {}
  Q ← V

  Enquanto |Q| ≠ 0
    u ← extrairMinimo(Q)
    S ← S ∪ {u}
    para cada v ∈ Adj[u]
      RELAXA(u, v, w)
    fim para
  fim enquanto
fim
```

Q = fila de prioridade
S = A lista S contém os vértices cujos pesos finais de caminhos mais curtos desde a origem s já foram determinados.

Dijkstra Exemplo

DIJKSTRA(G(V,E), w, s)
  INICIALIZA(G, s)
  S ← {}
  Q ← V

  Enquanto |Q| ≠ 0
    u ← extrairMinimo(Q)
    S ← S ∪ {u}
    para cada v ∈ Adj[u]
      RELAXA(u, v, w)
    fim para
  fim enquanto
fim

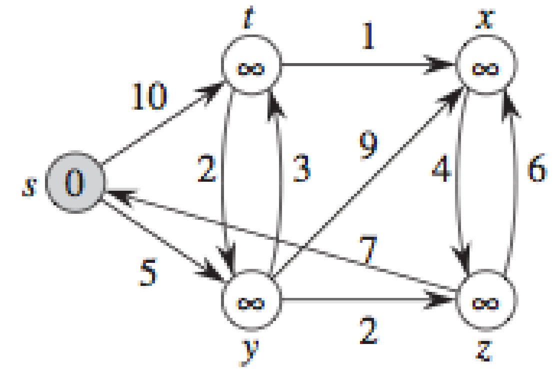


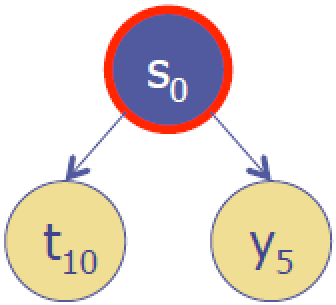
$Q=\{s_0, t_\infty, x_\infty, y_\infty, z_\infty\}$
$S=\{\}$

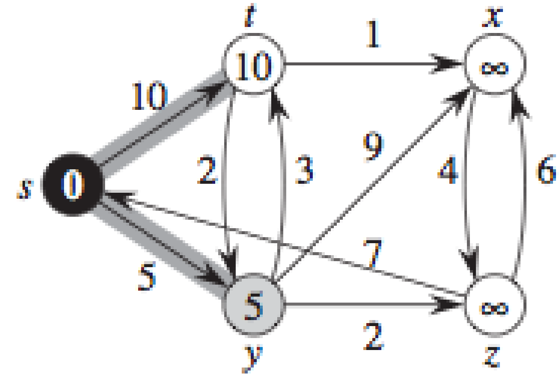
FONTE: http://professor.unisinos.br/pjaques/material/est2/13-grafos-caminhos-minimos.pdf

Dijkstra Exemplo

```
DIJKSTRA(G(V,E), w, s)
  INICIALIZA(G, s)
  S ← {}
  Q ← V

  Enquanto |Q| ≠ 0
    u ← extrarMinimo(Q)
    S ← S ∪ {u}
    para cada v ∈ Adj[u]
      RELAXA(u, v, w)
    fim para
  fim enquanto
fim
```







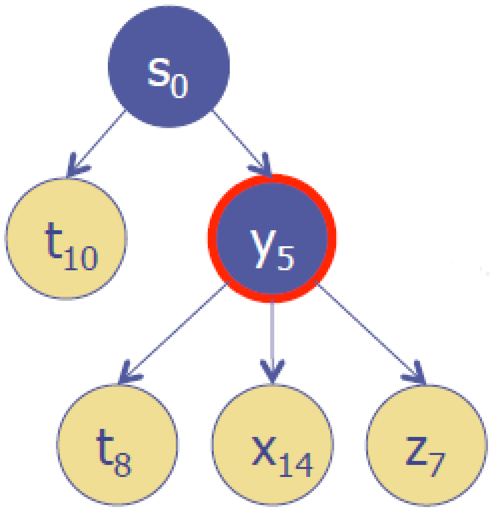
$Q=\{y_5, t_{10}, x_\infty, z_\infty\}$
$S=\{s_0\}$

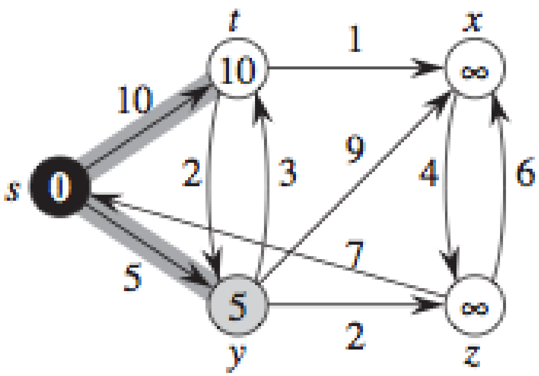
FONTE: http://professor.unisinos.br/pjaques/material/est2/13-grafos-caminhos-minimos.pdf

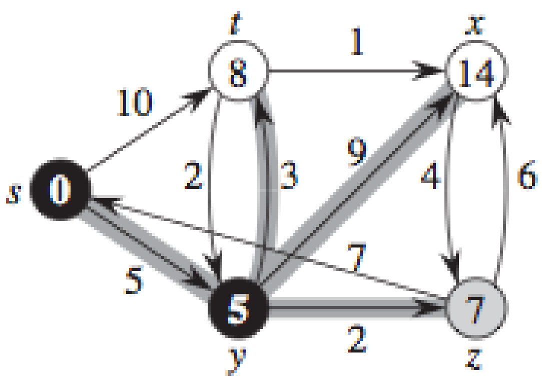
Dijkstra Exemplo

DIJKSTRA(G(V,E), w, s)
  INICIALIZA(G, s)
  S ← {}
  Q ← V

  Enquanto |Q| ≠ 0
    u ← extrarMinimo(Q)
    S ← S ∪ {u}
    para cada v ∈ Adj[u]
      RELAXA(u, v, w)
    fim para
  fim enquanto
fim







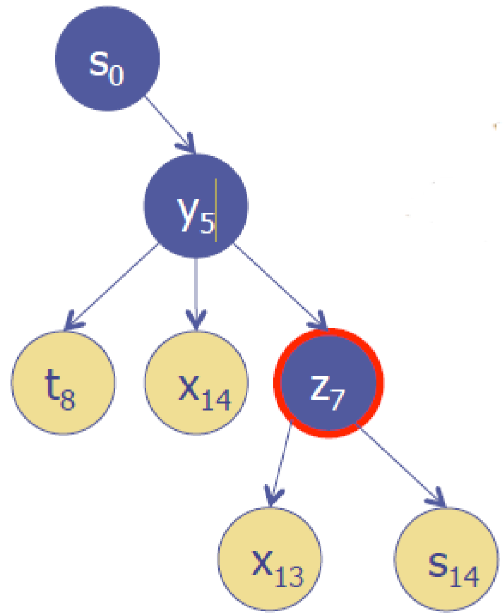
$Q=\{z_7, t_8, x_{14}\}$
$S=\{s_0, y_5\}$

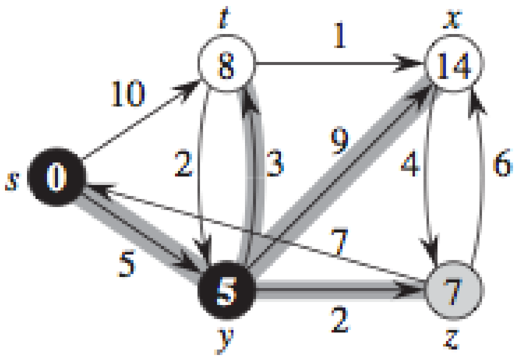
FONTE: http://professor.unisinos.br/pjaques/material/est2/13-grafos-caminhos-minimos.pdf

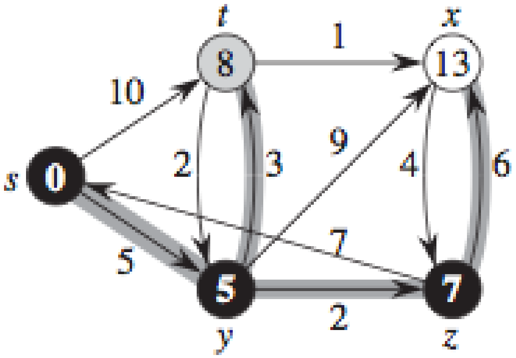
Dijkstra Exemplo

DIJKSTRA(G(V,E), w, s)
  INICIALIZA(G, s)
  S ← {}
  Q ← V

  Enquanto |Q| ≠ 0
    u ← extrarMinimo(Q)
    S ← S ∪ {u}
    para cada v ∈ Adj[u]
      RELAXA(u, v, w)
    fim para
  fim enquanto
fim



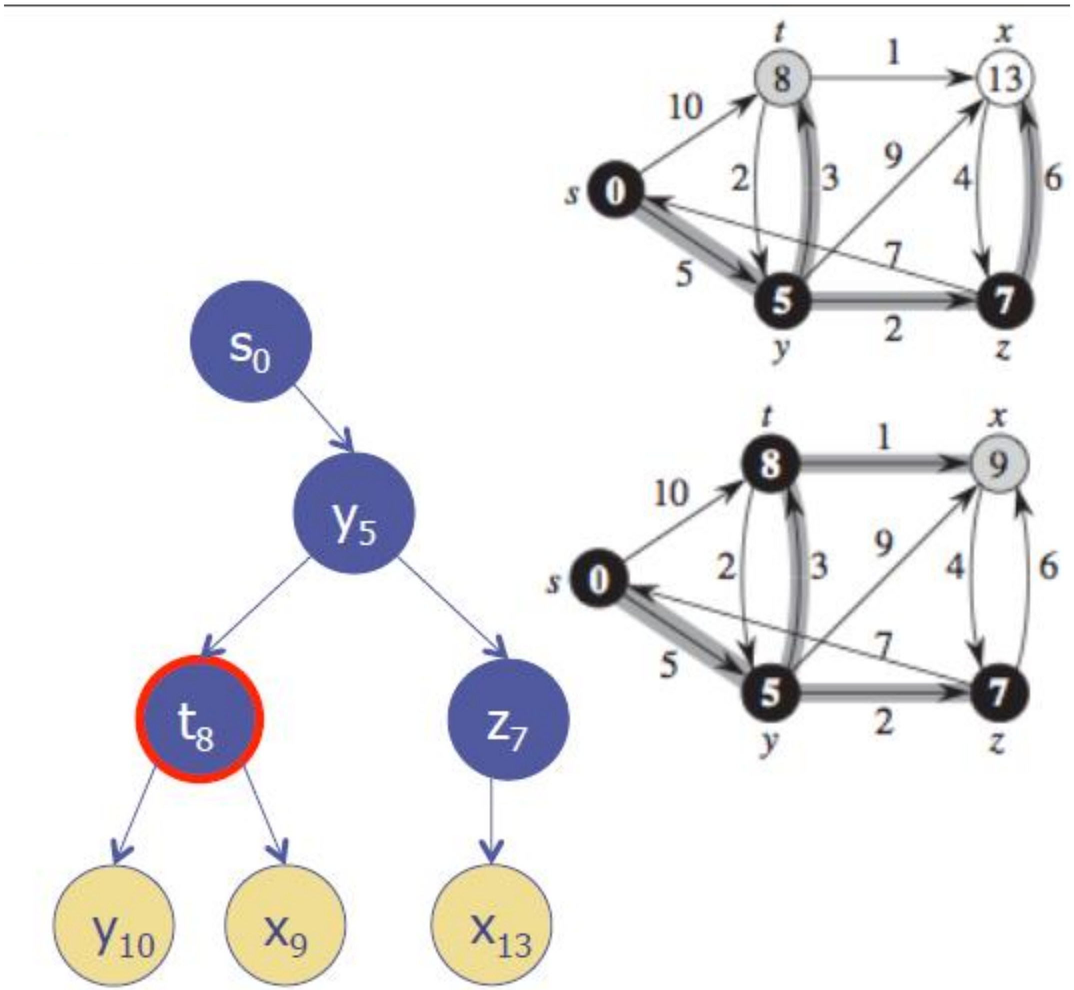




$Q=\{t_8, x_{13}\}$
$S=\{s_0, y_5, z_7\}$

Dijkstra Exemplo

```
DIJKSTRA(G(V,E), w, s)
  INICIALIZA(G, s)
  S ← {}
  Q ← V

  Enquanto |Q| ≠ 0
    u ← extrarMinimo(Q)
    S ← S ∪ {u}
    para cada v ∈ Adj[u]
      RELAXA(u, v, w)
    fim para
  fim enquanto
fim
```



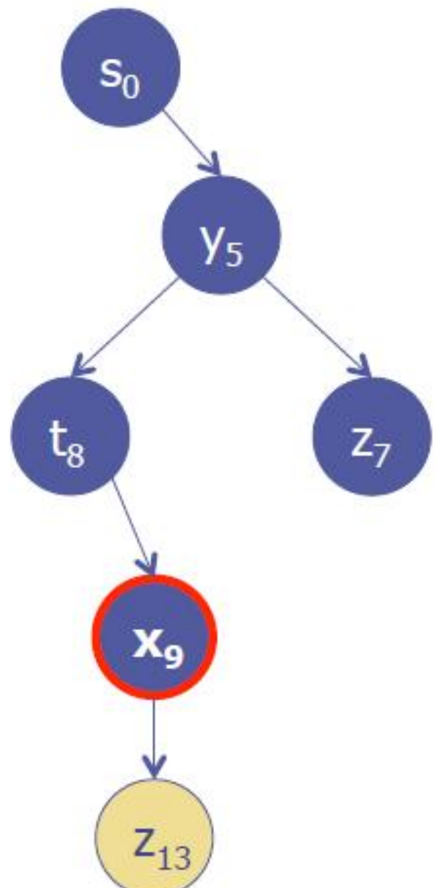
$Q=\{x_9\}$
$S=\{s_0, y_5, z_7, t_8\}$

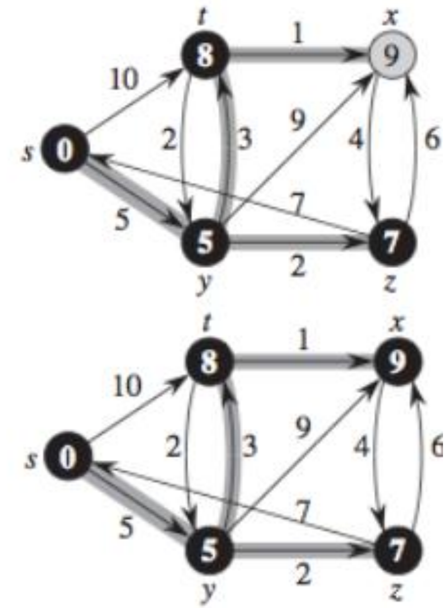
FONTE: http://professor.unisinos.br/pjaques/material/est2/13-grafos-caminhos-minimos.pdf

Dijkstra Exemplo

DIJKSTRA(G(V,E), w, s)
  INICIALIZA(G, s)
  S ← {}
  Q ← V

  Enquanto |Q| ≠ 0
    u ← extrarMinimo(Q)
    S ← S ∪ {u}
    para cada v ∈ Adj[u]
      RELAXA(u, v, w)
    fim para
  fim enquanto
fim

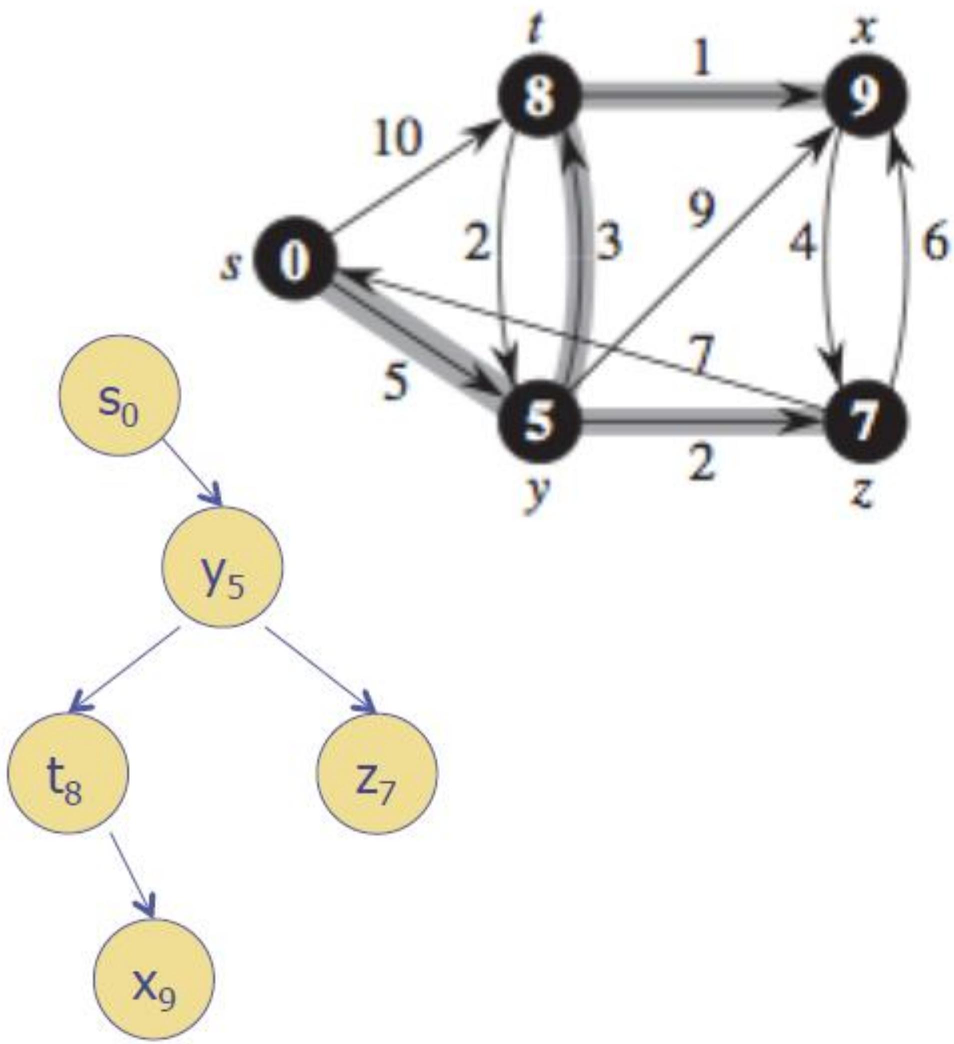




$Q=\{\}$
$S=\{s_0, y_5, z_7, t_8, x_9\}$

FONTE: http://professor.unisinos.br/pjaques/material/est2/13-grafos-caminhos-minimos.pdf

Dijkstra Exemplo



FONTE: http://professor.unisinos.br/pjaques/material/est2/13-grafos-caminhos-minimos.pdf

Algoritmo Dijskstra - complexidade

- **Vetor com Fila de prioridades**
  - Neste caso, extract-min leva tempo $O(n)$ e há n operações extract-min, com $n = |V|$ e $m = |E|$
  - Uma vez que cada aresta é examinada no máximo uma vez, a operação relax é executada no máximo uma vez, levando tempo $O(1)$
  - Podemos concluir que o algoritmo leva tempo: $O(V^2+E) = O(V^2)$

Algoritmo Dijskstra - complexidade

- **Heap binário com Fila de prioridades**
  - Quando o grafo é esparso, entretanto, é mais prático utilizarmos uma fila de prioridades implementada com heap binário
  - Neste caso, extract-min leva tempo O(lgV) e há n operações extract-min, com n = |V| e m = |E|
  - Cada aresta (u, v) é examinada uma vez, forçando uma operação relax(u,v,w) que custa no máximo O(lgV), quando a distância d[v] for reduzida
  - Portanto o algoritmo executa em tempo O(VlgV + ElgV) = O(ElgV) assumindo que E > V

Algoritmo de Bellman-Ford

- **O algoritmo de Bellman-Ford resolve o problema de caminhos mínimos com uma fonte no caso geral, em que as arestas podem possuir peso negativo**
- **O algoritmo retorna um valor lógico indicando se foi ou não encontrado um ciclo de comprimento negativo alcançável a partir de s**
- **Se não há ciclo negativo alcançável a partir de s, o algoritmo produz a árvore de caminhos mínimos com raiz em s e os seus respectivos comprimentos (pesos)**

Algoritmo de Bellman-Ford

Procedimento BELLMAN-FORD (G, s)

Entradas:

- *G*: um grafo dirigido contendo um conjunto *V* de *n* vértices e um conjunto *E* de *m* arestas dirigidas com pesos arbitrários.
- *s*: um vértice-fonte em *V*.

Resultado: O mesmo de DIJKSTRA

1. Iguale *shortes*[*v*] a ∞ para cada vértice *v* exceto *s*, iguale *shortest*[*s*] a 0 e *pred*[*v*] a NULL para cada vértice *s*.
2. Para $i = 1$ a $n - 1$:
   a. Para cada aresta (*u*, *v*) em E:
      i. Chame RELAX(*u*, *v*).

Algoritmo de Bellman-Ford

```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i ← 1 to |V[G]| − 1
     do for each edge (u, v) ∈ E[G]
            do RELAX(u, v, w)
for each edge (u, v) ∈ E[G]
     do if d[v] > d[u] + w(u, v)
          then return FALSE
return TRUE
```

Algoritmo de Bellman-Ford

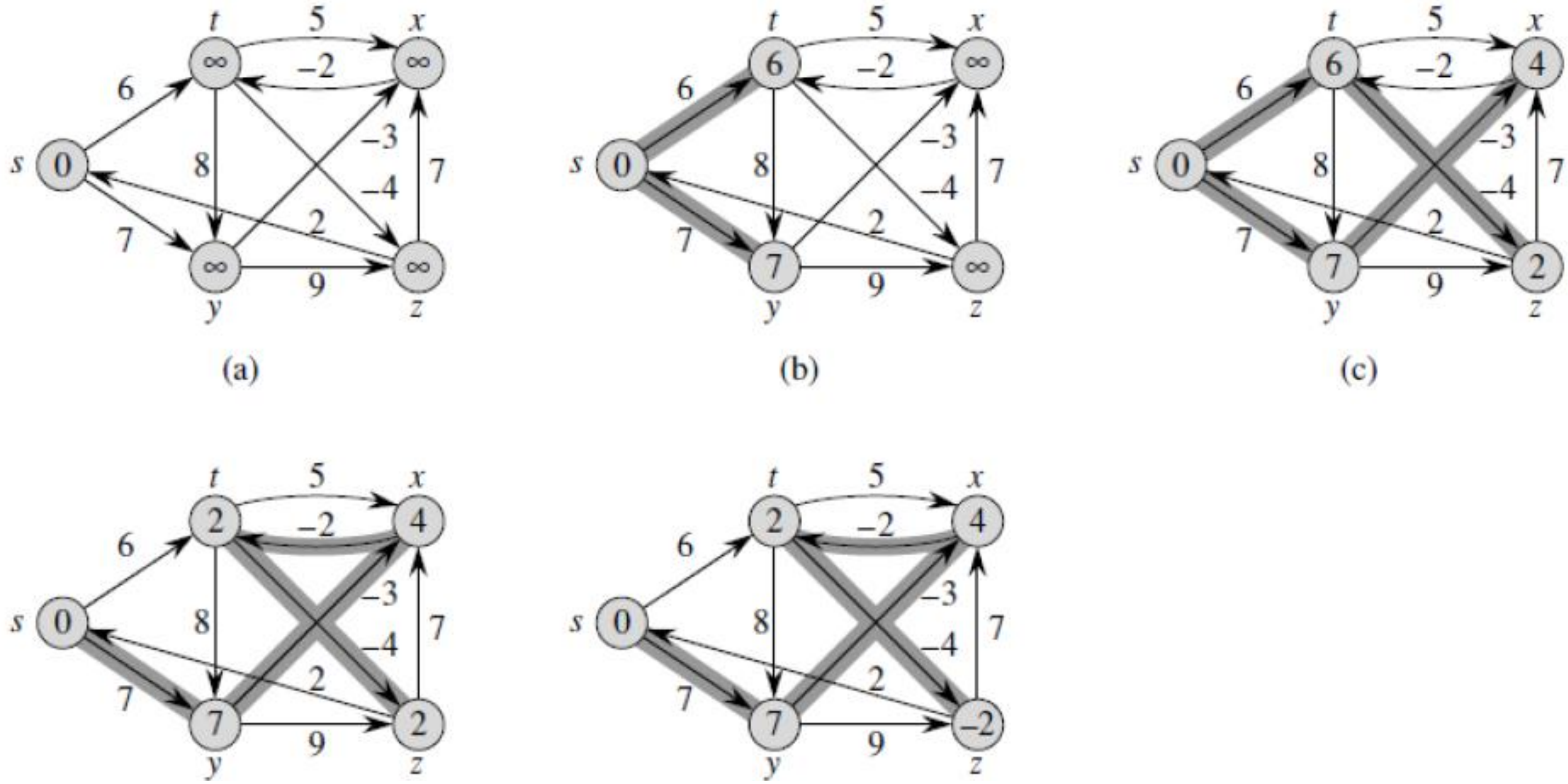
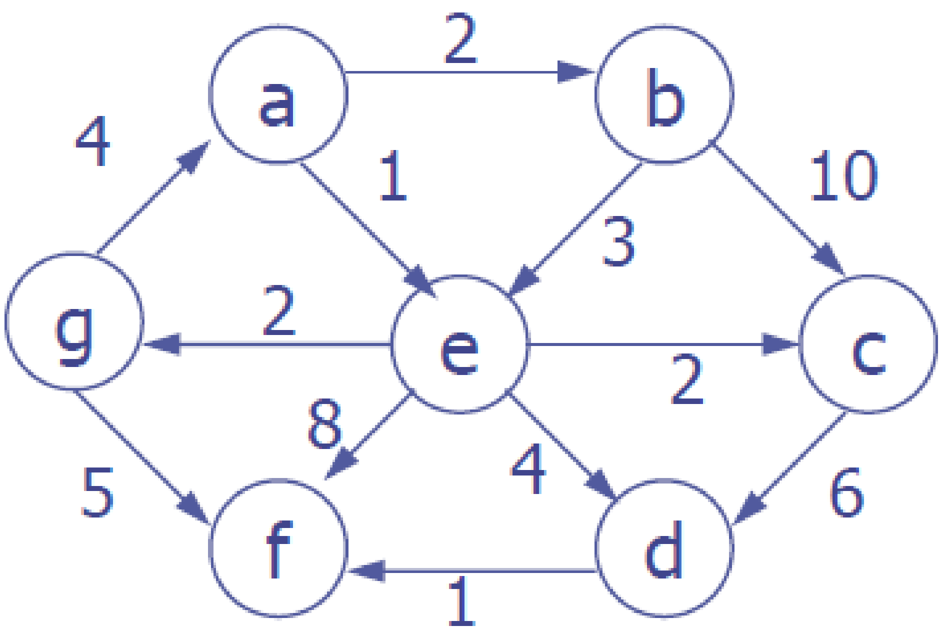


FIGURA 24.4 A execução do algoritmo de Bellman-Ford. A origem é o vértice s. Os valores de d são mostrados dentro dos vértices, e as arestas sombreadas indicam os valores de predecessores; se a aresta (u, v) estiver sombreada, então $\pi[v] = u$. Nesse exemplo específico, cada passagem relaxa as arestas na ordem (t,x), (t,y), (t,z), (x,t), (y,x), (y,z), (z,x), (z,s), (s,t), (s,y). **(a)** A situação imediatamente antes da primeira passagem sobre as arestas. **(b)–(e)** A situação após cada passagem sucessiva sobre as arestas. Os valores de d e π na parte (e) são os valores finais. O algoritmo de Bellman-Ford retorna TRUE nesse exemplo

Exercício

1) Encontre o menor caminho para o grafo abaixo utilizando o algoritmo de Dijkstra e começando pelo vértice a.



Referencias

- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. Algoritmos: Teoria e Prática. Editora Campus, 2002
- Ziviani, N. Projeto de Algoritmos Com Implementações em Pascal e C, Cengage Learning, 2004.
- Notas de aula. Prof. Rafael Fernandes DAI/IFMA
- Notas de aula. Profa. Leticia Bueno UFABC
- Notas de aula. Profa. Patrícia A. Jaques. UNISINOS
- **http://www.facom.ufu.br/~madriana/EBD/Didatica.pdf**
- **http://www.ic.unicamp.br/~zanoni/mo417/2011/aulas/handout/11-arvore-geradora-minima.pdf**