Projeto e Análise de Algoritmos

Prof. Hamilton José Brumatto

Bacharelado em Ciência da Computação - UESC

21 de maio de 2019

Caminhos Mínimos

O Problema

- Seja G um grafo orientado e suponha que para cada aresta (u, v) associamos um peso (custo, distância) w(u,v), denotado por (G, w).
  - **Problema do caminho mínimo entre dois vértices**: Dados dois vértices s e t em (G, w), encontrar um caminho (de peso) mínimo de s a t.
  - Este problema não é mais difícil que:
    Problema dos caminhos mínimos de mesma origem:
    Dados (G, w) e s, encontrar para cada vértice v de G, um caminho mínimo de s a v.

Subestrutura ótima de caminhos mínimos

Teorema

Seja (G, w) um grafo orientado e seja

$$P = (v_1, v_2, \ldots, v_k)$$

um caminho mínimo de v_1 a v_k.
Então para quaisquer i, j com $1 \leqslant i \leqslant j \leqslant k$

$$P_{ij} = (v_i, v_{i+1}, \ldots, v_j)$$

é um caminho mínimo de v_i a v_j

Subestrutura ótima de caminhos mínimos

Demonstração.

Suponha, por contradição, que existe $P'_{ij} < P_{ij}$, um caminho de v_i a v_j menor. Então, o caminho P' formado por: Prefixo de P de v_1 a v_i, P'_{ij} e o Posfixo de P de v_j a v_k seria um caminho menor que P de v_1 a v_k. □

Construindo Caminhos Mínimos

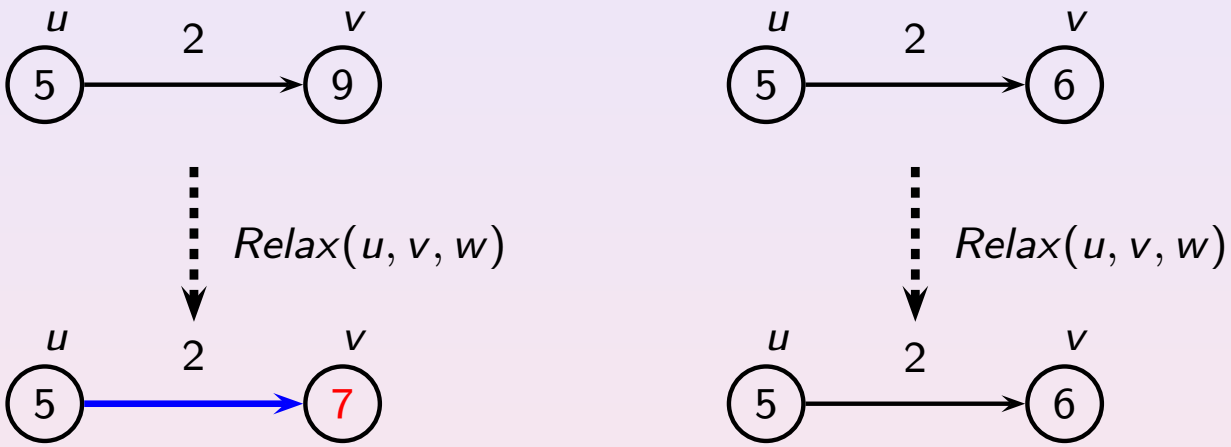
- A construção do algoritmo é baseada no algoritmo BFS (Busca em Largura).
- Cada vértice $v \in V(G)$ é associado a um predecessor $\pi[v]$.
- No final do algoritmo será construída uma Árvore de Caminhos Mínimos com raiz em s.
- Um caminho de s a v na Árvore é um caminho mínimo de s a v em (G, w)
- Para cada $v \in V(G)$ determinamos $dist(s, v)$, o peso de um caminho mínimo de s a v em (G, w) que representa a distância de s a v.
- Cada $v \in V(G)$ terá associado $d[v]$ que é uma estimativa de $dist(s, v)$.

Inicição de Parâmetros

Algoritmo INICIAORIGEMUNICA(G,s)
  para $v \in V[G]$ **faça**
    $d[v] \leftarrow \infty$
    $\pi[v] \leftarrow NULO$
  $d[s] \leftarrow 0$

- $d[v]$ é uma estimativa superior para o peso de um caminho mínimo de s a v. Ou seja, $d[v] \geqslant dist(s, v)$.
- $d[v]$ tem a informação da menor distância que o algoritmo encontrou para um caminho de s a v.
- O caminho encontrado pode ser recuperado por meio dos predecessores marcados em $\pi[v]$.

Processo de Relaxamento: melhorando a estimativa de $d[v]$ usando (u, v)



Algoritmo RELAXA(u,v,w)
se $d[v] > d[u] + w(u, v)$ **então**
$d[v] \leftarrow d[u] + w(u, v)$
$\pi[v] \leftarrow u$

Algoritmos de Caminhos Mínimos de Única Origem

Serão vistos três algoritmos que utiliza o relaxamento na busca de caminhos mínimos, cada algoritmo depende da natureza do grafo.

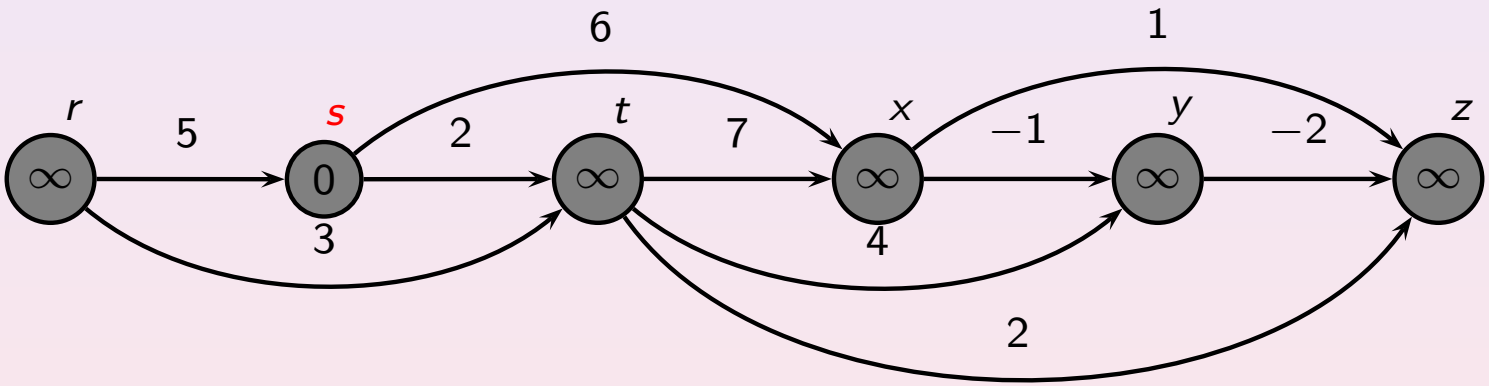
- G é acíclico: Algoritmo de Ordenação Topológica.
- (G, w) não tem arcos de peso negativo: Algoritmo de Dijkstra
- (G, w) tem arcos de peso negativo, mas não contém ciclos negativos: Algoritmo de Bellman-Ford

Caminhos mínimos em grafos acíclicos

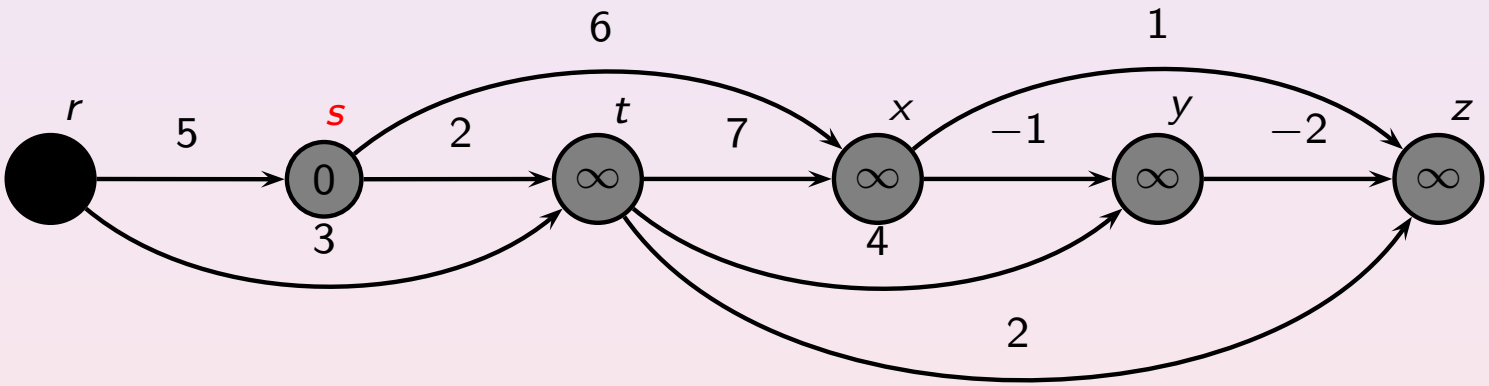
Entrada: $G = (V, E)$ orientado e acíclico, com função peso w nos arcos, e uma origem s.
Saída: Vetor $d[v] = dist(s, v)$ para $v \in V$ e uma Árvore de Caminhos Mínimos definida por $\pi[]$.

Algoritmo CaminhosMinimosGAciclico(G,w,s)
    OrdenacaoTopologica(G)
    IniciaOrigemUnica(G, s)
    para u na ordem topológica **faça**
        para $v \in Adj[u]$ **faça**
            Relaxa(u, v, w)

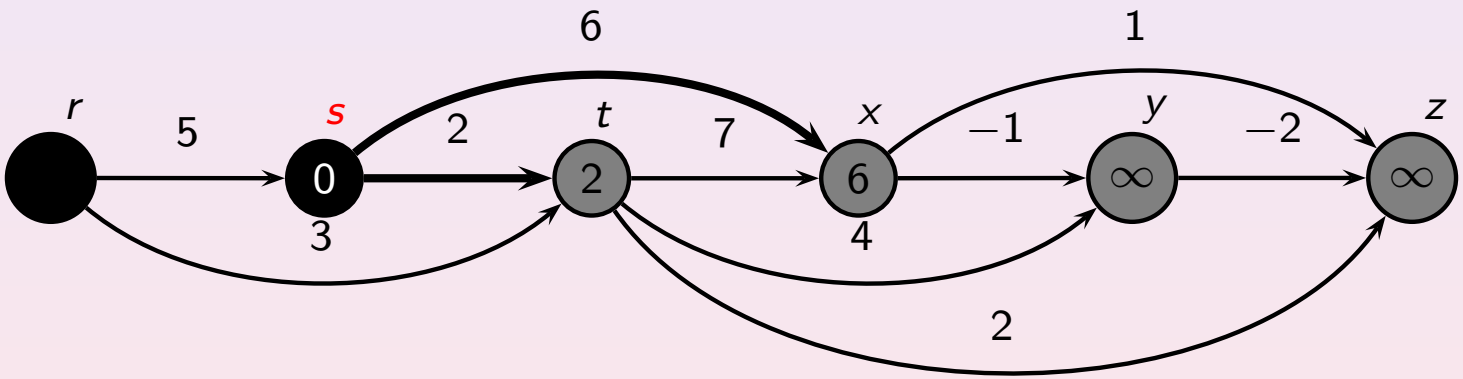
Exemplo



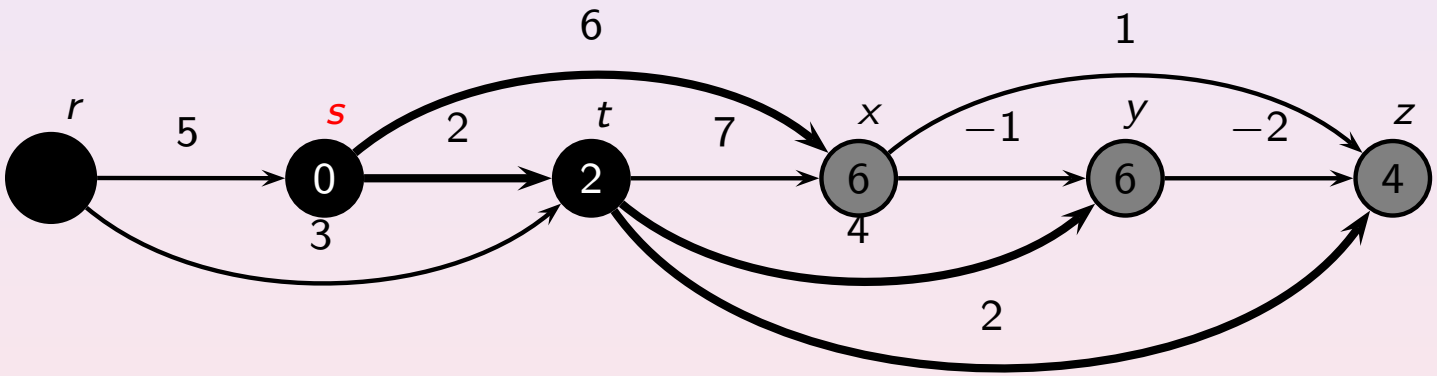
Exemplo



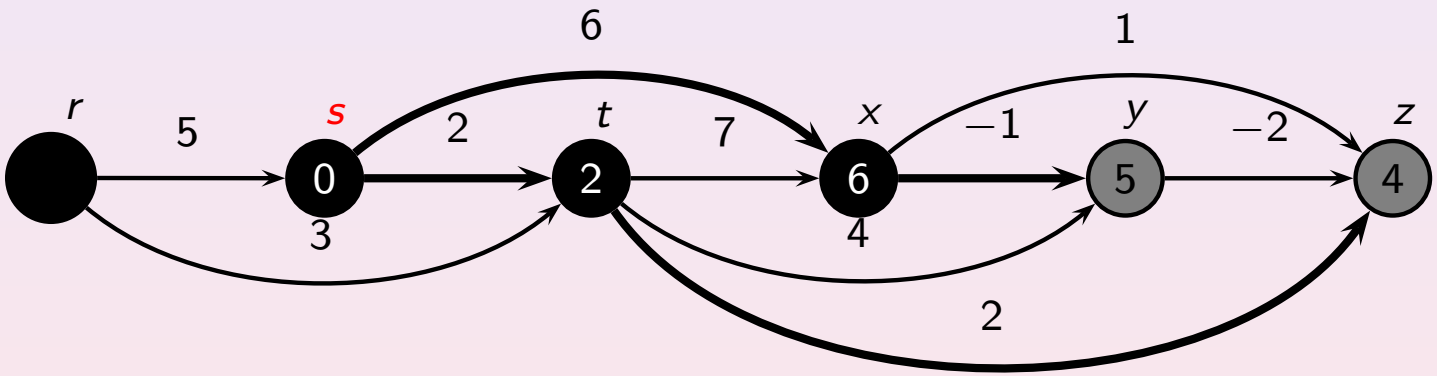
Exemplo



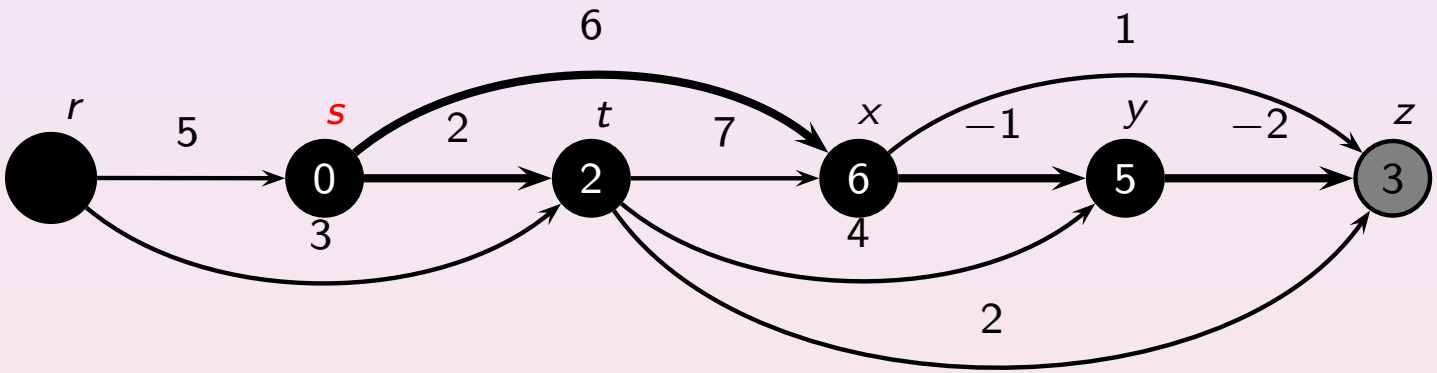
Exemplo



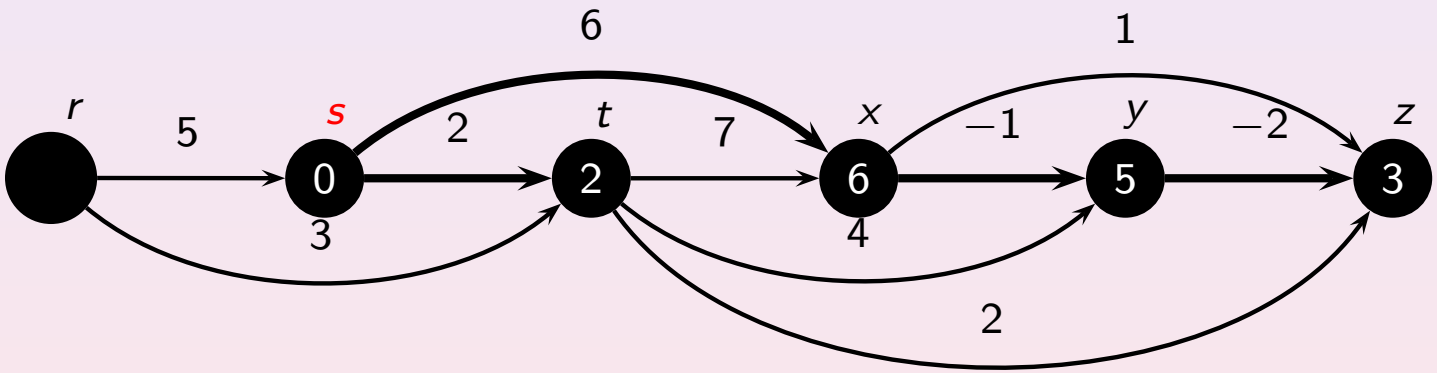
Exemplo



Exemplo



Exemplo



Complexidade

```
1: Algoritmo CaminhosMinimosGAciclico(G,w,s)
2:     OrdenacaoTopologica(G)
3:     IniciaOrigemUnica(G, s)
4:     para u na ordem topológica faça
5:         para v ∈ Adj[u] faça
6:             Relaxa(u, v, w)
```

Linha(s)	Tempo total
2	$O(V+E)$
3	$O(V)$
4 - 6	$O(V+E)$

Corretude

- A análise da corretude servirá para este e outros algoritmos que satisfazem as seguintes restrições:
  1. O algoritmo é iniciado com $IniciaOrigemUnica(G, s)$
  2. Um valor de $d[v]$ e $\pi[v]$ só pode ser modificado (reduzido) através de uma chamada de $Relaxa(u, v, w)$ para alguma aresta (u, v).

Corretude: Ao longo do algoritmo a propriedade sempre vale

Lema: Desigualdade de triângulos

Seja $G = (V, E)$ um grafo orientado ponderado com função peso $w : E \to R$ e vértice de origem s. Então, para todos os arcos $(u, v) \in E$, temos:

$$dist(s, v) \leqslant dist(s, u) + w(u, v)$$

Demonstração.

Vamos considerar $p = (s, \ldots, v)$ o caminho mínimo de s a v. Neste caso, $w(p) = dist(s, v)$ e não é maior que qualquer outro caminho, inclusive o caminho p' formado pelo caminho mínimo $p_{su} = (s, \ldots, u)$ e o arco (u, v). $w(p') = dist(s, u) + w(u, v)$. Se não existe caminho de s a v, então não existe caminho de s a u devido ao arco (u, v). Assim, $dist(s, v) = dist(s, u) = \infty$. □

Corretude: Ao longo do algoritmo a propriedade sempre vale

Lema: Propriedade do Limite Superior

$d[v] \geqslant dist(s, v)$ para $v \in V$ e quando $d[v]$ fica igual a $dist(s, v)$, seu valor nunca mais muda.

Demonstração.

Indução na iteração de relaxamento.

Base: $d[s] = 0 \geqslant dist(s, s)$ e $d[v] = \infty \geqslant dist(s, v)$ p/$v \neq s$.

Hipótese da Indução: Antes do relaxamento, $d[v] \geqslant dist(s, v)$.

Passo: Ao relaxar a aresta (u, v) Somente $d[v]$ pode mudar.

$$\begin{aligned} d[v] &= d[u] + w(u, v) \\ &\geqslant dist(s, u) + w(u, v) && \text{(por hipótese da indução)} \\ &\geqslant dist(s, v) && \text{(pela desigualdade de triângulos)} \end{aligned}$$

Se $d[v] = dist(s, v)$ o valor não diminui pois $d[v] \geqslant dist(s, v)$ e o relaxamento nunca aumenta o valor. □

Corretude: Ao longo do algoritmo a propriedade sempre vale

Lema: Propriedade de nenhum caminho

Se não existe caminho de s a v, então temos sempre que $d[v] = dist(s, v) = \infty$

Demonstração.

No início: $d[v] = \infty$ pelo $IniciaOrigemUnica(G, s)$. Como $dist(s, v) = \infty$, pelo lema da propriedade de limite superior, como $d[v] = dist(s, v)$ este valor nunca muda. □

Corretude: Ao longo do algoritmo a propriedade sempre vale

Lema: Propriedade do relaxamento

Imediatamente depois de relaxar a aresta (u, v), pela execução de $Relaxa(u, v, w)$, temos que $d[v] \leqslant d[u] + w(u, v)$.

Demonstração.

Se, imediatamente antes de relaxar a aresta (u, v), temos $d[v] > d[u] + w(u, v)$, então no relaxamento $d[v] = d[u] + w(u, v)$. Se, imediatamente antes de relaxar a aresta (u, v), temos $d[v] \leqslant d[u] + w(u, v)$, então no relaxamento o valor de $d[v]$ não muda. Assim imediatamente após o relaxamento da aresta (u, v) vale que $d[v] \leqslant d[u] + w(u, v)$ □

Corretude: Ao longo do algoritmo a propriedade sempre vale

Lema: Propriedade da convergência

Seja P um caminho mínimo de s a v, cuja última aresta é (u, v) e suponha que $d[u] = dist(s, u)$ antes da chamada $Relaxa(u, v, w)$. Então após a chamada, $d[v] = dist(s, v)$.

Demonstração.

Pela propriedade do limite superior, se $d[u] = dist(s, u)$ em algum ponto antes de relaxar a aresta (u, v), então essa igualdade se mantém válida daí em diante. Em particular, após o relaxamento da aresta (u, v), temos:

$$
\begin{aligned}
d[v] &\geqslant d[u] + w(u, v) \\
&= dist(s, u) + w(u, v) && \text{(Propriedade do relaxamento)} \\
&= dist(s, v) && \text{(Teorema de subestrutura ótima)}
\end{aligned}
$$

Corretude: Ao longo do algoritmo a propriedade sempre vale

Lema: Propriedade do relaxamento de caminho

Seja $P = (v_0 = s, v_1, \ldots, v_k)$ um caminho mínimo de v_1 a v_k e suponha que os arcos (v_0, v_1), (v_1, v_2),..., (v_{k-1}, v_k) são relaxadas nesta ordem. Então $d[v_k] = dist(s, v_k)$.

Demonstração.

Indução no relaxamento das arestas:

Base: $d[s] = 0 = dist(s, s)$. Pela propriedade do limite superior, este valor nunca muda.

Hipótese de Indução: Para o vértice v_{i-1}, antes de relaxar a aresta (v_{i-1}, v_i) vale que $d[v_{i-1}] = dist(s, v_{i-1})$.

Passo: Ao relaxar a aresta (v_{i-1}, v_i), pela propriedade da convergência, $d[v_i] = dist(s, v_i)$ e pela propriedade do limite superior, este valor nunca muda. □

Corretude: Algoritmo de Caminhos Mínimos em Grafos Acíclicos

Teorema

O algoritmo *CaminhosMinimosGAciclico*(G, w, s) computa corretamente.

Demonstração.

Como é realizada uma ordenação topológica, todos os arcos possuem cauda na esquerda e cabeça na direita. Qualquer caminho mínimo terá origem na esquerda e destino na direita. Como o relaxamento ocorre a partir dos vértices mais à esquerda, em qualquer caminho $P = (v_0 = s, v_1, \ldots, v_k)$ as arestas de um caminho são relaxadas na ordem (v_0, v_1),(v_1, v_2),..., (v_{k-1}, v_k). Pela propriedade do relaxamento de caminho, o algoritmo calcula corretamente $d[v] = dist(s, v)$ para todo $v \in V$. □

O algoritmo de Dijkstra

- Entrada: Um grafo orientado (G, w) (sem arcos de peso negativo) e um vértice s de G
- Saída:
  1. para cada $v \in V(G)$, o peso de um caminho mínimo de s a v.
  2. uma Árvore de Caminhos Mínimos com raiz em s.
     Um caminho de s a v nesta árvore é um caminho mínimo de s a v em (G, w).

O algoritmo

Algoritmo DIJKSTRA(G,w,s)
    $IniciaOrigemUnica(G, s)$
    $S \leftarrow \emptyset$
    $Q \leftarrow V[G]$
    enquanto $Q \neq \emptyset$ **faça**
        $u \leftarrow ExtrairMinimo(Q)$
        $S \leftarrow S \cup \{u\}$
        para $v \in Adj[u]$ **faça**
            $Relaxa(u, v, w)$

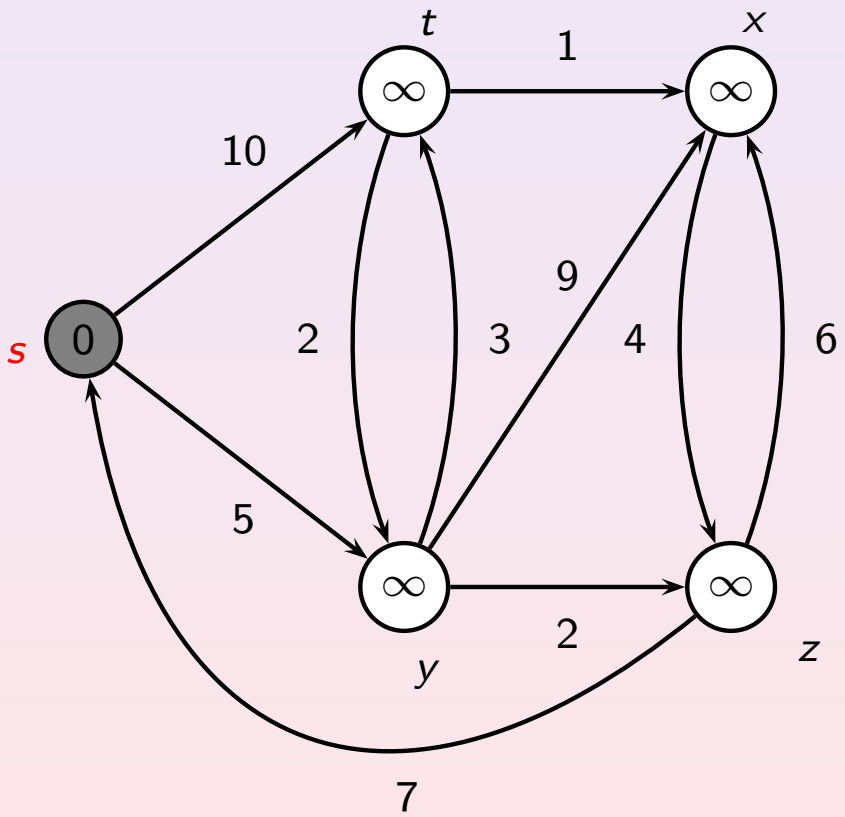
- Q é uma fila de prioridades
- S não é necessário mas facilita na análise.

Entendendo o algoritmo

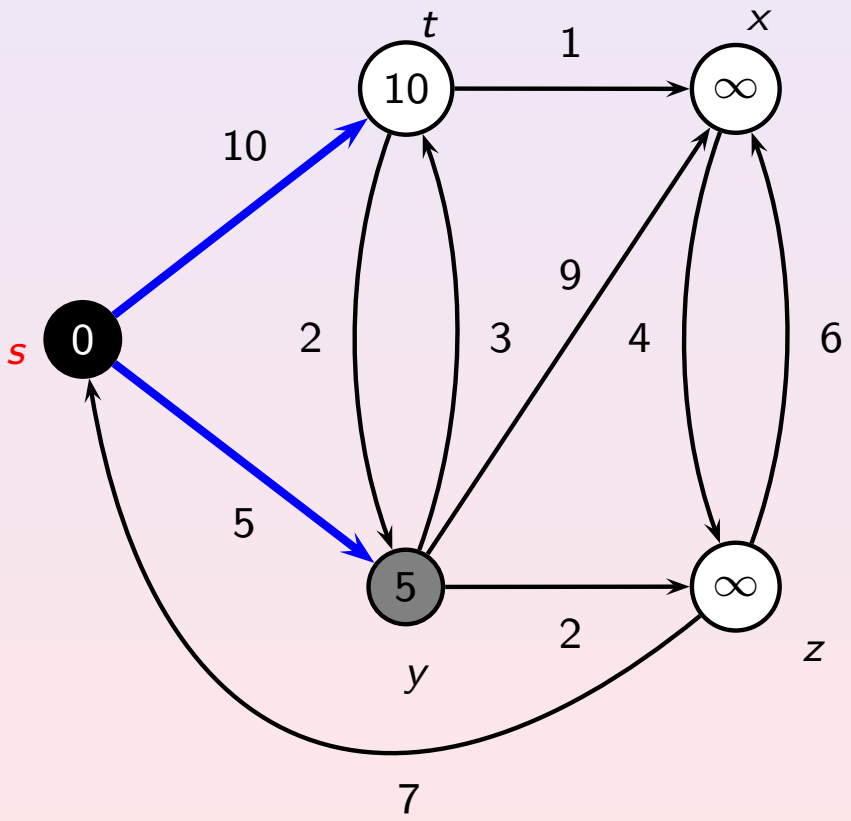
Em cada iteração, o algoritmo Dijkstra

- Escolhe um vértice u fora do conjunto S que esteja mais próximo a esse e acrescenta-o a S,
- Atualiza as distâncias estimadas dos vizinhos de u e
- Atualiza a Árvore dos Caminhos Mínimos.

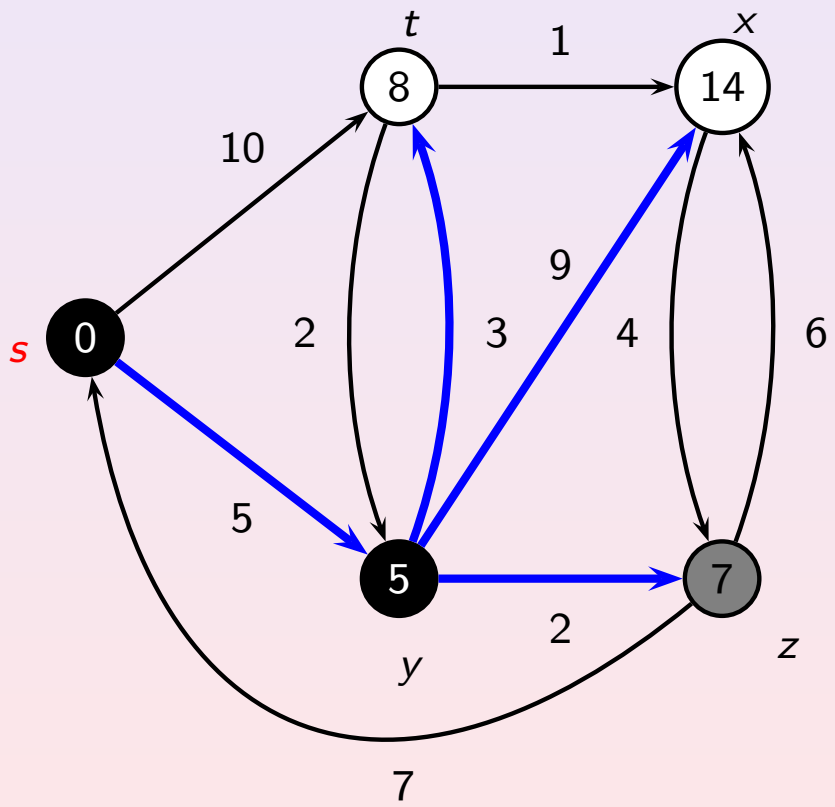
Exemplo do algoritmo Dijkstra



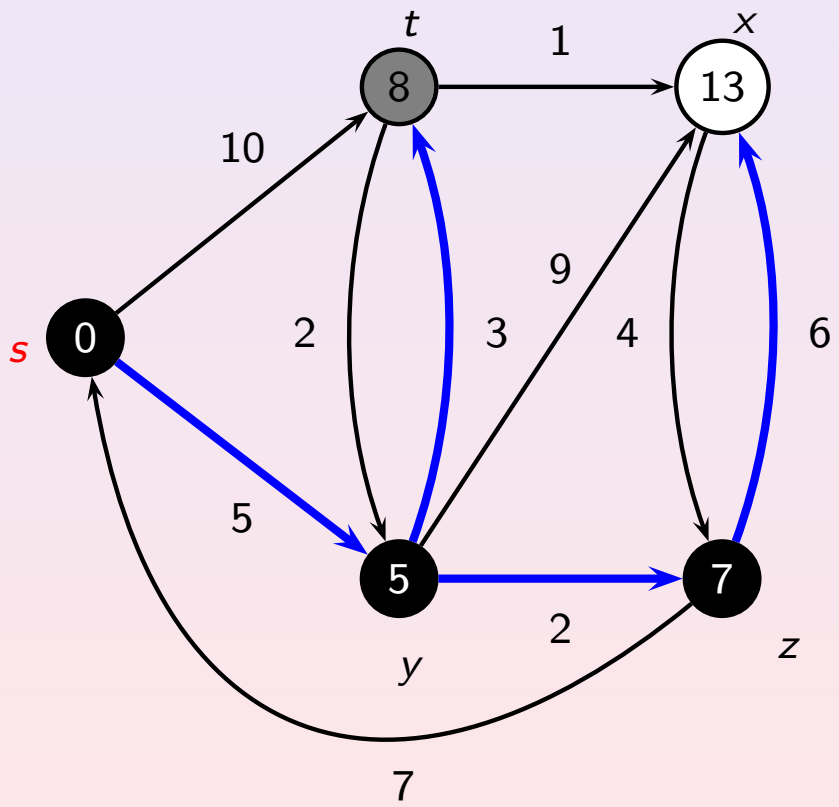
Exemplo do algoritmo Dijkstra



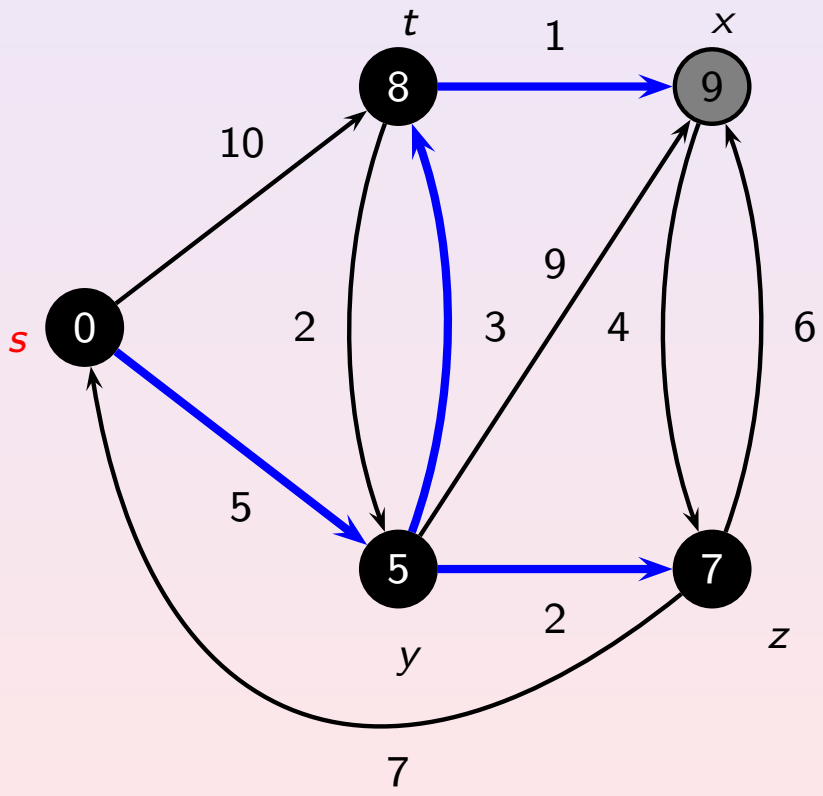
Exemplo do algoritmo Dijkstra



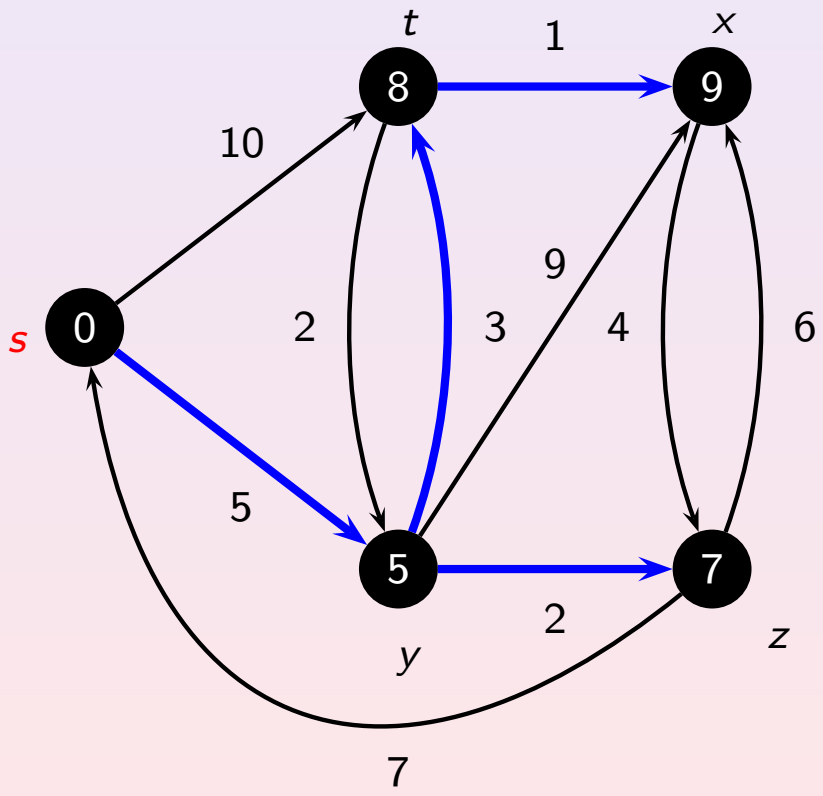
Exemplo do algoritmo Dijkstra



Exemplo do algoritmo Dijkstra



Exemplo do algoritmo Dijkstra



Complexidade

Algoritmo DIJKSTRA(G,w,s)
1. *IniciaOrigemUnica*(G, s)
2. $S \leftarrow \emptyset$
3. $Q \leftarrow V[G]$
4. **enquanto** $Q \neq \emptyset$ **faça**
5. $u \leftarrow ExtrairMinimo(Q)$
6. $S \leftarrow S \cup \{u\}$
7. **para** $v \in Adj[u]$ **faça**
8. *Relaxa*(u, v, w)

Depende da implementação de Q. As operações em Q são:

- Inserir (implícito na linha 3)
- ExtrairMinimo (linha 5) e
- DiminuirChave (implícito na linha 8, rotina *Relaxa*)

Complexidade

- $|V|$ operações Inserir.
- $|V|$ operações ExtrairMinimo.
- Cada vértice $u \in V(G)$ é inserido em S (no máximo) uma vez e cada aresta (u, v) com $v \in Adj[u]$ é examinada no $Relaxa(u, v, w)$ (no máximo) uma vez nas linhas 7-8, são $|E|$ operações de DiminuirChave.

Complexidade

No total são $|V|$ chamadas de ExtrairMinimo e $|E|$ chamadas de DiminuirChave.

- Q como um vetor: $d[v]$ na posição v do vetor. Inserir e DiminuirChave são $\Theta(1)$ e ExtrairMinimo gasta tempo $O(V)$, resultando em $O(V^2 + E) = O(V^2)$.
- Q como fila de prioridade Heap Mínimo. Inserir, ExtrairMinimo e DiminuirChave gastam tempo $O(\log V)$, resultando em um total de $O((V + E) \log V)$.
- Q como Heap de Fibonacci. ExtrairMinimo é $O(\log V)$ e DiminuirChave é $O(1)$, resulta em uma complexidade menor: $O(V \log V + E)$.

Corretude

- O algoritmo de Dijkstra é um algoritmo guloso, a escolha gulosa no algoritmo é: ExtrairMinimo.
- Precisamos provar que esta escolha gulosa é correta.

Demonstração.

Seja u o vértice obtido em ExtrairMinimo e $P_{s,k}$ um caminho mínimo de s a k. Vamos supor que obtemos o caminho mínimo relaxando primeiro o arco (x, y) a partir da escolha de x, antes de qualquer arco com cauda em u. Se o arco (u, v) pertence ao caminho mínimo, como os arcos não possuem peso negativo, $d[x] \leqslant d[y] \leqslant d[u] \leqslant d[v]$, só não representa uma contradição na escolha gulosa se $d[x] = d[y] = d[u]$, neste caso, o caminho $P_{x,u}$ tem valor nulo, a escolha de u também leva a um caminho mínimo. □

Exemplo: URI 1454 - O País das Bicicletas

Como você já deve saber, a bicicleta é um dos meios de transportes mais populares da China. Quase todos os chineses possuem a sua, e utilizam-na para trabalho, recreação, e outras atividades.
Após muitos anos pedalando, Mr. Lee já não têm a mesma disposição para encarar as diversas subidas da cidade onde mora. E a cidade em que Mr. Lee vive é extremamente montanhosa. Por razões sentimentais, ele não quer mudar para uma cidade mais plana. Resolveu, então, que tentaria evitar grandes altitudes em seus caminhos mesmo que, para isso, tivesse que pedalar um pouco mais.
Mr. Lee obteve com o serviço topográfico chinês uma coleção de mapas de sua cidade, em que cada rua desses mapas possui a informação da maior altitude encontrada quando trafegada. Tudo que ele precisa fazer agora é determinar rotas que minimizem a altura percorrida entre pares (origem, destino).
Sabendo que você planeja visitar a cidade em que ele mora no próximo ano, Mr. Lee pediu sua ajuda. Em uma primeira etapa, ele deseja que você implemente um programa que receba mapas topográficos da cidade e uma coleção de pares (origem, destino). Para cada par, seu programa deve imprimir a maior altura encontrada em uma rota entre a origem e o destino. Lembre-se que as rotas devem minimizar tais alturas. As rotas propriamente ditas, serão determinadas em uma segunda etapa (quando você chegar à China para visitá-lo).
Como o transporte utilizado é uma bicicleta, você pode considerar que todas as ruas da cidade são de mão dupla. Não demore, pois Mr. Lee conta com você. :-)

URI 1454 - Entrada:

Entrada:
Seu programa deve estar preparado para trabalhar com diversos mapas, doravante denominados instâncias. Cada instância tem a estrutura que segue.
Na primeira linha são fornecidos dois inteiros **n** ($0 \leqslant \mathbf{n} \leqslant 100$) e **m** ($0 \leqslant \mathbf{m} \leqslant 4950$) que representam, respectivamente, os números de interseções e de ruas. Por razões de clareza, as interseções são numeradas de 1 a n, inclusive; toda rua começa e termina em uma interseção; e não existem interseções fora das extremidades de uma rua.
Nas próximas m linhas são fornecidos três inteiros: i e j ($1 \leqslant \mathbf{i}, \mathbf{j} \leqslant n$) que indicam a existência de uma rua entre as interseções **i** e **j**; e **h** que representa a maior altitude encontrada quando a rua é trafegada. Esses inteiros estão separados por espaços em branco.
Na linha seguinte, é dado um inteiro **k** ($1 \leqslant \mathbf{k} \leqslant 50$) que representa o número de pares (origem, destino) que serão especificados nas próximas **k** linhas. Cada par é formado por dois inteiros **i** e **j** como acima. Isto é, origem e destino são interseções de ruas, e também estão separados por espaços em branco.
Valores $\mathbf{n} = \mathbf{m} = 0$ indicam o final das instâncias e não devem ser processados.

URI 1454 - Saída

Saída:
Para cada instância solucionada, você deverá imprimir um identificador `Instancia h` em que **h** é um número inteiro, sequencial e crescente a partir de 1. Nas próximas **k** linhas, você deve imprimir as maiores alturas encontradas nas rotas entre os **k** pares (origem, destino) fornecidos, um valor por linha, na ordem da entrada.
Uma linha em branco deve ser impressa após cada instância.

URI 1454 - País das Bicicletas: Abordagem

- O grafo é representado por um vetor, onde cada índice é um nó do grafo, representado pelo próprio valor do índice (usaremos nós de 0 a $n-1$).
- Cada nó (elemento do vetor) é representado por um par, o primeiro elemento do par, é a distância d do nó, e o segundo é um vetor de vizinhos.
- O vetor de vizinhos possui como elemento um par, onde o primeiro elemento do par é o índice do vizinho, e o segundo a altura para chegar lá.
- Eis o nosso grafo:

```
vector<pair<int, vector<pair<int, int> > > > g(n);
```

URI 1454 - País das Bicicletas: Abordagem

- Não precisamos da estrutura π (pai).
- Vamos usar uma fila de prioridade, como no STL a fila de prioridade retorna o maior, precisamos mudar o operador "<" na comparação dos vértices:

```
bool operator<(const pair<int, vector<pair<int, int> > > &a,
               const pair<int, vector<pair<int, int> > > &b) {
                 return a.first > b.first;
}
```

URI 1454 - País das Bicicletas: Abordagem

- Vamos aplicar o algoritmo de Dijkstra.
- A rotina *IniciarOrigemUnica*(G, s) precisa acontecer antes de cada consulta.
- Vamos criar a rotina `reset` que irá colocar infinito (`MAXINT`) em cada vértice.
- Também, para o vértice `s` vamos colocar o valor `MININT` pois como **h** não tem a restrição de ser positivo, poderá ser negativo, logo 0 não é o menor valor.

```
void reset(vector<pair<int, vector<pair<int, int> > > > &g) {
  for(auto it = g.begin(); it!= g.end(); it++) it->first=MAXINT;
}
```

URI 1454: Construindo o grafo

```
int n, m, x, y, w, k, s, d, inst=0;

cin >> n >> m;
while(n) {
 inst++; cout << "Instancia " << inst << endl;
 vector<pair<int, vector<pair<int, int> > > > g(n);
 for(int i=0; i<m; i++) {
  cin >> x >> y >> w; x--; y--;
  g[x].second.push_back(make_pair(y,w));
  g[y].second.push_back(make_pair(x,w));
 }
```

URI 1454: Consulta e Dijkstra

```
cin >> k;
for(int i=0; i<k; i++) {
 cin >> s >> d; s--; d--;
 if(s==d) cout << 0 << endl;
 else {
  reset(g); g[s].first=MININT; // IniciaOrigemUnica
  priority_queue<pair<int, vector<pair<int, int> > > > Q; Q.push(g[s]);
  while(!Q.empty()) {
   auto u = Q.top(); Q.pop();
   for(auto v = u.second.begin(); v != u.second.end(); v++) {
    int h = max(u.first,v->second);
    if(h < g[v->first].first) { // Relaxa
     g[v->first].first = h; Q.push(g[v->first]);
    }
   }
  }
  cout << g[d].first << endl;
 }
}
cout << endl;
cin >> n >> m;
```

Algoritmo de Bellman Ford

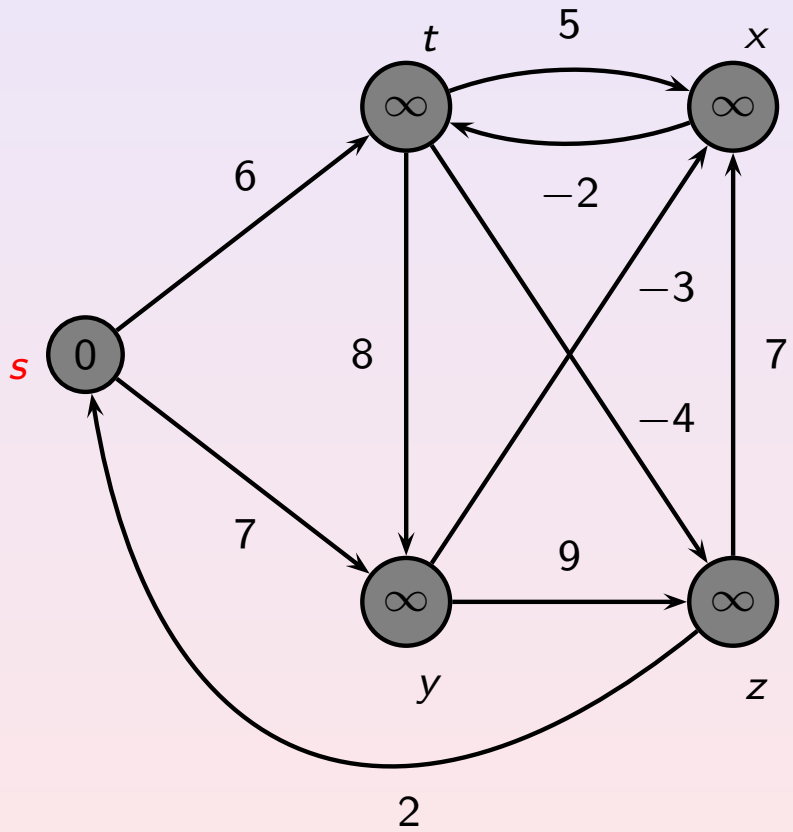
- Os algoritmos vistos consideram algumas restrições: Ou não são permitidos ciclos, ou não são permitidos arcos de peso negativo.
- O algoritmo de Bellman Ford permite relaxar estas restrições, são permitidos ciclos e arcos de peso negativo, no entanto não é permitido um ciclo de peso negativo (ou simplesmente ciclo negativo).
- Se houver um ciclo negativo o algoritmo irá parar indicando a presença de um ciclo negativo.

O Algoritmo

- Entrada: Grafo (G, w) orientado e um vértice de origem s de G.
- Saída: `FALSO` - se existe um ciclo negativo atingível a partir de s, ou `VERDADE` - neste caso também devolve uma Árvore de Caminhos Mínimos com raiz s e o vetor $d[]$ com a distância de s a cada vértice.

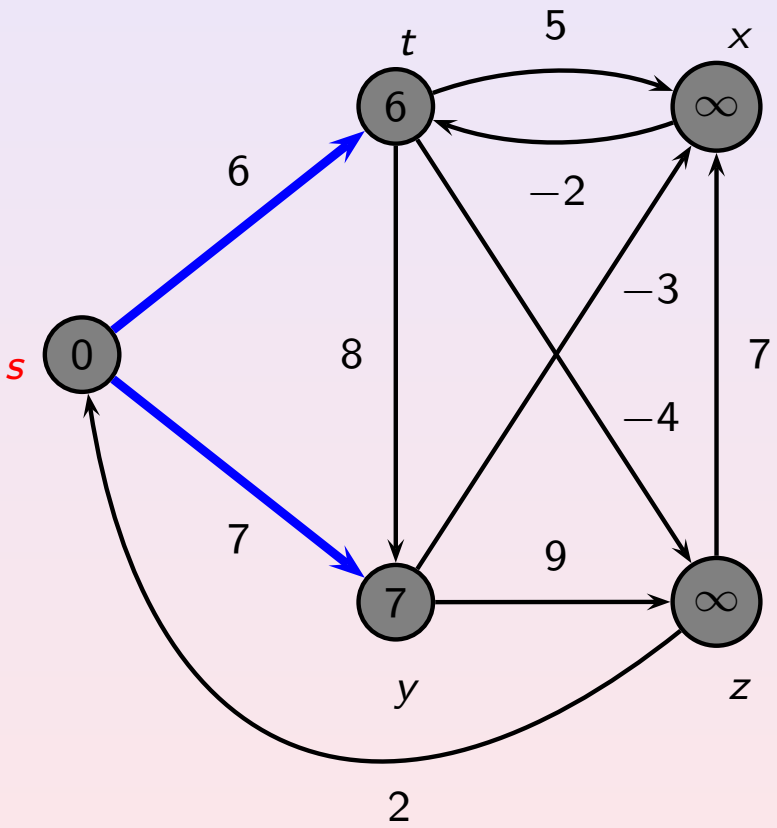
Algoritmo BELLMAN-FORD(G,w,s)
  IniciaOrigemUnica(G, s)
  para $i \leftarrow 1$ **até** $|V(G)| - 1$ **faça**
    para $(u, v) \in E(G)$ **faça**
      Relaxa(u, v, w)
  para $(u, v) \in E(G)$ **faça**
    se $d[v] > d[u] + w(u, v)$ **então retorne** `FALSO`
  retorne `VERDADE`

Exemplo de aplicação do algoritmo



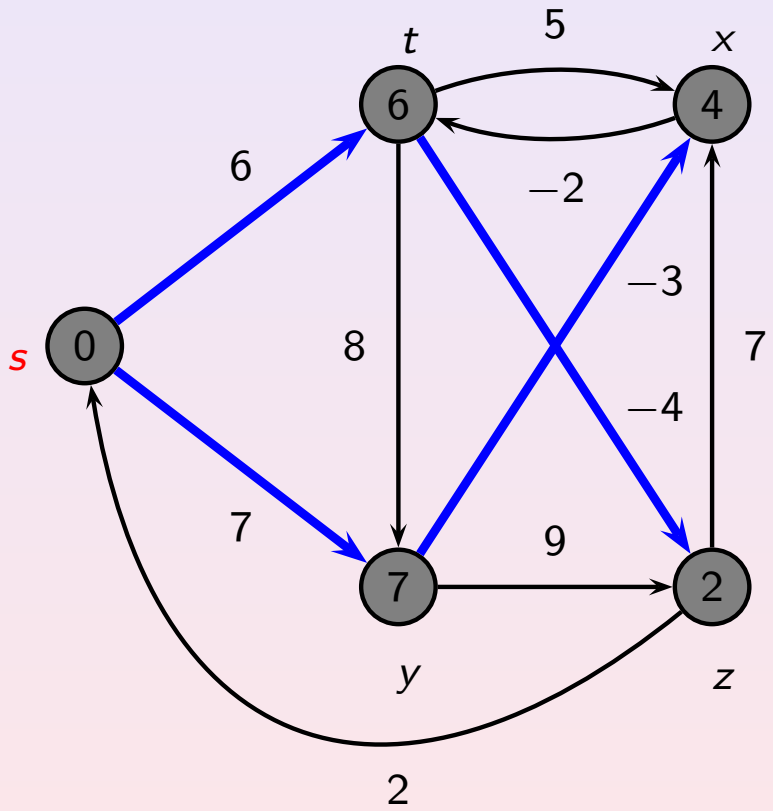
Ordem:
$(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$.

Exemplo de aplicação do algoritmo



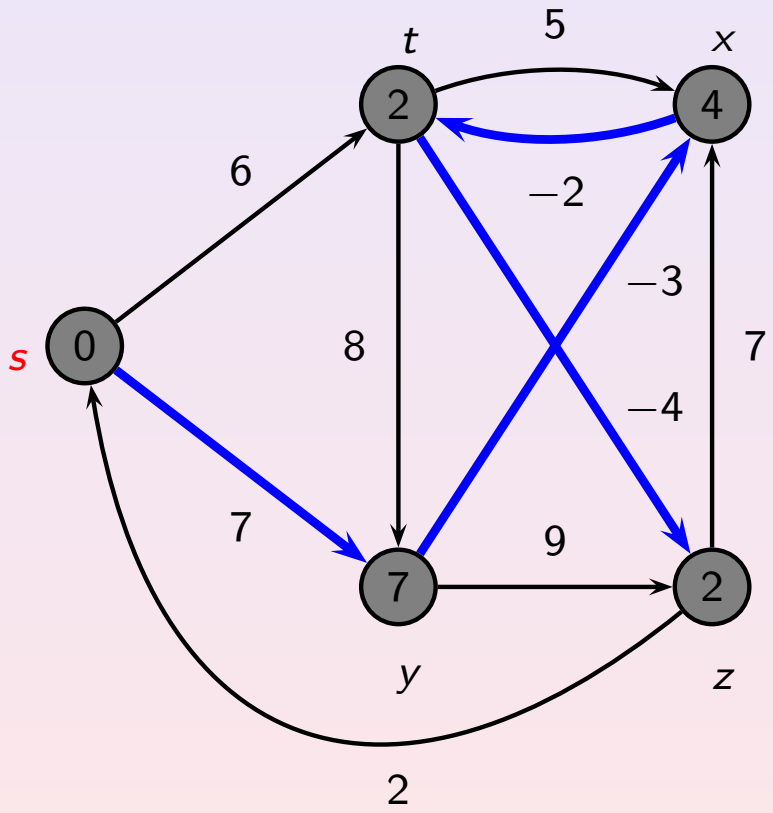
Ordem:
$(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y).$

Exemplo de aplicação do algoritmo



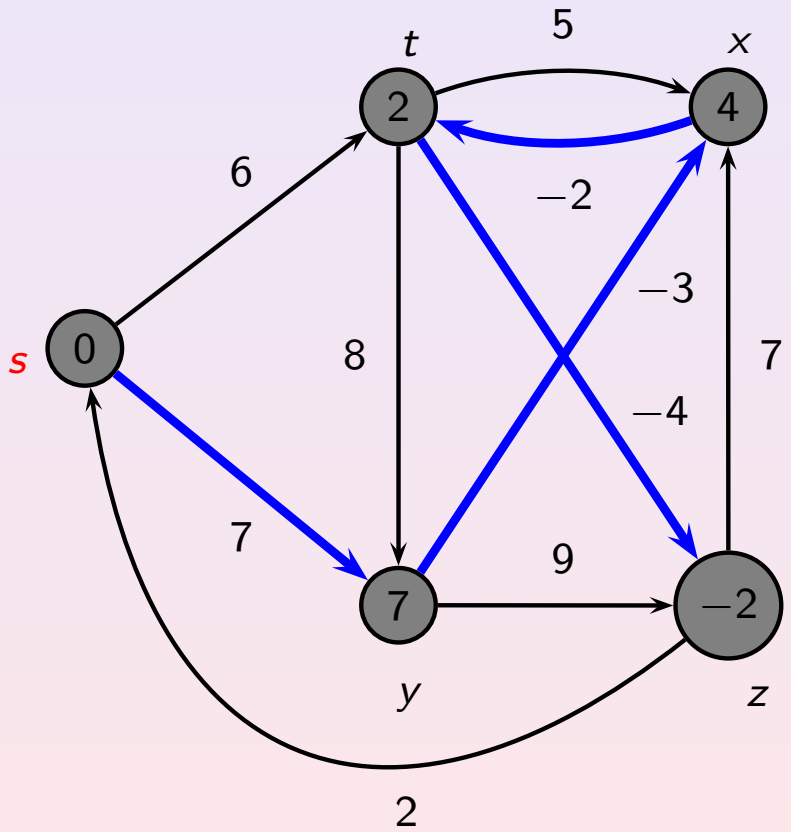
Ordem:
$(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$.

Exemplo de aplicação do algoritmo



Ordem:
$(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y).$

Exemplo de aplicação do algoritmo



Ordem:
$(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y).$

Análise da complexidade e corretude

- Complexidade de tempo é $O(VE)$
- Para corretude, precisamos mostrar que o algoritmo relaxa as arestas de um caminho na ordem correta.
- Com $|V| - 1$ relaxamentos termina, se houver ciclo negativo, novo relaxamento deve mudar algum valor.

Demonstração.

Um caminho que possui $|V|$ vértices, possui $|V| - 1$ arestas. Pela propriedade do Limite Superior se $d[v] = dist(s, v)$ seu valor nunca mais muda. Pela propriedade da Convergência, se $d[u] = dist(s, u)$ e (u, v) é um caminho mínimo de s a v, ao relaxar a aresta (u, v), $d[v] = dist(s, v)$. Por indução no número de vezes que as arestas são relaxadas, na i-ésima iteração, tendo $d[v_i] = dist(s, i)$ a aresta (v_i, v_{i+1}) ao ser relaxada, faz com que $d[v_{i+1}] = dist(s, v_{i+1})$. As arestas são relaxadas na ordem correta. □

Atividades baseadas no CLRS

1. Ler os capítulos: 24, 24.1, 24.2, 24.3 e 24.5.
2. Exercícios 24.1-1, 24.1-2, 24.1-3, 24.1-4, 24.2-1, 24.2-2, 24.2-4, 24.3-1, 24.3-2, 24.3-3, 24.3-4, 24.3-8, 24.5-1, 24.5-4, 24.5-7.
3. Problema: 24-3