

Universidade Federal do Pará
Instituto de Ciências Exatas e Naturais
Faculdade de Computação

GRAFOS

Caminhos Mínimos com uma Fonte

Nelson Cruz Sampaio Neto
nelsonneto@ufpa.br

Sumário

1. Introdução
2. Representação
3. Relaxamento
4. Algoritmo de Dijkstra
5. Algoritmo de Bellman-Ford
6. Algoritmo de Floyd-Warshall

Introdução

Cenário

- Um motorista deseja encontrar o caminho mais curto da cidade de Belém a São Paulo.
- Dado um mapa das rodovias brasileiras, como podemos determinar a rota mais curta?

Introdução

Solução possível

- Uma possibilidade consiste em enumerarmos todos os possíveis caminhos de Belém a São Paulo.
- Adicionar as distâncias em cada roda.
- Selecionar a rota mais curta.

Esse método é satisfatório?



Introdução

Definição

- Em problemas de caminhos mínimos, é dado um grafo ponderado $G = (V, E)$ cujas arestas têm pesos w .
- O peso de um caminho $c = (v_0, v_1, \dots, v_k)$ é a soma dos pesos das arestas que o formam:

$$w(c) = \sum_{i=0}^{k-1} w(v_i, v_{i+1})$$

Introdução

Definição

- O peso do caminho mais curto entre **u** e **v** é :

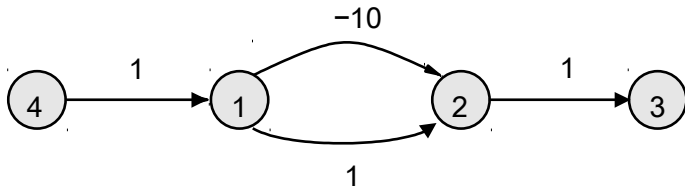
$$\delta(u, v) = \begin{cases} \text{Min } \{ w(c) : c \text{ é caminho de } u \text{ até } v \} \\ +\infty \text{ se não existir caminho} \end{cases}$$

Introdução

Definição

- Em algumas instâncias do problema de caminhos mínimos com uma fonte, podem existir arestas com pesos negativos.
- Se o grafo $G = (V, E)$ não tem ciclo com peso negativo alcançável a partir da fonte s , então, $\delta(s, v)$ permanece bem definido para todo $v \in V$.

Pesos negativos



Custos negativos ocorrem naturalmente em muitos problemas práticos. Por exemplo, se o custo positivo representa despesa, o custo negativo representa **lucro**. Sob custos negativos, um caminho mínimo **maximiza** lucro.

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, os algoritmos que aceitam valores negativos nas arestas detectam a presença de ciclo negativo.

Introdução

Objetivo

- Obter os caminhos mais curtos a partir de uma origem.
- Dado um grafo ponderado $G = (V, E)$, desejamos obter o caminho mais curto (ou seja, de menor peso) a partir de um dado vértice origem $s \in V$ até qualquer $v \in V$.
- No exemplo do problema Belém - São Paulo:
 - Os vértices representam as cidades;
 - As arestas representam as rodovias; e
 - Os pesos representam as distâncias entre duas cidades.

Introdução

Variantes

1. Caminhos mínimos com destino único:
 - * Encontre o caminho mais curto até o vértice v a partir de cada vértice $u \in V$.
 - * Calcule G^T e encontre o caminho mais curto a partir de v .
2. Caminhos mínimos entre um par de vértices:
 - * Encontre o caminho mais curto de u até v .
3. Caminhos mínimos entre todos os pares de vértices:
 - * Encontre o caminho mínimo entre cada par $u, v \in V$.
 - * Outra solução é o algoritmo de Floyd-Warshall.

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.
- Dado um grafo $G = (V, E)$, mantemos para cada vértice $v \in V$ o seu predecessor $\pi[v]$ que é outro vértice ou *nil*.
- Os atributos π são definidos de tal maneira que a cadeia de predecessores originada em v nos dá o caminho mais curto de s para v em termos dos pesos de cada aresta de G .

Relaxamento

- Vários algoritmos são baseados na técnica de relaxamento.
- Para cada vértice $v \in V$, mantemos um atributo $d[v]$, que é um limite superior para o caminho mais curto entre s e v :

$$d[v] \geq \delta(s, v)$$

- O atributo $d[v]$ é uma estimativa para $\delta(s, v)$.

Relaxamento

Initialize-Single-Source (G, s)

- 1) for each $v \in V[G]$
- 2) do $d[v] = \infty$
- 3) $\pi[v] = \text{nil}$
- 4) $d[s] = 0$

Após a inicialização, $\pi[v] = \text{nil}$ para todo $v \in V$; $d[s] = 0$; e $d[v] = \infty$ para todo $v \in V - \{s\}$.

Relaxamento

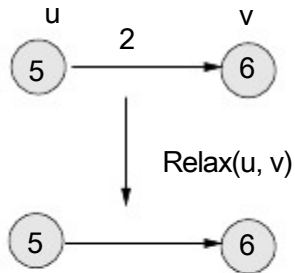
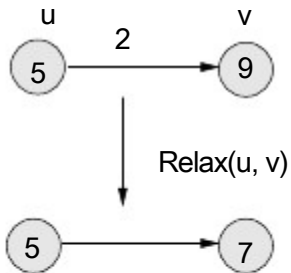
Relax (u, v, w)

- 1) if $d[v] > d[u] + w(u, v)$
- 2) then $d[v] = d[u] + w(u, v)$
- 3) $\pi[v] = u$

➤ O propósito de relaxar a aresta (u, v) consiste em testar se é possível melhorar a estimativa do caminho mais curto encontrado, até então, da fonte até o vértice v passando-se pelo vértice u .

➤ Para toda aresta $(u, v) \in E$, tem-se $\delta(s, v) \leq \delta(s, u) + w(u, v)$.

Relaxamento



Algoritmo de Dijkstra

Princípios

- O algoritmo de Dijkstra resolve de forma ótima o problema de caminhos mínimos com uma fonte em um grafo ponderado.
- O algoritmo mantém um conjunto **S** de vértices, onde para todo $v \in S$, temos $d[v] = \delta(s, v)$.

Algoritmo de Dijkstra

Princípios (cont.)

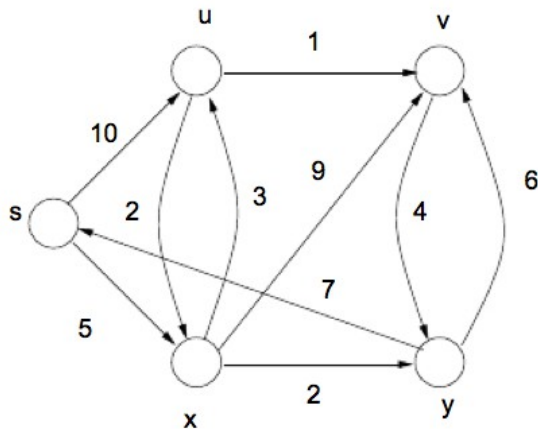
- O algoritmo iterativamente seleciona um vértice $u \in V - S$ com a menor estimativa de distância e insere u em S .
- Ao mesmo tempo que “relaxa” as arestas que emanam de u .
- As arestas não podem apresentar pesos negativos.

Algoritmo de Dijkstra

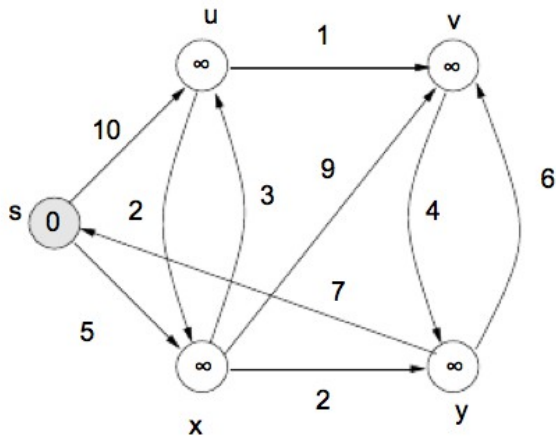
Dijkstra (G, w, s)

- 1) Initialize-Single-Source(G, s)
- 2) $S = \emptyset$
- 3) $Q = V[G]$ // cria fila de prioridade Q com os vértices do grafo
- 4) while ($Q \neq \emptyset$)
- 5) do $u = \text{extract-min}(Q)$
- 6) $S = S \cup \{u\}$
- 7) for each $(u, v) \in \text{Adj}[u]$
- 8) do Relax (u, v, w)

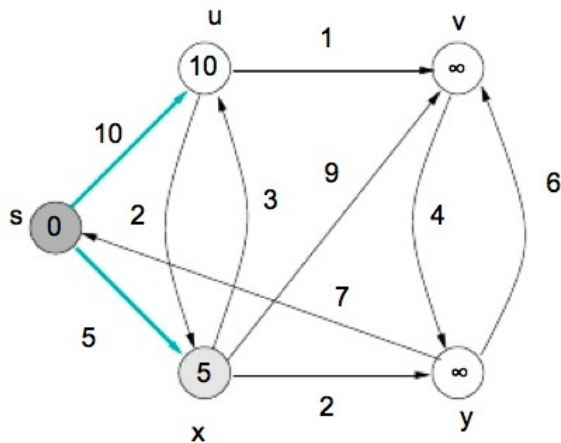
Exemplo de execução do Dijkstra



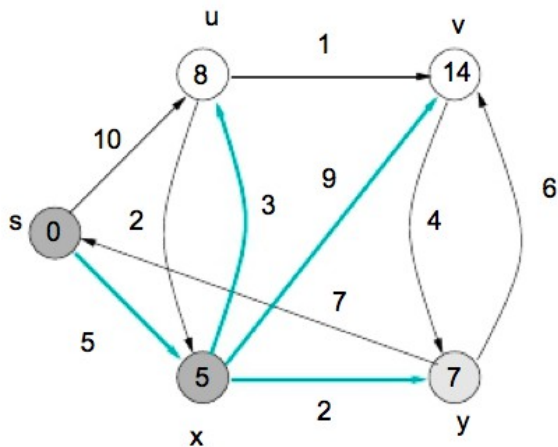
Dijkstra: inicialização



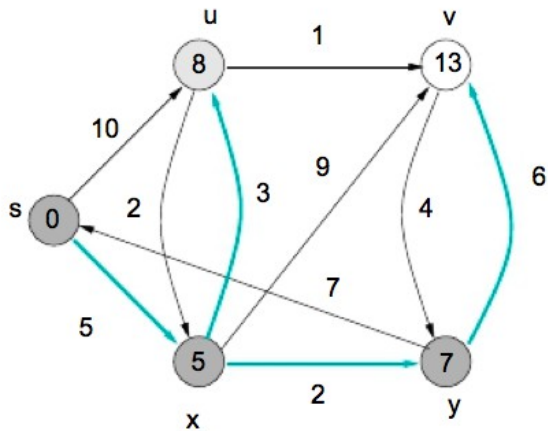
Dijkstra: iteração 1



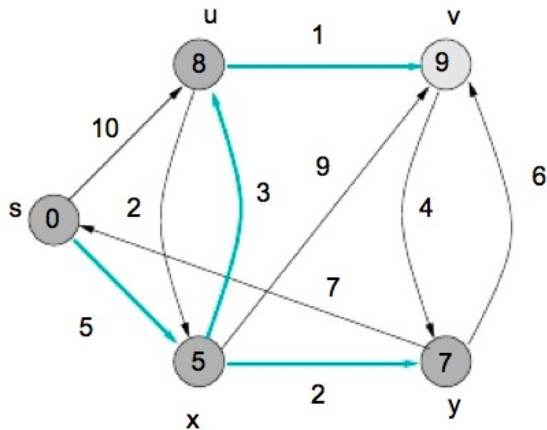
Dijkstra: iteração 2



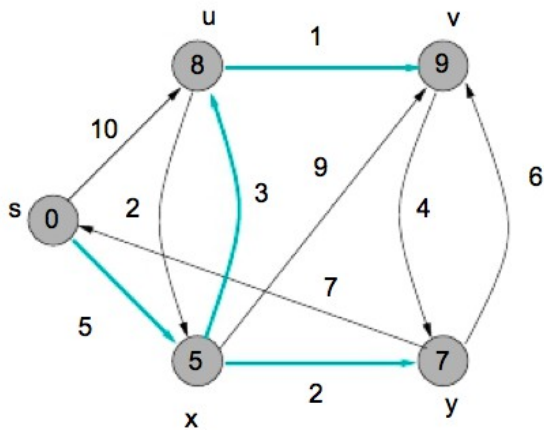
Dijkstra: iteração 3



Dijkstra: iteração 4



Dijkstra: iteração 5



Análise do algoritmo de Dijkstra

Vetor como fila de prioridades

- Neste caso, Extract-min leva tempo $O(v)$ e há $|V|$ operações para esse procedimento.
- Uma vez que cada aresta é examinada no máximo uma vez, a operação Relax é executada no máximo $|E|$ vezes.
- Assim, algoritmo tem complexidade no tempo:

$$O(v^2 + e) = O(v^2).$$

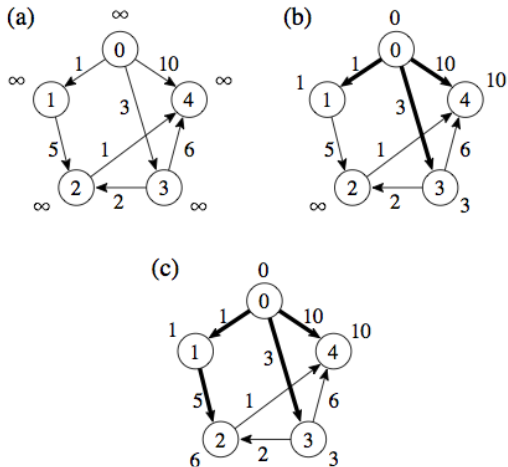
Análise do algoritmo de Dijkstra

Heap binário como fila de prioridades

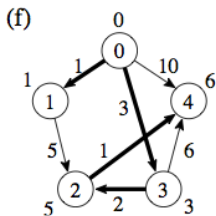
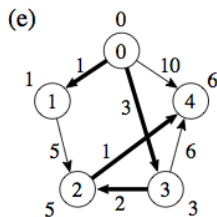
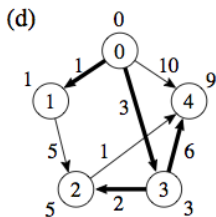
- Quando o grafo é esparso é mais prático utilizarmos uma fila de prioridades implementada p.e. com heap binário.
- O Extract-min leva tempo $O(\log v)$ e há $|V|$ operações para esse procedimento.
- A operação Relax custa no máximo $O(\log v)$, quando a distância $d[v]$ for reduzida, e há $|E|$ operações.
- Complexidade no tempo: $O(v \log v + e \log v) = O(e \log v)$.

Análise do algoritmo de Dijkstra

- Hoje, sabe-se que para grafos esparsos o algoritmo de Dijkstra pode ser implementado de forma mais eficiente usando lista de adjacência para armazenar o grafo e um **heap de Fibonacci** para implementar a fila de prioridade.
- Com essa estrutura, o algoritmo tem a seguinte complexidade no tempo: $O(v \log v + e)$.
- Isso porque a operação de decremento no heap de Fibonacci leva $O(1)$ em uma análise amortizada.

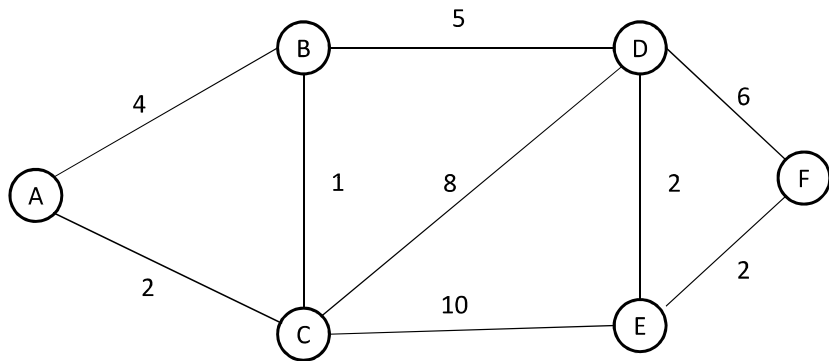


Iteração	S	$d[0]$	$d[1]$	$d[2]$	$d[3]$	$d[4]$
(a)	\emptyset	∞	∞	∞	∞	∞
(b)	$\{0\}$	0	1	∞	3	10
(c)	$\{0, 1\}$	0	1	6	3	10



Iteração	S	$d[0]$	$d[1]$	$d[2]$	$d[3]$	$d[4]$
(d)	$\{0, 1, 3\}$	0	1	5	3	9
(e)	$\{0, 1, 3, 2\}$	0	1	5	3	6
(f)	$\{0, 1, 3, 2, 4\}$	0	1	5	3	6

Problema: Encontrar o menor caminho entre A e F.



Algoritmo de Bellman-Ford

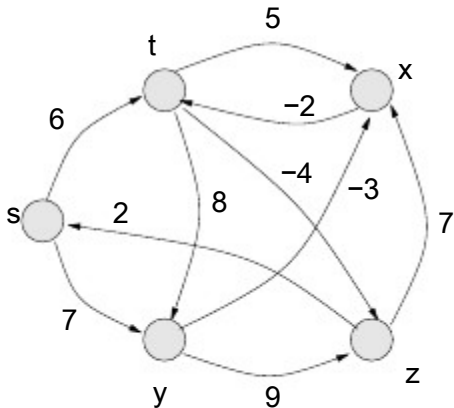
- O algoritmo de Bellman-Ford resolve o problema de caminhos mínimos com uma fonte no caso geral, onde as arestas podem ter peso negativo.
- O algoritmo retorna um valor Booleano indicando se foi, ou não, encontrado um ciclo negativo alcançável a partir de s .
- Caso negativo, os caminhos mínimos com raiz em s e seus respectivos pesos são determinados.

Algoritmo de Bellman-Ford

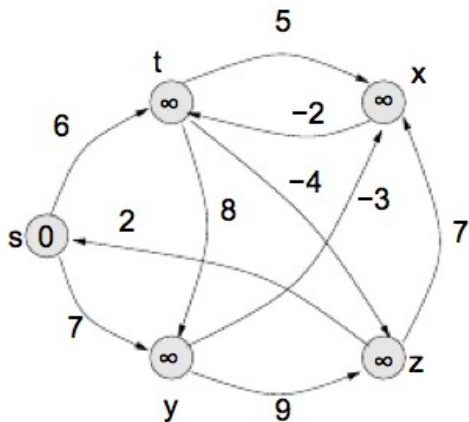
Bellman-Ford(G, w, s)

- 1) Initialize-Single-Source(G, s)
- 2) for $i = 1$ to $|V[G]| - 1$
- 3) do for each $(u, v) \in E[G]$
- 4) do Relax(u, v, w)
- 5) for each $(u, v) \in E[G]$
- 6) do if $d[v] > d[u] + w(u, v)$
- 7) then return FALSE
- 8) return TRUE

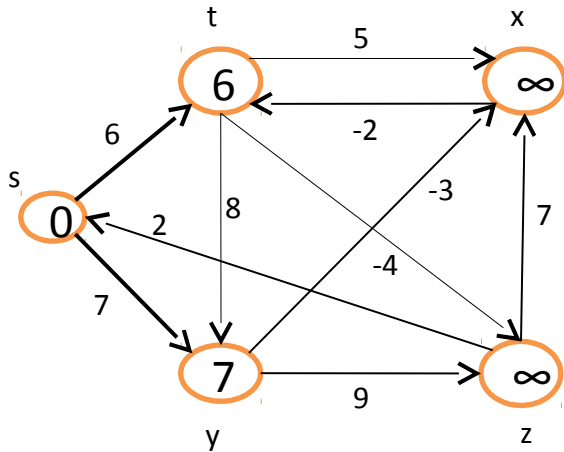
Exemplo de execução do Bellman-Ford



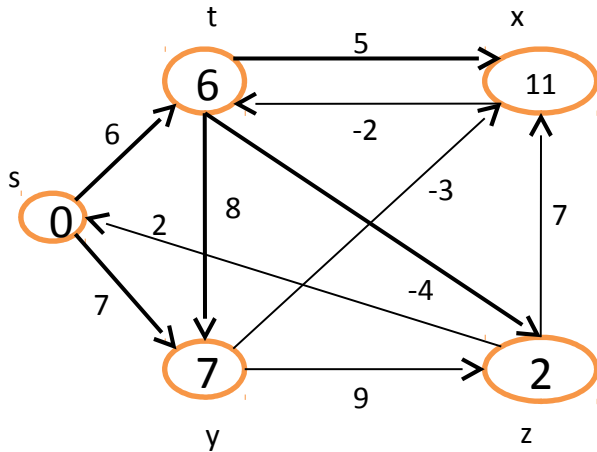
Bellman-Ford: inicialização



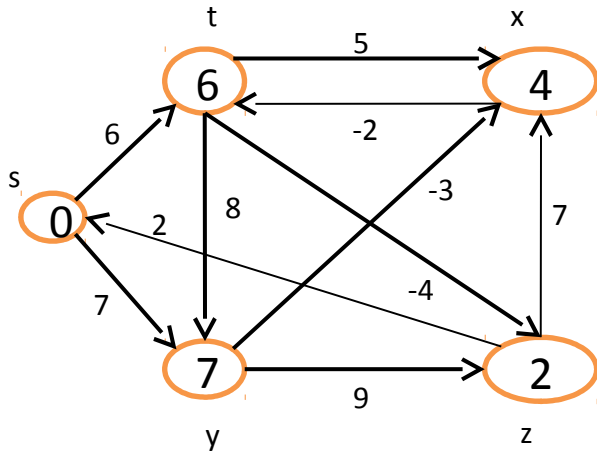
Bellman-Ford: iteração 1



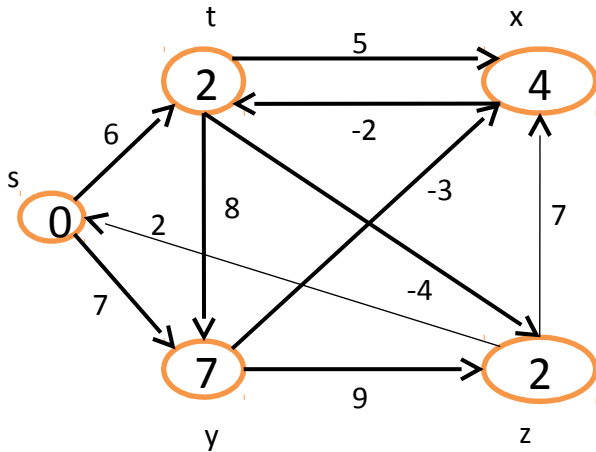
Bellman-Ford: iteração 1



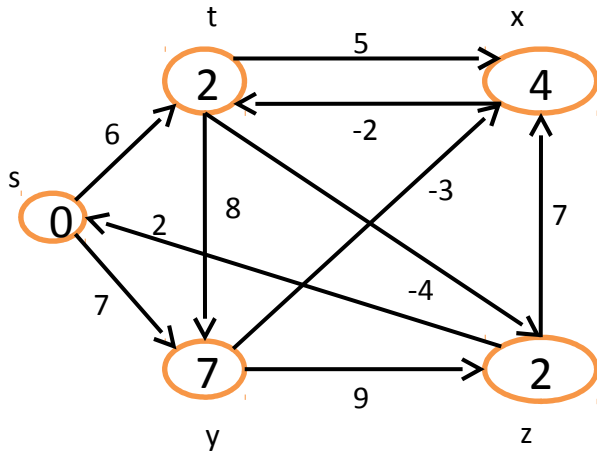
Bellman-Ford: iteração 1



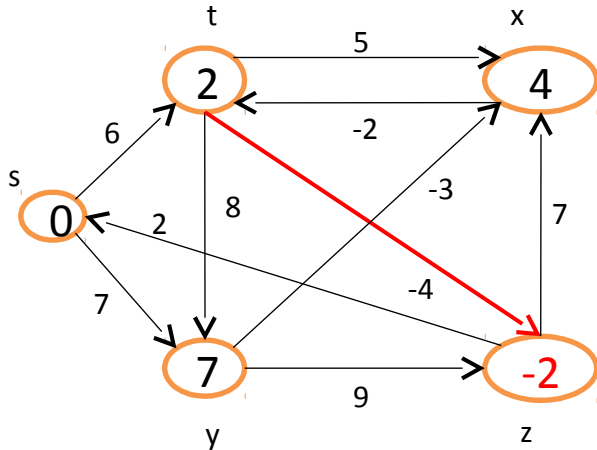
Bellman-Ford: iteração 1



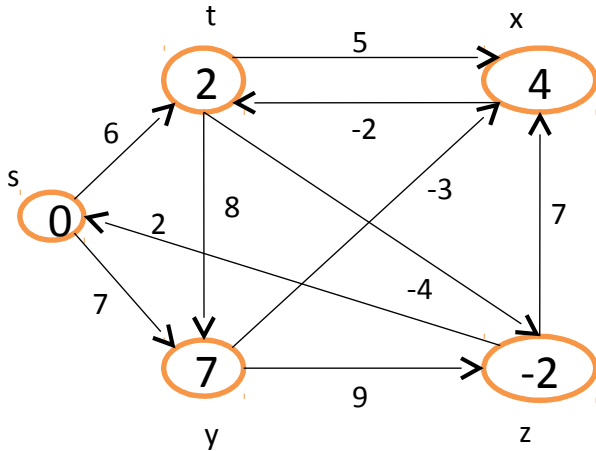
Bellman-Ford: iteração 1



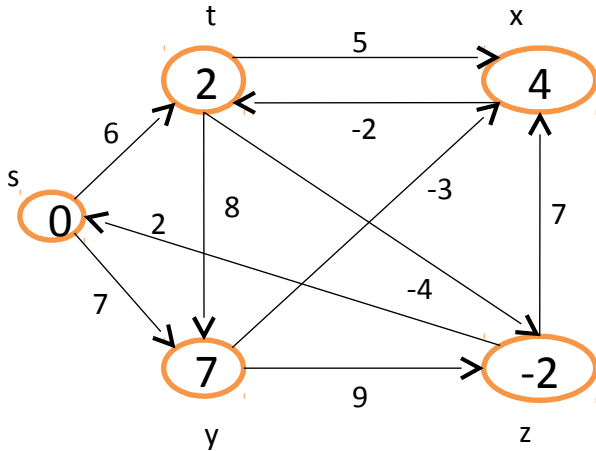
Bellman-Ford: iteração 2



Bellman-Ford: iteração 3



Bellman-Ford: iteração 4



Análise do algoritmo de Bellman-Ford

- Fazendo $|V| = v$ e $|E| = e$, podemos verificar que Bellman-Ford executa em tempo $O(v \cdot e)$.
- Para um grafo denso, $e = O(v^2)$, o tempo de execução do algoritmo é $O(v^3)$.
- O tempo de processamento pode ser otimizado inserindo um “escape” após o passo 4, caso a procedure Relax não altere nenhum limite superior.

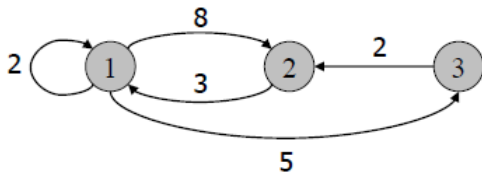
Algoritmo de Floyd-Warshall

- Suponha que um grafo ponderado representa as rotas de uma companhia aérea conectando diversas cidades.
- O objetivo é construir uma tabela com os menores caminhos entre todas as cidades.
- Esse é um exemplo de problema que exige encontrar os caminhos mais curtos entre todos os pares de vértices.

Algoritmo de Floyd-Warshall

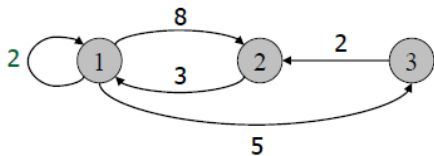
- Uma possível solução é usar o algoritmo de Dijkstra utilizando cada vértice como origem alternadamente.
- Uma solução mais direta é utilizar Floyd-Warshall.
- O algoritmo de Floyd-Warshall usa uma matriz $|V| \times |V|$ para calcular e armazenar os tamanhos dos caminhos mais curtos.

Algoritmo de Floyd-Warshall



	1	2	3
1			
2			
3			

Algoritmo de Floyd-Warshall

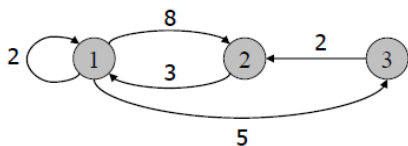


	1	2	3
1	0	8	5
2	3	0	∞
3	∞	2	0

- Pesos de self-loops não são considerados
- A matriz A é percorrida $|V|$ vezes
- A cada iteração k ,

$$A[v,w] = \min(A[v,w], A[v,k] + A[k,w]).$$

Algoritmo de Floyd-Warshall



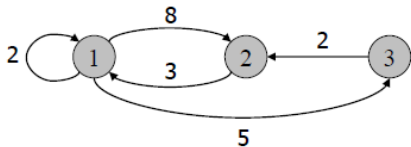
	1	2	3
1	0	8	5
2	3	0	∞
3	∞	2	0

Ou seja:

$$A[1,1] = \min(A[1,1], A[1,1] + A[1,1]).$$

$k = 1$

Algoritmo de Floyd-Warshall



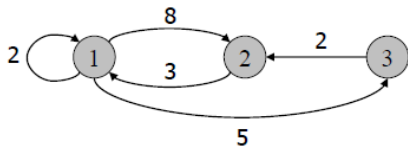
	1	2	3
1	0	8	5
2	3	0	∞
3	∞	2	0

Ou seja:

$$A[1,2] = \min(A[1,2], A[1,1] + A[1,2]).$$

k = 1

Algoritmo de Floyd-Warshall



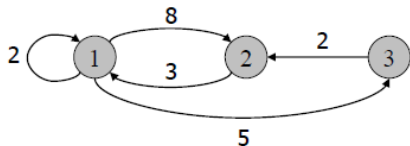
	1	2	3
1	0	8	5
2	3	0	∞
3	∞	2	0

Ou seja:

$$A[1,3] = \min(A[1,3], A[1,1] + A[1,3]).$$

k = 1

Algoritmo de Floyd-Warshall



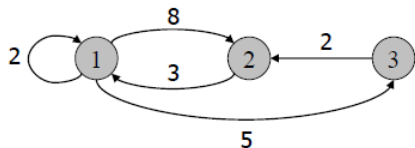
	1	2	3
1	0	8	5
2	3	0	∞
3	∞	2	0

Ou seja:

$$A[2,1] = \min(A[2,1], A[2,1] + A[1,1]).$$

$$k = 1$$

Algoritmo de Floyd-Warshall



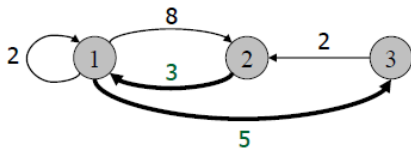
Ou seja:

$$A[2,2] = \min(A[2,2], A[2,1] + A[1,2]).$$

	1	2	3
1	0	8	5
2	3	0	∞
3	∞	2	0

$$k = 1$$

Algoritmo de Floyd-Warshall



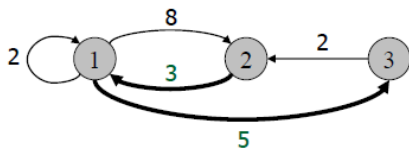
	1	2	3
1	0	8	5
2	3	0	∞
3	∞	2	0

Ou seja:

$$A[2,3] = \min(A[2,3], A[2,1] + A[1,3]).$$

$$k = 1$$

Algoritmo de Floyd-Warshall



	1	2	3
1	0	8	5
2	3	0	8
3	∞	2	0

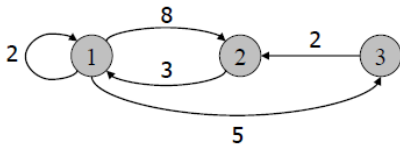
Ou seja:

$$A[2,3] = \min(A[2,3], A[2,1] + A[1,3]).$$

$$\pi[3] = 1$$

$$k = 1$$

Algoritmo de Floyd-Warshall



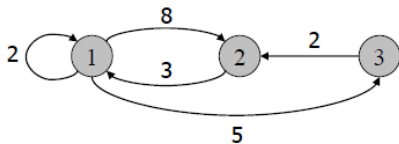
	1	2	3
1	0	8	5
2	3	0	8
3	∞	2	0

Ou seja:

$$A[3,1] = \min(A[3,1], A[3,1] + A[1,3]).$$

$$k = 1$$

Algoritmo de Floyd-Warshall



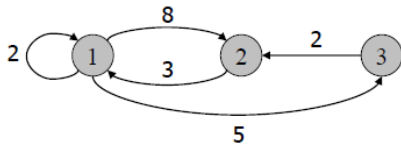
	1	2	3
1	0	8	5
2	3	0	8
3	∞	2	0

Ou seja:

$$A[3,2] = \min(A[3,2], A[3,1] + A[1,2]).$$

k = 1

Algoritmo de Floyd-Warshall



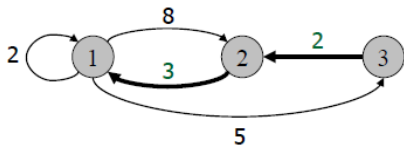
	1	2	3
1	0	8	5
2	3	0	8
3	∞	2	0

Ou seja:

$$A[3,3] = \min(A[3,3], A[3,1] + A[1,3]).$$

$$k = 1$$

Algoritmo de Floyd-Warshall

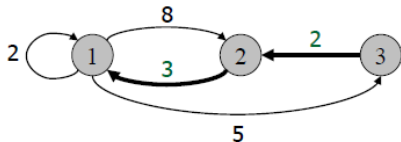


$$A[3,1] = \min(A[3,1], A[3,2] + A[2,1]).$$

	1	2	3
1	0	8	5
2	3	0	8
3	∞	2	0

$$k = 2$$

Algoritmo de Floyd-Warshall



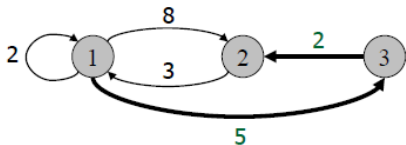
$$A[3,1] = \min(A[3,1], A[3,2] + A[2,1]).$$

$$\pi[1] = 2$$

	1	2	3
1	0	8	5
2	3	0	8
3	5	2	0

$$k = 2$$

Algoritmo de Floyd-Warshall

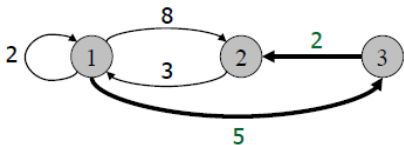


$$A[1,2] = \min(A[1,2], A[1,3] + A[3,2]).$$

	1	2	3
1	0	8	5
2	3	0	8
3	5	2	0

$$k = 3$$

Algoritmo de Floyd-Warshall



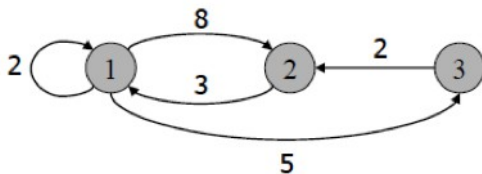
$$A[1,2] = \min(A[1,2], A[1,3] + A[3,2]).$$

$$\pi[2] = 3$$

	1	2	3
1	0	7	5
2	3	0	8
3	5	2	0

$$k = 3$$

Algoritmo de Floyd-Warshall



	1	2	3
1	0	7	5
2	3	0	8
3	5	2	0

$$\pi[1] = 2$$

$$\pi[2] = 3$$

$$\pi[3] = 1$$

Algoritmo de Floyd-Warshall

```
início
  para v:=1 até G.NumVertices faça
    para w:=1 até G.NumVertices faça
      se v = w então
         $A[v,w] := 0;$ 
      senão
         $A[v,w] := \text{peso da aresta } (v, w);$ 

  para k:=1 até G.NumVertices faça
    para v:=1 até G.NumVertices faça
      para w:=1 até G.NumVertices faça
        se  $A[v,k] + A[k,w] < A[v,w]$  então
           $A[v,w] := A[v,k] + A[k,w];$ 
           $\pi[w] := k;$ 

fim;
```

Análise do algoritmo de Floyd-Warshall

- O algoritmo de Floyd-Warshall tem complexidade $O(v^3)$.
- O algoritmo de Floyd-Warshall apresenta desempenho similar ao algoritmo de Dijkstra com vetor de prioridade.
- Se o algoritmo de Dijkstra for implementado p.e. com heap binário, sua complexidade para encontrar os caminhos mais curtos entre todos os pares de vértices é $O(v \text{ e } \log v)$.
- Para identificar a presença de ciclos negativos: $A_{ii} < 0$.