

PCC173/BCC463 - Otimização em Redes

Marco Antonio M. Carvalho

Departamento de Computação
Instituto de Ciências Exatas e Biológicas
Universidade Federal de Ouro Preto



1 Algoritmo de Johnson para Grafos Esparsos

Fonte

Este material é baseado no material

- ▶ Alex Chumbley, Karleigh Moore, and Jimin Khim. *Johnson's Algorithm*. Disponível em <https://brilliant.org/wiki/johnsons-algorithm/>.
- ▶ Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2002). *Algoritmos: teoria e prática*. Editora Campus.

Licença

Este material está licenciado sob a Creative Commons BY-NC-SA 4.0. Isto significa que o material pode ser compartilhado e adaptado, desde que seja atribuído o devido crédito, que o material não seja utilizado de forma comercial e que o material resultante seja distribuído de acordo com a mesma licença.

Histórico

O algoritmo proposto por Donald B. Johnson em 1977^a determina os caminhos mais curtos entre todos os pares de vértices em grafos:

- ▶ Direcionados;
- ▶ Ponderados;
- ▶ Com arcos de peso negativo;
- ▶ Sem ciclos de peso negativo.

Para grafos esparsos, ele é assintoticamente melhor do que o algoritmo de *Floyd-Warshall*.

^aJohnson, Donald B. (1977), "Efficient algorithms for shortest paths in sparse networks", Journal of the ACM 24 (1): 1–13, doi:10.1145/321992.321993.

Algoritmo de *Johnson* para Grafos Esparsos

Princípio

O algoritmo de *Johnson* retorna uma matriz de pesos de caminhos mais curtos ou informa que o grafo de entrada contém um ciclo de peso negativo.

Utiliza como sub-rotinas os algoritmos de *Dijkstra* e *Bellman-Ford*.

Utiliza também uma técnica de **reponderação** para eliminar arcos de peso negativo.

Algoritmo de *Johnson* para Grafos Esparsos

Reponderação

Se todos os pesos de arcos w em um grafo $G=(V, E)$ são não negativos, podemos encontrar caminhos mais curtos entre todos os pares de vértices executando o algoritmo de *Dijkstra* uma vez a partir de cada vértice.

Se G possui arcos de peso negativo, mas nenhum ciclo de peso negativo, simplesmente calculamos um novo conjunto de pesos de arcos não negativos que nos permita aplicar o mesmo método.

O novo conjunto de pesos \hat{w} deve satisfazer a duas propriedades importantes:

- 1 Para todos os pares de vértices $u, v \in V$, um caminho p é um caminho mais curto de u até v usando a função de peso w se e somente se p também é um caminho mais curto desde u até v usando a função de peso \hat{w} ;
- 2 Para todos os arcos (u, v) , o novo peso $\hat{w}(u, v)$ é não negativo.

Reponderação

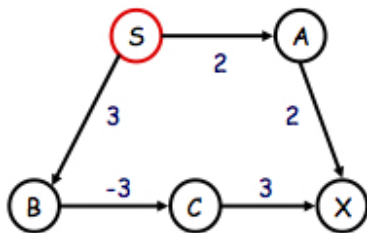
Intuitivamente, podemos tentar reponderar os pesos dos arcos subtraindo o valor do arco de menor custo.

Entretanto, esta estratégia não funciona.

Se houver múltiplos caminhos entre os vértices u e v , então cada um deles deve ser acrescido do mesmo valor, de maneira a preservar a consistência do caminho mais curto em ambos os grafos.

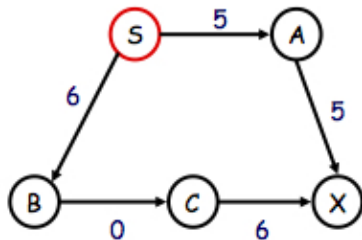
O algoritmo de Johnson utiliza um pré-processamento que associa a cada vértice v um peso $h(v)$, tal que os caminhos entre todos os pares de vértices são acrescidos do mesmo valor, e nenhuma aresta possui peso negativo.

Algoritmo de *Johnson* para Grafos Esparsos



Grafo original.

Algoritmo de *Johnson* para Grafos Esparsos



Reponderação que implica em inconsistência dos caminhos mais curtos (e.g., entre s e x).

Algoritmo de *Johnson* para Grafos Esparsos

Reponderação

O pré-processamento de G para determinar uma nova função de peso \hat{w} pode ser executado em tempo $O(nm)$.

Para um grafo orientado ponderado $G = (V, E)$ com a função de peso $w : E \rightarrow \mathcal{R}$, seja $h : V \rightarrow \mathcal{R}$ qualquer função que mapeia vértices para números reais. Para cada arco $(u, v) \in E$, definimos

$$\hat{w}(u, v) = w(u, v) + h(u) - h(v)$$

A função $\hat{w}(u, v)$ atende às duas propriedades postas anteriormente: consistência dos menores caminhos e não negatividade.

A prova de corretude será omitida, porém, pode ser encontrada no livro *Algoritmos: Teoria e Prática*.

Reponderação

Para a utilização da função definida anteriormente, um vértice artificial s , sem antecessores e adjacente a todos os demais vértices usando arcos de peso zero, deve ser adicionado ao grafo.

A função $h(v)$ pode ser definida como o comprimento do caminho mais curto entre a origem s e o vértice v ($\forall v \in V$), usando os pesos determinados por w .

É importante observar que os caminhos mais curtos citados acima podem conter arcos de peso negativo, portanto, podem ser calculados pelo algoritmo de *Bellman-Ford*.

Algoritmo de *Johnson* para Grafos Esparsos

Terminologia

- ▶ w : pesos originais dos arcos do grafo;
- ▶ \hat{w} : pesos reponderados dos arcos do grafo;
- ▶ δ : comprimentos de caminhos mais curtos calculados usando w ;
- ▶ $\hat{\delta}$: comprimentos de caminhos mais curtos calculados usando \hat{w} ;
- ▶ $G' = (V', E')$: grafo criado a partir do grafo original, porém, com o acréscimo do vértice s e respectivos arcos de peso zero;
- ▶ D : matriz $n \times n$ utilizada para armazenar os caminhos mais curtos.

Algoritmo de *Johnson* para Grafos Esparsos

Entrada: Grafo $G = (V, E)$

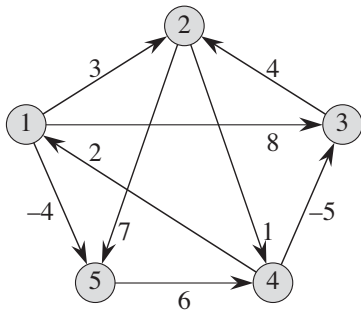
```
1  $G' \leftarrow G$ ;  
2  $V' \leftarrow V' \cup \{s\}$ ;  
3  $E' \leftarrow E' \cup (s, v) \forall v \in V$ ;  
4  $w(s, v) \leftarrow 0 \forall v \in V$ ;  
5 se  $Bellman-Ford(G', w, s) = \text{FALSO}$  então  
6 |   Imprima "Contém ciclo negativo!";  
7 senão  
8 |   para cada vértice  $v \in V'$  faça  
9 |   |    $h(v) \leftarrow \delta(s, v)$ ; //calculado por Bellman-Ford  
10 |  fim  
11 fim  
12 para cada arco  $(u, v) \in E'$  faça  
13 |    $\hat{w}(u, v) \leftarrow w(u, v) + h(u) - h(v)$ ;  
14 fim  
15 Crie uma nova matriz  $D$   $n \times n$ ;  
16 para cada vértice  $u \in V$  faça  
17 |    $Dijkstra(G, \hat{w}, u)$ ; //calcula  $\hat{\delta}(u, v) \forall v \in V$   
18 |   para cada vértice  $v \in V$  faça  
19 |   |    $d_{uv} \leftarrow \hat{\delta}(u, v) + h(v) - h(u)$ ;  
20 |  fim  
21 fim  
22 retorna  $D$ ;
```

Algoritmo de *Johnson* para Grafos Esparsos

Complexidade

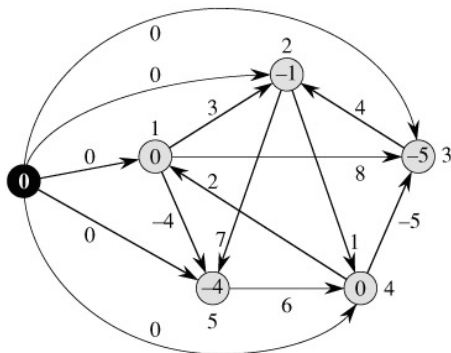
Se implementarmos a fila de prioridade no algoritmo de *Dijkstra* por um *heap* de *Fibonacci*, o algoritmo de Johnson é executado em tempo $O(n^2 \lg n + nm)$, que é assintoticamente mais rápido que o algoritmo de Floyd-Warshall, $\Theta(n^3)$, em grafos esparsos.

Exemplo



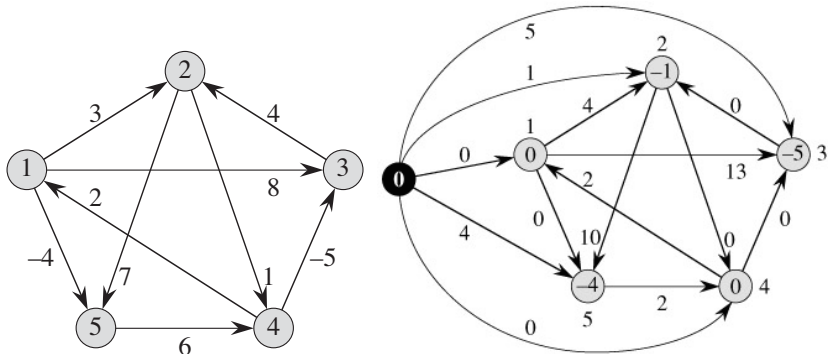
Grafo original.

Exemplo



Grafo G' com a função peso w original. O novo vértice s é destacado em preto.
O rótulo de cada vértice v é $h(v) = \delta(s, v)$.

Exemplo



Cada arco (u, v) é reponderado com a função peso $\hat{w}(u, v) = w(u, v) + h(u) - h(v)$.

Algoritmo de *Johnson* para Grafos Esparsos

Exemplo

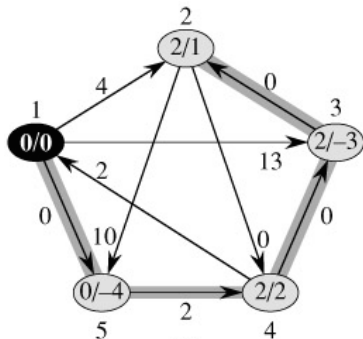
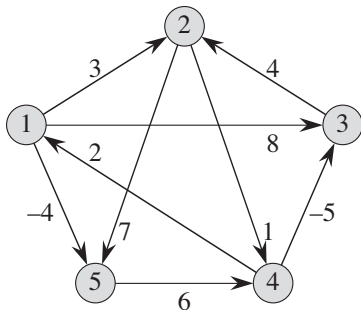
Os slides a seguir ilustram a execução do algoritmo de *Dijkstra* em cada vértice de G , usando a função de peso \hat{w} .

Em cada figura, o vértice de origem é o destacado em preto, e os arcos sombreados estão na árvore de caminhos mais curtos calculada pelo algoritmo.

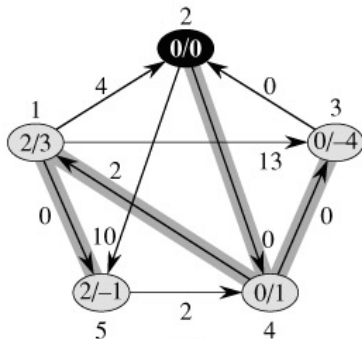
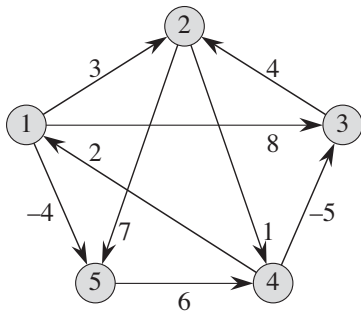
Dentro de cada vértice v estão indicados os valores $\hat{\delta}(u, v)$ e $\delta(u, v)$, separados por uma barra.

O valor $d_{uv} = \delta(u, v)$ é igual a $\hat{\delta}(u, v) + h(v) - h(u)$.

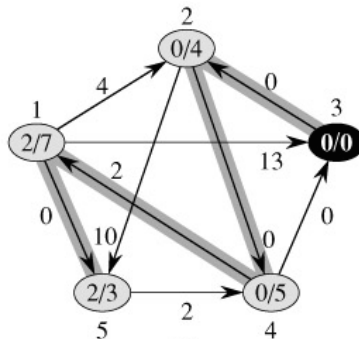
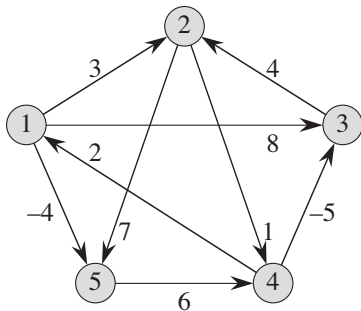
Exemplo



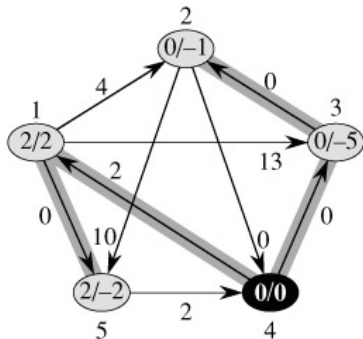
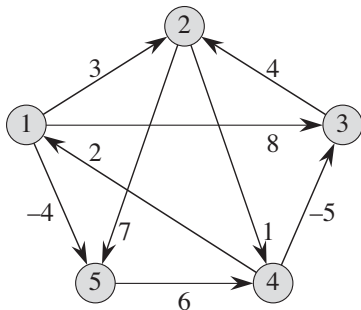
Exemplo



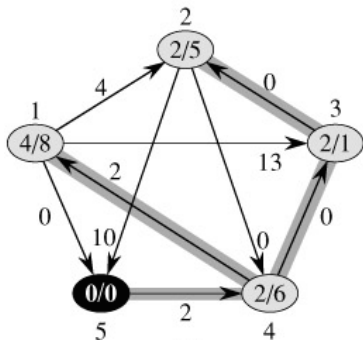
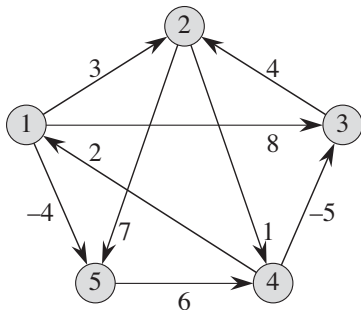
Exemplo



Exemplo



Exemplo



Dúvidas?

