PCC173/BCC463 - Otimização em Redes

Marco Antonio M. Carvalho

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

Conteúdo

Aviso

Fonte

Este material é baseado nos livros

- Goldbarg, M., & Goldbarg, E. (2012). *Grafos: conceitos, algoritmos e aplicações*. Elsevier.
- Goldbarg, M. C., & Luna, H. P. L. (2005). *Otimização combinatória e programação linear: modelos e algoritmos*. Elsevier.

Licença

Arestas de Peso Negativo

Para além das distâncias geográficas, caminhos mais curtos podem modelar diversas outras situações reais, incluindo aquelas que para serem modeladas necessitam de arestas cujo peso é negativo:

- Movimentações financeiras, nas quais é possível obter lucro ou prejuízo, principalmente quando há utilização de câmbio;
- Um taxista que recebe mais dinheiro do que gasta com combustível a cada viagem: se o táxi roda vazio, ele gasta mais do que recebe;
- Um entregador que necessita atravessar um pedágio e pode acabar pagando mais do que recebe para entregar encomendas;
- A energia gerada e consumida durante uma reação química.

Algoritmo de *Bellman-Ford*

Shimbel ou *Ford-Moore-Bellman*?

Há registros históricos de que este algoritmo foi proposto por *Shimbel*[a] em 1955 (que não levou o crédito, vide a *Lei de Eponímia de Stigler*[b]).

Alguns autores denominam o algoritmo de *Ford-Moore-Bellman*, em homenagem a outros três autores que propuseram o mesmo algoritmo em anos diferentes:

- Lester Ford (1956);
- Edward Moore (1957);
- Richard Bellman (1958).

---

[a] Shimbel, A. (1955). Structure in communication nets. Proceedings of the Symposium on Information Networks. New York, NY: Polytechnic Press of the Polytechnic Institute of Brooklyn. pp. 199–203.

[b] "Nenhuma descoberta científica recebe o nome de seu descobridor" – Robert K. Mentor

Algoritmo de *Bellman-Ford* - SSSPP

Princípio

Calcula caminhos mais curtos (SSSPP) via programação dinâmica *bottom-up*.

Ao invés de fechar um vértice por iteração, como o algoritmo de Dijkstra, examina todos os vértices de um grafo orientado por iteração até que atualizações não sejam possíveis.

Em um grafo com n vértices, qualquer caminho possui no máximo $n-1$ arestas, portanto, cada vértice é examinado no máximo $n-1$ vezes.

Com esta estratégia, é possível calcular caminhos mínimos em grafos com arestas de peso negativo.

Assim como o algoritmo de *Dijkstra*, baseia-se no princípio de relaxação: uma aproximação da distância da origem até cada vértice é gradualmente atualizada por valores mais precisos até que a solução ótima seja atingida.

Algoritmo de *Bellman-Ford* - SSSPP

Princípio

"Um caminho de i para j com k+1 arestas pode ser obtido a partir de um caminho de i para j com k arestas".

Se, em alguma iteração do algoritmo os caminhos até cada um dos vértices permanecerem inalterados, não haverá atualizações nas próximas iterações e o algoritmo pode terminar.

Entretanto, se houver atualizações na última iteração do algoritmo, é sinal de que há pelo menos um ciclo negativo no grafo, dado que algum caminho terá n arestas ou mais.

Algoritmo de *Bellman-Ford* - SSSPP

Terminologia

- $\Gamma^{-}(i)$: Conjunto de vértices antecessores do vértice atual;
- $dt[i]$: Vetor que armazena a distância entre o vértice de origem e o vértice i;
- $rot[i]$: Vetor que armazena o índice do vértice anterior ao vértice i, no caminho cuja distância está armazenada em $dt[i]$;
- $altera$: variável *booleana* que indica se houve alguma atualização na iteração atual.

Algoritmo de Bellman-Ford - SSSPP

Entrada: Grafo $G = (V, E)$ e matriz de pesos D=$\{d_{ij}\}$ para todos os arcos (i, j)

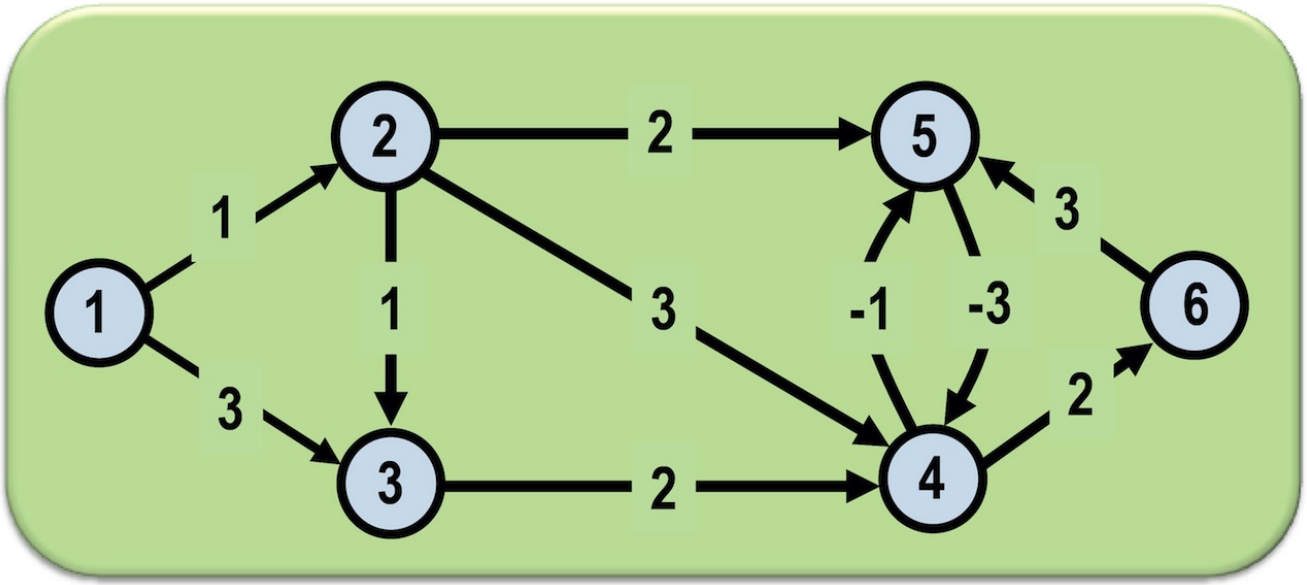
```
1  dt[1] ← 0; rot[1] ← ∞; //considerando o vértice 1 como o inicial
2  para i ← 2 até n faça
3  |   se ∃ (1, i) ∈ E então rot[i] ← 1; dt[i] ← d_1i;
4  |   senão rot[i] ← 0; dt[i] ← ∞;
5  fim
6  para k ← 1 até n-1 faça
7  |   altera ← falso;
8  |   para i ← 2 até n faça
9  |   |   para j ∈ Γ⁻(i) faça
10 |   |   |   se dt[i] > dt[j]+d_ji então
11 |   |   |   |   dt[i] ← dt[j]+d_ji;
12 |   |   |   |   rot[i] ← j;
13 |   |   |   |   altera ← verdadeiro; //indica que houve alteração
14 |   |   |   fim
15 |   |   fim
16 |   fim
17 |   se altera = falso então k ← n ;
18 fim
```

Ciclos de Custo Negativo

Bellman-Ford – Detecção

Em caminhos sem ciclos, o caminho mais longo consiste em $n - 1$ arestas, ou iterações no laço principal do algoritmo.

Se na iteração n do algoritmo alguma atualização de distâncias for feita é detectado o ciclo.



Exemplo de ciclo negativo entre os vértices 4 e 5.

Algoritmo de Bellman-Ford - SSSPP

Complexidade 1

- Após a inicialização, o laço **para** da linha 7 é repetido por no máximo n-1 vezes;
- Em cada iteração, são calculados caminhos com k arestas entre a origem e os demais vértices do grafo;
- Para cada um dos n-1 vértices, todos seus antecessores são examinados;
- O vértice original não é atualizado, logo, n-2 antecessores são analisados no máximo;
- Logo, em uma implementação simples, a complexidade é $O(n^3)$.

Complexidade 2

Em 1970, Jin Yen[a] propôs uma implementação deste método de complexidade $O(nm)$ no pior caso:

1. Se dt[v] não se alterar desde a última vez em que Γ^+ foi examinado, então não é necessário examinar seus arcos de saída novamente;
2. O comprimento do laço externo é reduzido de $n-1$ para $n/2$ através de uma ordenação linear de vértices e posterior partição dos mesmos.

---

[a] Yen, Jin Y. (1970). "An algorithm for finding shortest routes from all source nodes to a given destination in general networks". Quarterly of Applied Mathematics 27: 526–530.

Algoritmo de Bellman-Ford - SSSPP

Complexidade 3

Em 2012, Bannister e Eppstein[a] propuseram a substituição da ordenação linear por uma permutação aleatória, reduzindo para $n/3$ o comprimento do laço externo.

---

[a] Bannister, M. J.; Eppstein, D. (2012). Randomized speedup of the Bellman–Ford algorithm. Analytic Algorithmics and Combinatorics (ANALCO12), Kyoto, Japan. pp. 41–47.

Algoritmo de Bellman-Ford - SSSPP

Aplicação

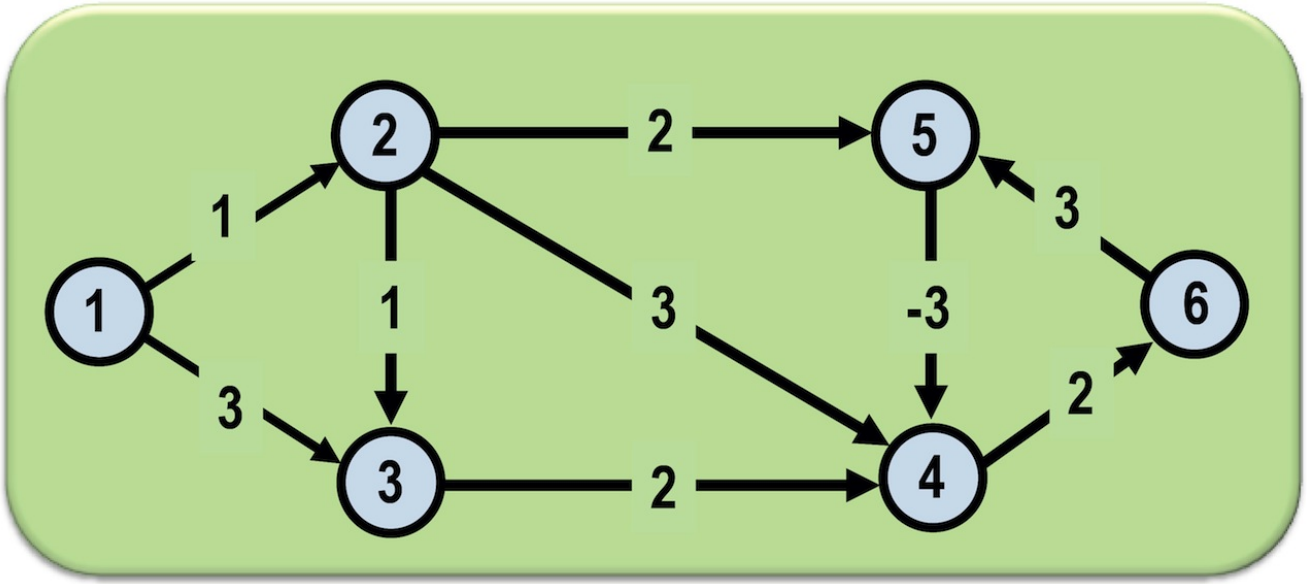
Uma das aplicações do algoritmo é em protocolos de roteamento em redes de dados.

O algoritmo é distribuído entre os nós da rede, de maneira que cada nó calcula sua distância em relação aos demais, compartilhando seu resultado para uso pelos outros nós.

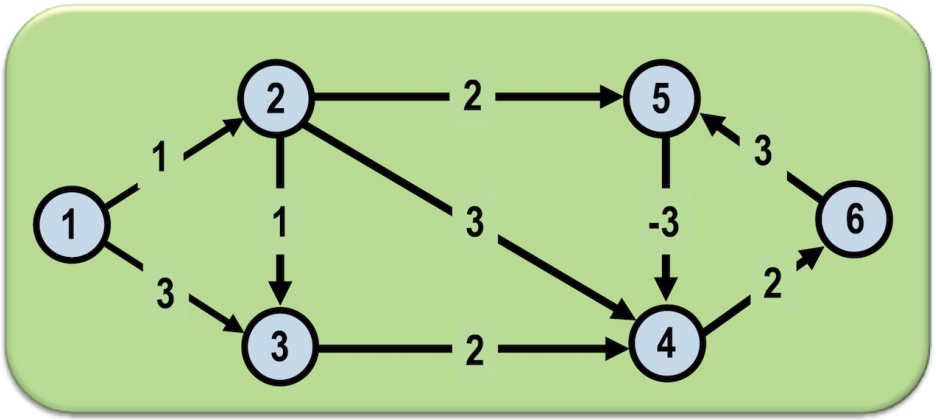
Entretanto, o algoritmo possui dificuldades relacionadas a escalabilidade e tolerância a falhas em nós da rede.

Além disso, quaisquer modificações na topologia da rede demoram a serem refletidas pelo algoritmo, dado que a atualização das distâncias é gradual.

Exemplo



Exemplo



dt

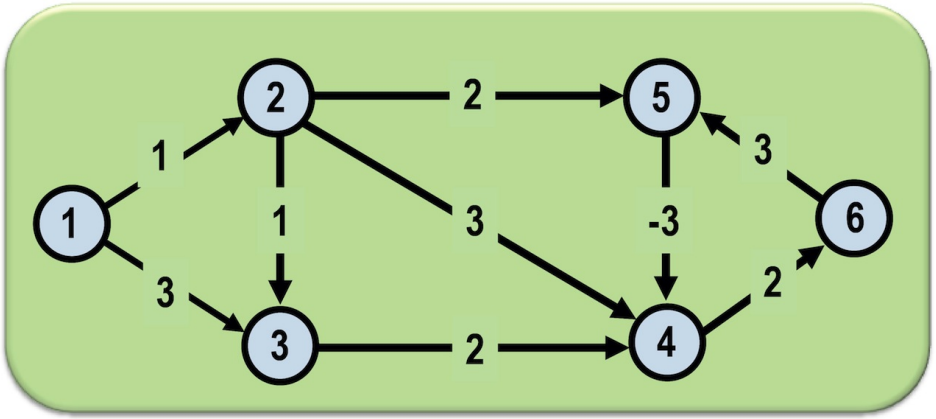
2	3	4	5	6
1	3	∞	∞	∞

rot

2	3	4	5	6
1	1	0	0	0

Vetores após a inicialização do algoritmo.

Exemplo



i=2, N={1};

- j=1, $dt[1]+d_{12} = 1$

dt

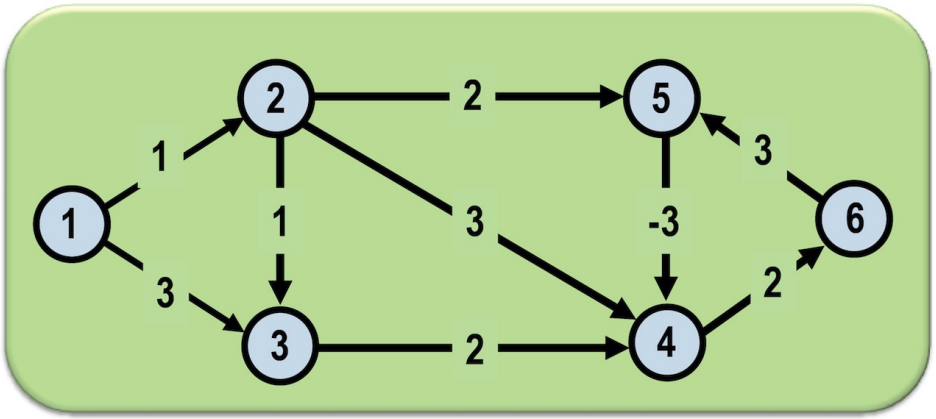
2	3	4	5	6
1	3	∞	∞	∞

rot

2	3	4	5	6
1	1	0	0	0

Iteração k=1 (continua...)

Exemplo



i=3, N={1, 2};

- j=1, $dt[1]+d_{13} = 3$
- j=2, $dt[2]+d_{23} = 2$

dt

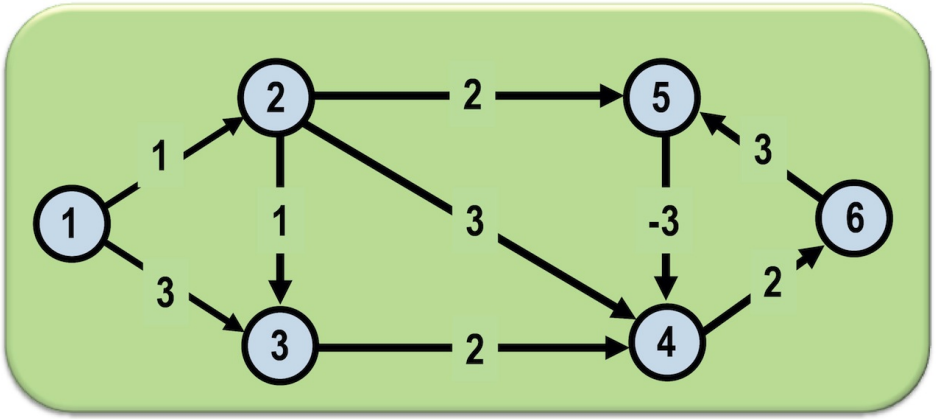
2	3	4	5	6
1	2	∞	∞	∞

rot

2	3	4	5	6
1	2	0	0	0

Iteração k=1 (continua...)

Exemplo



i=4, N={2, 3, 5};

- j=2, $dt[2]+d_{24} = 4$
- j=3, $dt[3]+d_{34} = 4$
- j=5, $dt[5]+d_{54} = \infty$

dt

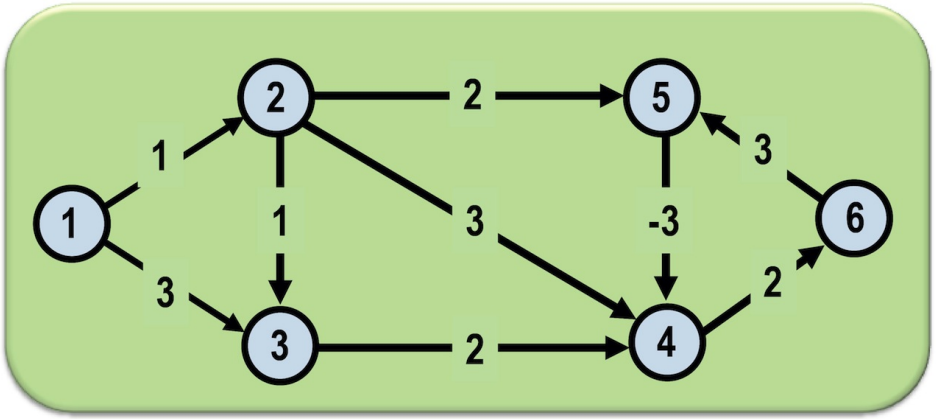
2	3	4	5	6
1	2	4	∞	∞

rot

2	3	4	5	6
1	2	2	0	0

Iteração k=1 (continua...)

Exemplo



i=5, N={2, 6};

- j=2, $dt[2]+d_{25} = 3$
- j=6, $dt[6]+d_{65} = \infty$

dt

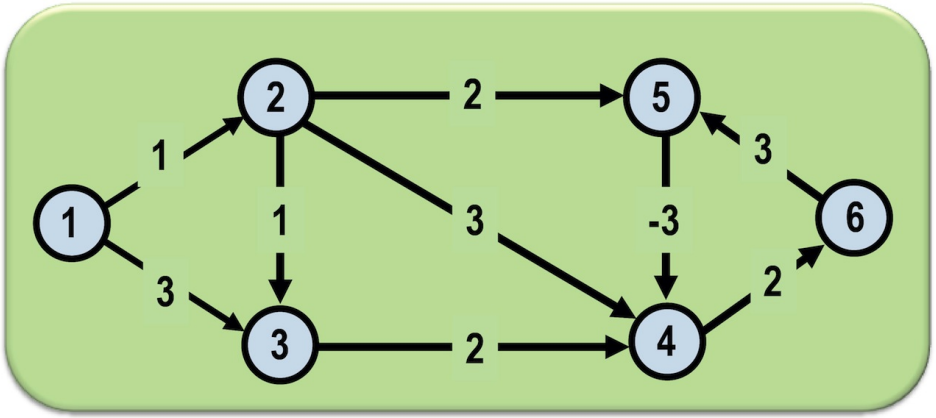
2	3	4	5	6
1	2	4	3	∞

rot

2	3	4	5	6
1	2	2	2	0

Iteração k=1 (continua...)

Exemplo



i=6, N={4};

- j=4, $dt[4]+d_{46} = 6$

dt

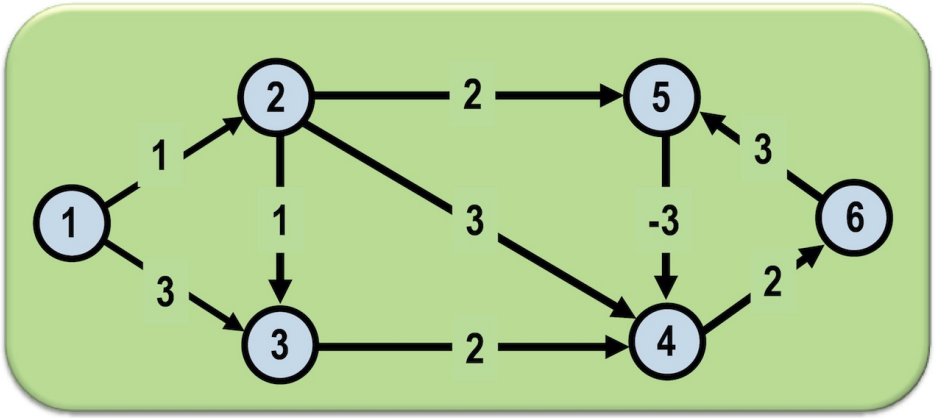
2	3	4	5	6
1	2	4	3	6

rot

2	3	4	5	6
1	2	2	2	4

Iteração k=1 (final)

Exemplo



i=2, N={1};

- j=1, $dt[1]+d_{12} = 1$

dt

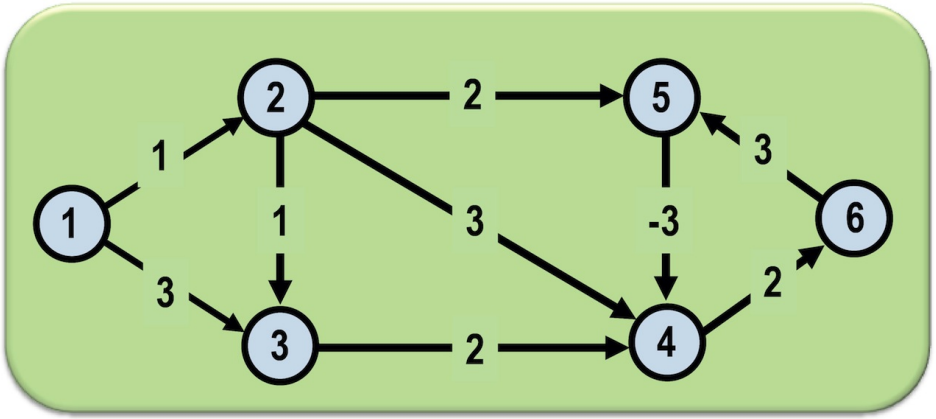
2	3	4	5	6
1	2	4	3	6

rot

2	3	4	5	6
1	2	2	2	4

Iteração k=2 (continua...)

Exemplo



i=3, N={1, 2};

- j=1, $dt[1]+d_{13} = 3$
- j=2, $dt[2]+d_{23} = 2$

dt

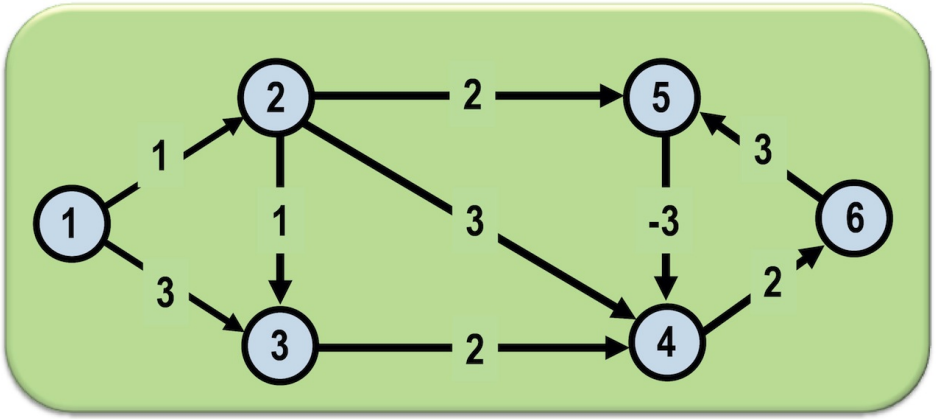
2	3	4	5	6
1	2	4	3	6

rot

2	3	4	5	6
1	2	2	2	4

Iteração k=2 (continua...)

Exemplo



i=4, N={2, 3, 5};

- j=2, $dt[2]+d_{24} = 4$
- j=3, $dt[3]+d_{34} = 4$
- j=5, $dt[5]+d_{54} = 0$

dt

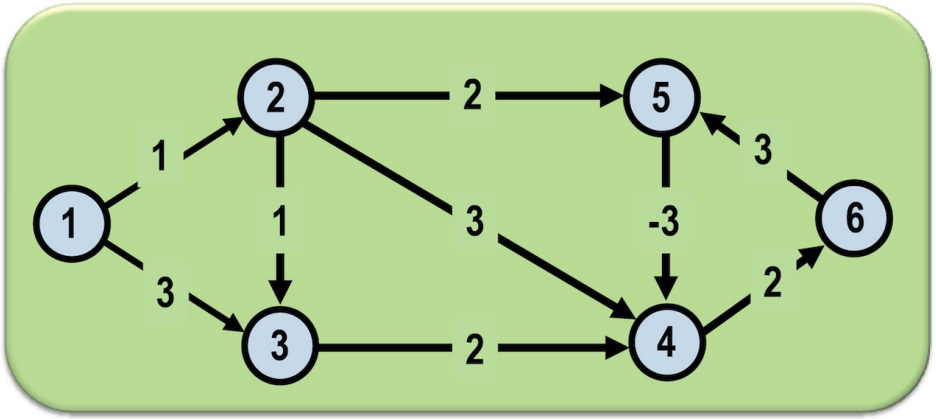
2	3	4	5	6
1	2	0	3	6

rot

2	3	4	5	6
1	2	5	2	4

Iteração k=2 (continua...)

Exemplo



i=5, N={2, 6};

- j=2, $dt[2]+d_{25} = 3$
- j=6, $dt[6]+d_{65} = 9$

dt

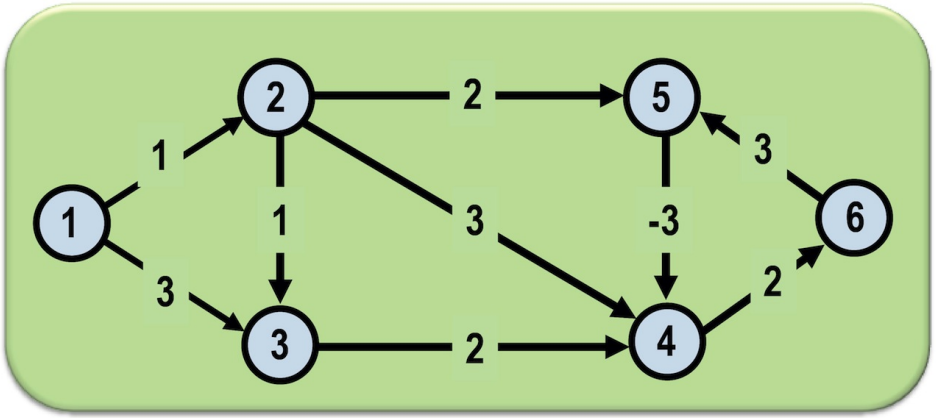
2	**3**	**4**	**5**	**6**
1	2	0	3	6

rot

2	**3**	**4**	**5**	**6**
1	2	5	2	4

Iteração k=2 (continua...)

Exemplo



i=6, N={4};

- j=4, $dt[4]+d_{46} = 2$

dt

2	3	4	5	6
1	2	0	3	2

rot

2	3	4	5	6
1	2	5	2	4

Iteração k=2 (final)

Exemplo

Final

Na próxima iteração, em que k=3, nenhuma alteração é realizada, e com isto, o algoritmo termina!

Dúvidas?