BCC204 - Teoria dos Grafos

Marco Antonio M. Carvalho

(baseado nas notas de aula do prof. Haroldo Gambini Santos)
Departamento de Computação
Instituto de Ciências Exatas e Biológicas
Universidade Federal de Ouro Preto

2 de setembro de 2019

UFOP
Universidade Federal
de Ouro Preto

decom
departamento
de computação

Avisos

Verifiquem a visualização de algoritmos!

- `https://visualgo.net/`

Site da disciplina:

- `http://www.decom.ufop.br/marco/`

Lista de e-mails:

- `pcc173@googlegroups.com`

Para solicitar acesso:

- `http://groups.google.com/group/pcc173`

Conteúdo

Avisos

Verifiquem a visualização de algoritmos!

- https://visualgo.net/

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.

Ford-Moore-Bellman?

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).

Algoritmo de *Bellman-Ford*

Princípio

Calcula caminhos mais curtos 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.

Princípio

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.

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

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

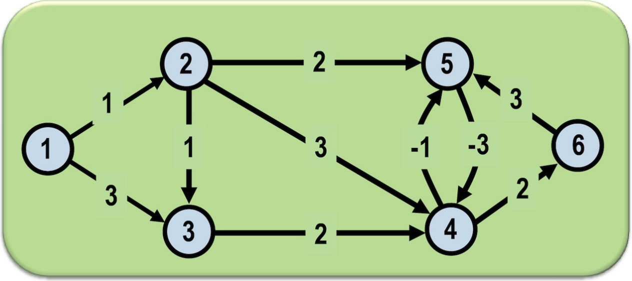
```
dt[1] ← 0; rot[1] ← ∞; //considerando o vértice 1 como o inicial
para i ← 2 até n faça
    se ∃ (1, i) ∈ E então rot[i] ← 1; dt[i] ← d_1i;
    senão rot[i] ← 0; dt[i] ← ∞;
fim
para k ← 1 até n-1 faça
    altera ← falso;
    para i ← 2 até n faça
        para j ∈ Γ⁻(i) faça
            se dt[i] > dt[j]+d_ji então
                dt[i] ← dt[j]+d_ji;
                rot[i] ← j;
                altera ← verdadeiro; //indica que houve alteração
            fim
        fim
    fim
    se altera = falso então  k ← n ;
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

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 $O(n^3)$.

Algoritmo de Bellman-Ford

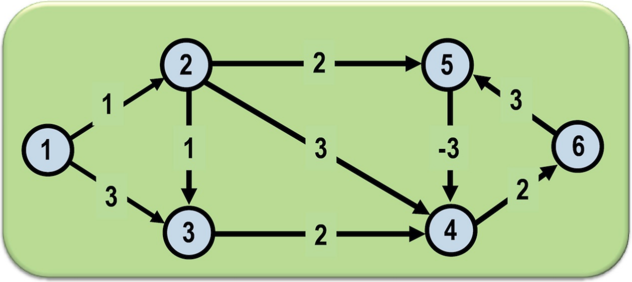
Complexidade 2

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

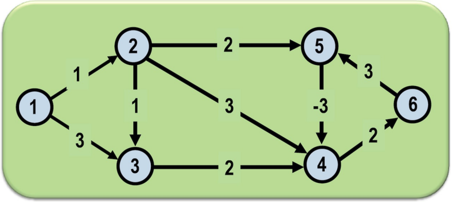
1. Se dt$[v]$ não se alterar desde a última vez em que Γ^+ foi examinado, então não é necessário examinar seus arcos de saída novamente;
2. O comprimento do laço externo é reduzido de $n-1$ para $n/2$ através de uma ordenação linear de vértices e posterior partição dos mesmos.

[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.

Exemplo



Exemplo



dt

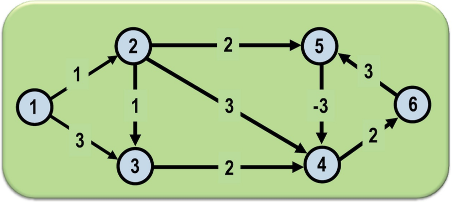
2	3	4	5	6
1	3	∞	∞	∞

rot

2	3	4	5	6
1	1	0	0	0

Vetores após a inicialização do algoritmo.

Exemplo



i=2, $\Gamma^-(i)$={1};

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

dt

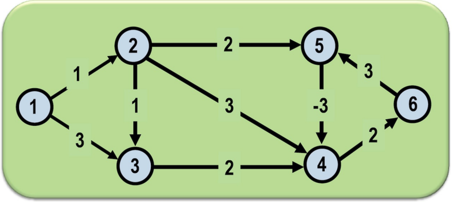
2	3	4	5	6
1	3	∞	∞	∞

rot

2	3	4	5	6
1	1	0	0	0

Iteração k=1 (continua...)

Exemplo



i=3, $\Gamma^{-}(i)$={1, 2};

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

dt

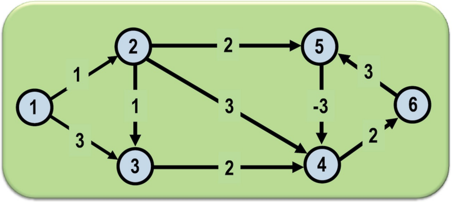
2	3	4	5	6
1	2	∞	∞	∞

rot

2	3	4	5	6
1	2	0	0	0

Iteração k=1 (continua...)

Exemplo



i=4, $\Gamma^{-}(i)$={2, 3, 5};

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

dt

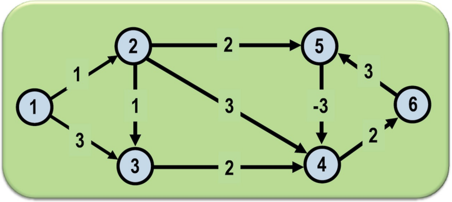
2	3	4	5	6
1	2	4	∞	∞

rot

2	3	4	5	6
1	2	2	0	0

Iteração k=1 (continua...)

Exemplo



i=5, $\Gamma^-(i)$={2, 6};

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

dt

2	3	4	5	6
1	2	4	3	∞

rot

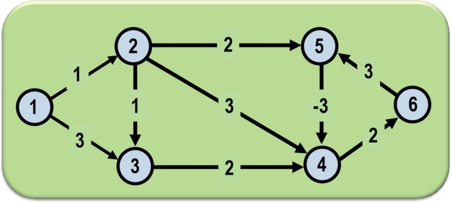
2	3	4	5	6
1	2	2	2	0

Iteração k=1 (continua...)

Exemplo

i=6, $\Gamma^-(i)$={4};

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



dt

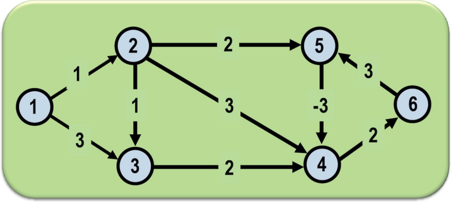
2	3	4	5	6
1	2	4	3	6

rot

2	3	4	5	6
1	2	2	2	4

Iteração k=1 (final)

Exemplo



i=2, $\Gamma^-(i)$={1};

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

dt

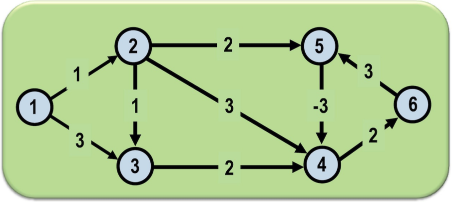
2	3	4	5	6
1	2	4	3	6

rot

2	3	4	5	6
1	2	2	2	4

Iteração k=2 (continua...)

Exemplo



i=3, $\Gamma^-(i)$={1, 2};

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

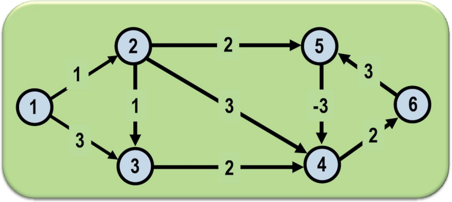
dt

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...)



i=4, $\Gamma^{-}(i)$={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

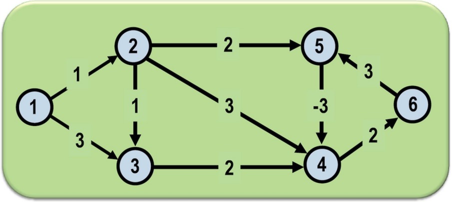
2	3	4	5	6
1	2	0	3	6

rot

2	3	4	5	6
1	2	5	2	4

Iteração k=2 (continua...)

Exemplo



i=5, $\Gamma^-(i)$={2, 6};

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

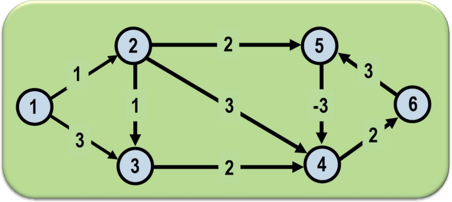
dt

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...)



i=6, $\Gamma^{-}(i)$={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)

Final

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

Aplicação

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

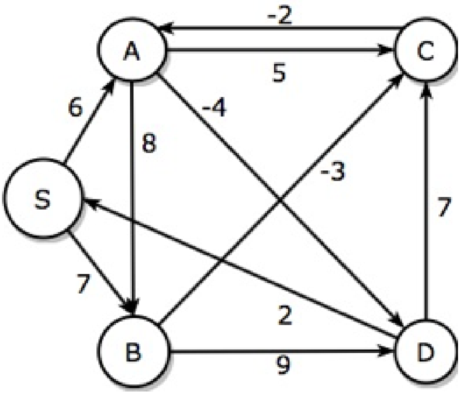
O algoritmo é distribuído entre os nós da rede, de maneira que cada nó calcula sua distância em relação aos demais, compartilhando seu resultado para uso pelos outros nós.

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.

Exercício

Execute o algoritmo de Bellman-Ford para o grafo abaixo.



Dúvidas?