GRAFOS – MENOR CAMINHO

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil

Agenda

1 Algoritmo de Dijkstra

2 Algoritmo de Floyd-Warshall

3 Algoritmo de Bellman-Ford

4 Bibliografia

Algoritmos gulosos

Algoritmos gulosos: solução construída a partir de uma sequência de passos, cada passo expandindo uma solução obtida previamente

Em cada passo, a nova solução deve ser:

- Viável: satisfaz as restrições do problema
- Localmente ótima: a melhor escola dentre as opções
- Irrevogável: não pode ser alterada em passos seguintes

Algoritmo de Dijkstra

Seja G um grafo ponderado, dado o nó $v \in V$ (*source*), encontrar o menor caminho de v para todos os outros vértices de G

- Single-source shortest paths

Aplicações

- Planejamento de transporte
- Roteamento de pacotes
- Redes sociais
- Reconhecimento de voz
- Robótica
- etc.

Algoritmo de Dijkstra: grafos (não-)dirigidos com pesos não-negativos

Algoritmo de Dijkstra

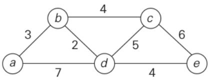
Primeiro, encontra o vértice mais próximo de v (origem)

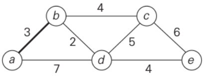
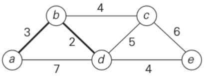
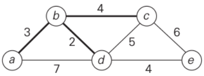
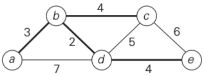
Depois, encontra o segundo vértice mais próximo de v

Na i-ésima iteração

- Conhece $i - 1$ vértices mais próximos de v
- Os $i - 1$ vértices, v e arestas selecionadas formam uma árvore T_i
- Como não há pesos negativos, o próximo vértice mais próximo de v é um adjacente aos vértices de T_i
- Escolhe um vértice v' a ser incorporado à T_i a partir do menor peso e, em seguida, atualiza o menor caminho dos vértices adjacentes à v' se $d_{v'} + w(v', u) < d_u$

Algoritmo de Dijkstra



Tree vertices	Remaining vertices	Illustration
a(−, 0)	**b(a, 3)** c(−, ∞) d(a, 7) e(−, ∞)	
b(a, 3)	c(b, 3 + 4) **d(b, 3 + 2)** e(−, ∞)	
d(b, 5)	**c(b, 7)** e(d, 5 + 4)	
c(b, 7)	**e(d, 9)**	
e(d, 9)		

[1]

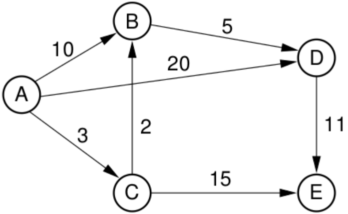
[1] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Algoritmo de Dijkstra

Algoritmo: void Dijkstra(Graph G, int s, int[] D)

```
H[0] ← (s, 0) ;                                   // heap para arestas
for i ← 0 to n(G) − 1 do
    D[i] ← ∞;
    setMark(G, i, UNVISITED);
D[s] ← 0;
for i ← 0 to n(G) − 1 do
    repeat
        v ← vertex(removemin(H));
    until getMark(G, v) = UNVISITED;
    setMark(G, v, VISITED);
    if D[v] = ∞ then return;
    w ← first(G, v);
    while w < n(G) do
        if getMark(G, w) ≠ VISITED ∧ D[w] > D[v] + weight(G, v, w) then
            D[w] ← D[v] + weight(G, v, w);
            insert(H, (w, D[w]));
        w ← next(G, v, w);
```

Algoritmo de Dijkstra[2]



	A	B	C	D	E
Initial	0	∞	∞	∞	∞
Process A	0	10	3	20	∞
Process C	0	5	3	20	18
Process B	0	5	3	10	18
Process D	0	5	3	10	18
Process E	0	5	3	10	18

Eficiência temporal

- Matriz s/ heap: $\Theta(\mid V \mid^2 + \mid E \mid) = \Theta(\mid V \mid^2)$, pois $\mid E \mid \in O(\mid V \mid^2)$
  - Melhor quando o grafo é denso
- Lista c/ heap: $\Theta((\mid V \mid + \mid E \mid) \; log \mid V \mid)$
  - Melhor quando o grafo é esparso

[2] Fonte: C. Shaffer. Data Structures and Algorithm Analysis. 2013.

Agenda

1 Algoritmo de Dijkstra

2 Algoritmo de Floyd-Warshall

3 Algoritmo de Bellman-Ford

4 Bibliografia

Algoritmo de Floyd

Seja G um grafo ponderado, encontrar o menor caminho entre todos os vértices de G

- All-pairs shortest paths

Floyd: grafos (não-)dirigidos sem ciclo de tamanho negativo

Baseado no algoritmo de Warshall para cálculo de fecho transitivo

Ideia geral:

- Vértice k como intermediário no menor caminho de i para j
- Para k, já consideramos $\{0, 1, ...k - 1\}$ como intermediários
- Dois casos possíveis
  - k não é um vértice intermediário
  - k é um vértice intermediário
    se o caminho for menor, atualiza matriz

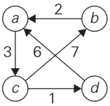
Algoritmo de Floyd

Algoritmo: void Floyd(Graph G, int[][] D)

```
for i ← to n(G) − 1 do
    for j ← to n(G) − 1 do
        if i = j then D[i][j] ← 0;
        else if weight(G, i, j) ≠ 0 then D[i][j] ← weight(G, i, j);
        else D[i][j] ← ∞;
for k ← to n(G) − 1 do
    for i ← to n(G) − 1 do
        for j ← to n(G) − 1 do
            if D[i][k] ≠ ∞ ∧ D[k][j] ≠ ∞ ∧
            D[i][j] > D[i][k] + D[k][j] then
                D[i][j] ← D[i][k] + D[k][j];
```

Eficiência temporal: $\Theta(|\ V\ |^3)$

Algoritmo de Floyd



$$D^{(0)} = \begin{array}{cc} & \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \end{array} & \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

$$D^{(1)} = \begin{array}{cc} & \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \end{array} & \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \mathbf{5} & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \mathbf{9} & 0 \end{bmatrix} \end{array}$$

$$D^{(2)} = \begin{array}{cc} & \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \end{array} & \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ \mathbf{9} & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix} \end{array}$$

$$D^{(3)} = \begin{array}{cc} & \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \end{array} & \begin{bmatrix} 0 & \mathbf{10} & 3 & \mathbf{4} \\ 2 & 0 & 5 & \mathbf{6} \\ 9 & 7 & 0 & 1 \\ 6 & \mathbf{16} & 9 & 0 \end{bmatrix} \end{array}$$

$$D^{(4)} = \begin{array}{cc} & \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \end{array} & \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ \mathbf{7} & 7 & 0 & 1 \\ 6 & 16 & 9 & 0 \end{bmatrix} \end{array}$$ [3]

[3] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Agenda

1 Algoritmo de Dijkstra

2 Algoritmo de Floyd-Warshall

3 Algoritmo de Bellman-Ford

4 Bibliografia

Algoritmo de Bellman-Ford

Seja G um grafo ponderado, dado o nó $v \in V$ (*source*), encontrar o menor caminho de v para todos os outros vértices de G

- Single-source shortest paths

Algoritmo de Bellman-Ford: grafos (não-)dirigidos podendo ter pesos negativos e ciclos de tamanhos negativos

Ideia geral:

- Iteração 1: menores caminhos com no máximo 1 aresta
- Iteração i: menores caminhos com no máximo i arestas
- Iteração |V|-1: menores caminhos com no máximo |V|-1 arestas

Por fim, itera mais uma vez:

- Se encontrar novo melhor caminho: ciclo negativo

Algoritmo de Bellman-Ford

Algoritmo: void BellmanFord(Graph G, int s, int[] D)

```
for i ← 0 to n(G) − 1 do
    D[i] ← ∞;
D[s] ← 0;
for k ← 0 to n(G) − 2 do
    for i ← 0 to n(G) − 1 do
        j ← first(G, i);
        while j < n(G) do
            if D[j] > D[i] + weight(G, i, j) then
                D[j] ← D[i] + weight(G, i, j);
            j ← next(G, i, j);
for i ← 0 to n(G) − 1 do
    j ← first(G, i);
    while j < n(G) do
        if D[j] > D[i] + weight(G, i, j) then
            negative cycle detected
        j ← next(G, i, j);
```

Algoritmo de Bellman-Ford

Parecido com o algoritmo de Dijkstra, mas não é guloso

Eficiência espacial: $\Theta(|V||E|) = \Theta(|V|^3)$, pois $|E| \in O(|V|^2)$

Agenda

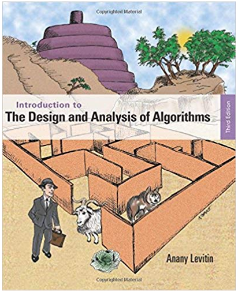
1 Algoritmo de Dijkstra

2 Algoritmo de Floyd-Warshall

3 Algoritmo de Bellman-Ford

4 Bibliografia

Bibliografia + leitura recomendada



Capítulo 9 (pp. 308–311)
Capítulo 9 (pp. 333–337)
Anany Levitin.
Introduction to the Design and Analysis of Algorithms.
3a edição. Pearson. 2011.

Capítulo 11 (pp. 389–393)
Capítulo 16 (pp. 513–515)
Clifford Shaffer.
Data Structures and Algorithm Analysis.
Dover, 2013.

GRAFOS – MENOR CAMINHO

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil