Caminho mais curto

Num grafo pesado, com pesos w, o peso do caminho

$$p = v_0\, v_1\, \ldots\, v_k$$

é a soma dos pesos dos arcos que o integram

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

O caminho p é mais curto que o caminho p' se o peso de p é menor que o peso de p'

Cálculo dos caminhos mais curtos

Algoritmos

Cálculo dos caminhos mais curtos num grafo orientado acíclico (DAG), com pesos possivelmente negativos

Algoritmo de Dijkstra, para grafos sem pesos negativos

Algoritmo de Bellman-Ford, para quaisquer grafos pesados

Estes algoritmos calculam os caminhos mais curtos de um nó s para os restantes nós do grafo (*single-source shortest paths*)

Caminhos mais curtos

Subrotinas comuns aos diversos algoritmos

O peso do caminho mais curto de s a qualquer outro nó é inicializado com ∞

INITIALIZE-SINGLE-SOURCE(G, s)

```
for each vertex v in G.V do
    v.d <- INFINITY   // peso do caminho mais curto de s a v
    v.p <- NIL        // predecessor de v nesse caminho
s.d <- 0
```

Se o caminho de s a v, que passa por u e pelo arco (u, v), tem menor peso do que o mais curto anteriormente encontrado, encontrámos um caminho mais curto

RELAX(u, v, w)

```
if u.d + w(u,v) < v.d then
    v.d <- u.d + w(u,v)
    v.p <- u
```

Caminhos mais curtos a partir de um vértice

Algoritmo para DAGs

$G = (V, E)$ – DAG pesado (pode ter pesos negativos)

DAG-SHORTEST-PATHS(G, w, s)

```
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, taken in topologically sorted order do
    for each vertex v in G.adj[u] do
        RELAX(u, v, w)
```

Caminhos mais curtos a partir de um vértice

Algoritmo de Dijkstra

$G = (V, E)$ – grafo pesado orientado (sem pesos negativos)

DIJKSTRA(G, w, s)

```
INITIALIZE-SINGLE-SOURCE(G, s)
S <- EMPTY
Q <- G.V                         // priority queue (key: u.d)
while Q != EMPTY do
    u <- EXTRACT-MIN(Q)
    S <- S + {u}
    for each vertex v in G.adj[u] do
        RELAX(u, v, w)       // may alter v.d key in Q
```

Quando é encontrado um novo caminho mais curto para um vértice (na função RELAX), é necessário reorganizar a fila Q (DECREASE-KEY)

Caminhos mais curtos a partir de um vértice

Algoritmo de Bellman-Ford

$G = (V, E)$ – grafo pesado orientado (pode ter pesos negativos)

BELLMAN-FORD(G, w, s)

```
INITIALIZE-SINGLE-SOURCE(G, s)
for i <- 1 to |G.V| - 1 do
    for each edge (u,v) in G.E do
        RELAX(u, v, w)
for each edge (u,v) in G.E do
    if u.d + w(u,v) < v.d then
        return FALSE
return TRUE
```

Complexidade dos algoritmos

$G = (V, E)$

	Compl. Temporal
Percurso em largura	$O(V + E)$
Percurso em profundidade	$O(V + E)$
Grafo transposto	$O(V + E)$
Cálculo das componentes fortemente conexas	$O(V + E)$
Ordenação topológica (ambos os algoritmos)	$O(V + E)$
Algoritmos de Prim e de Kruskal	$O(E \log V)$
Caminhos mais curtos num DAG	$O(V + E)$
Algoritmo de Dijkstra	$O(E \log V)$
Algoritmo de Bellman-Ford	$O(VE)$
Algoritmo de Floyd-Warshall	$O(V^3)$

Pressupostos

Grafo representado através de listas de adjacências (excepto algoritmos de Bellman-Ford e de Floyd-Warshall)

Algoritmos de Prim e de Dijkstra recorrem a uma fila tipo *heap* binário (EXTRACT-MIN e DECREASE-KEY com complexidade temporal logarítmica no número de elementos da fila)

Algoritmo de Kruskal usa Partição com compressão de caminho