Algoritmo de Dijkstra em LISP

Léo Willian Kölln

10 de Agosto de 2006

Curso de Ciências da Computação
Programação Funcional - INE5363

INE - Departamento de Informática e Estatística
CTC - Centro Tecnológico
UFSC - Universidade Federal de Santa Catarina

1 O Algoritmo de Dijkstra

O Algoritmo de Dijkstra, assim nomeado assim após sua descoberta pelo Cientista de Computação alemão Edsger Dijkstra. E um algoritmo que resolve o problema do caminho mais curto de única origem para um grafo orientado utilizando pesos não negativos de arestas.

Por exemplo, se os vértices de um grafo representam cidades e o peso das arestas representam as distâncias entre um par de cidades diretamente conectadas por estradas, o algoritmo de Dijkstra pode ser usado para encontrar a rota mais curta entre duas cidades.

A entrada do algoritmo consiste de um grafo G orientado e com pesos de arestas e um vértice de origem "s"que está em G. Nós iremos considerar V o conjunto de todos os vértices no grafo G. Cara aresta do grafo é um par ordenado de vértices (u,v) representando uma conexão do vértice u até o vértice v. O conjunto de todas as arestas será denotado por E. Os pesos das arestas serão dados por uma dada função de peso w: E -> [0, 8); significando que w(u,v) é um custo não negativo para se mover diretamente do vértice u até o vértice v. O custo de uma aresta pode ser pensado como (de maneira generalizada) a distância entre esses dois vértices. O custo de um caminho entre dois vértices é a soma dos custos de cada aresta neste caminho. Para um dado par de vértices s e t em v, o algoritmo encontra o caminho de s até t com o custo mais baixo (ex: o menos caminho). Também pode ser usado para encontrar custos de menor caminho de um único vértice s até todos os outros vértices no grafo.

1.1 Descrição do Algoritmo

O algoritmo funciona mantendo para cara vértice v o custo d[v] do caminho mais curto encontrado entre s e v. Inicialmente este valor é 0 para o vértice de origem s (d[s]=0), e infinito para todos os outros vértices, representando o fato de que não conhecemos nenhum caminho levando a estes vértices (d[v]=8 para cada v em V, exceto s). Quando o algoritmo termina, d[v] será o custo do caminho mais curto de s até v, ou infinito, se tal caminho não existir.

A operação básica do algoritmo de Dijkstra é uma relaxação de arestas. Se existe aresta de u até v, então o menor caminho conhecido de s até v (d[u]) pode ser estendido como o caminho de s até v adicionando uma aresta (u,v) ao seu final. Este caminho terá o tamanho d[u]+w(u,v). Se este é menor do que o atual d[v], podemos substituir o atual valor de d[v] pelo novo valor. A relaxação de arestas é aplicada até que todos os valores d[v] representem o custo do menor caminho de s até v. O algoritmo é organizado de modo que cada aresta (u,v) terá sido relaxada apenas uma vez, quando d[u] tiver alcançado seu valor final.

A noção de "relaxação"vem de uma analogia entre estimar o menor caminho e o cumprimento de uma "mola helicoidal de tensão", que não é susceptível a compressões. Inicialmente o caminho de menor custo é uma estimativa exagerada, assim como uma mola esticada. Quando o caminho mais curto é encontrado, o custo estimado é reduzido, e a mola é relaxada. Eventualmente, o caminho

mais curto, se existir, é encontrado e a mola estará relaxada a seu cumprimento normal.

O algoritmo mantém dois conjuntos de vértices, S e Q. O conjunto S contém todos os vértices para os quais sabemos os valores d[v] que já é o custo do menor caminho, e o conjunto Q contém todos os outros vértices. O conjunto S começa vazio, e em cada passo um vértice é movido de Q até S. Este vértice é escolhido como o vértice com o menor valor de d[u]. Quando um vértice u é movido para S, o algoritmo relaxa cada aresta de saída (u,v).

1.2 Pseudocódigo

```
function Dijkstra(G, w, s)
for each vertex v in V[G]
d[v] := infinity
previous[v] := undefined
d[s] := 0
S := empty set
Q := V[G]
while Q is not an empty set
u := Extract_Min(Q)
S := S union {u}
for each edge (u,v) outgoing from u
if d[u] + w(u,v) < d[v]
d[v] := d[u] + w(u,v)
previous[v] := u
```

1.3 Implementação do Algoritmo de Dijkstra em CLISP

```
; Retira da lista Q o vértice de menor valor na lista de estimativa
(defun retiraMenorValor (Q estimativa)
(setq verticeResultante nil)
; Ordenamos a lista em ordem crescente
(setq estimativaOrdenada (sort estimativa #'(lambda (a1 a2)(< (nth 1 a1) (nth 1 a2)))))
; Depois procuramos até encontrar o vértice que procuramos que pertence a Q
(dolist (vertice Q)
(setq verticeResultante (car (assoc vertice estimativaOrdenada)))
(when (not (eq verticeResultante nil)) (return))
)
; Agora retiramos de Q o vértice encontrado
(set 'Q (remove verticeResultante Q))
(let (resultado) verticeResultante)
)

; Retorna lista de todos as arestas em G que partem de u
(defun retornaSaidas(u G)
(setq arestasSaida '())
(dolist (aresta G)
; Quando encontrarmos uma aresta que sai de u, adicionamos a aresta na lista
(when (equal u (nth 0 aresta)) (push aresta arestasSaida))
```

```
)
(let (resultado) arestasSaida)
)
; Retorna o peso da aresta (u,v) em G
(defun retornaPesoAresta(u v G)
(setq peso nil)
(dolist (aresta G)
; Quando encontrarmos a primeira aresta (u,v) em G
;setamos peso com o valor do peso da aresta em G e retornamos
(when
(and
(equal (nth 0 aresta) u)
(equal (nth 1 aresta) v)
)
(setq peso (nth 2 aresta)) (return)
)
)
(let (resultado) peso)
)

; Retorna lista com do caminho de menor custo de s até w
;baseado na lista de precedentes Lp
(defun caminhoDeMenorCusto(s w Lp)
(setq u w) ; Vértice de destino
(setq caminhoResultante '())
(push u caminhoResultante) ; Colocamos o vértice inicial no final do caminho resultante
(loop
(when (or (equal u nil) (equal u s)) (return)) ; Fim do caminho
(setq u (cadr (assoc u Lp))) ; Seta u para seu precedente
(push u caminhoResultante)
)
(let (resultado) caminhoResultante)
)

; Retorna ((Lista das estimativas) (Lista dos precedentes))
(defun dijkstraCompletoLK (G s)
; Primeiro geramos o grafo para que os pesos das arestas fiquem ordenados (opcional)
(setq grafoOrdenado
(sort G #'(lambda (a1 a2)(< (nth 2 a1) (nth 2 a2))))
)
; Geramos a lista V com todos os vértices do grafo
(setq V '())
(dolist (aresta grafoOrdenado)
; Cada aresta (u,v) possui dois vértices, faremos o teste em todos
; Verifica se o primeiro vértice esta presente "u"
(setf V (adjoin (nth 0 aresta) V :test #'equal))
; Verifica se o segundo vértice esta presente "v"
(setf V (adjoin (nth 1 aresta) V :test #'equal))
)
```

```
; Criamos uma lista de associações para guardar as estimativas
;de custo de cada vértice e uma de precedentes
(setq estimativa '())
(setq precedente '())
(dolist (vertice V)
; Estimativas iniciais são infinitas
(push (list vertice most-positive-fixnum) estimativa)
; Os precedentes ficam indefinidos
(push (list vertice nil) precedente)
)
(setf (cadr (assoc s estimativa)) 0) ; Setamos o valor do vértice de origem como 0
(setq Q (copy-seq V)) ; Faremos uma cópia da lista de vértices do grafo para Q
(setq S '()) ; S será uma lista inicialmente vazia
; Iniciamos o Loop
(loop
(when (eq Q nil) (return)) ; Se Q for vazio, terminamos o loop
(setq u (retiraMenorValor Q estimativa))
(push u S)
(dolist (arestaSaida (retornaSaidas u grafoOrdenado))
(when
(<
(+
; Custo da aresta entre u e v
(retornaPesoAresta (nth 0 arestaSaida) (nth 1 arestaSaida) grafoOrdenado)
; Estimativa do vértice u
(cadr (assoc u estimativa))
) ; Soma o custo do caminho com a estimativa do u atual
(cadr (assoc (nth 1 arestaSaida) estimativa)) ; Estimativa do vértice de destino v
)
; Caso a soma for menor que a estimativa do vértice destino, relaxamos o vértice (u,v)
(setf
(cadr (assoc (nth 1 arestaSaida) estimativa))
(+
(retornaPesoAresta (nth 0 arestaSaida) (nth 1 arestaSaida) grafoOrdenado)
(cadr (assoc u estimativa))
)
)
(setf (cadr (assoc (nth 1 arestaSaida) precedente)) u)
)
)
)
(let (resultado) (list estimativa precedente))
)
```

1.3.1 Exemplo de Execução

Se executarmos as seguintes linhas junto com o algoritmo definido acima.

```
(setq
resultadoDijkstra
```

```
(dijkstraCompletoLK
'((s u 10) (s x 5) (u v 1) (u x 2) (x u 3) (x v 9) (x y 2) (v y 4) (y v 6) (y s 7))
's
)
)
(print "Caminhos de menor custo: ")
(prin1 (car resultadoDijkstra))
(print "Precedentes: ")
(prin1 (cdr resultadoDijkstra))
(print "Caminho de menor custo de s para v: ")
(prin1 (caminhoDeMenorCusto 's 'v (cadr resultadoDijkstra)))
```

Teremos o seguinte resultado.

```
"Caminhos de menor custo: " ((S 0) (X 5) (Y 7) (U 8) (V 9))
"Precedentes: " (((U X) (V U) (X S) (Y X) (S NIL)))
"Caminho de menor custo de s para v: " (S X U V)
```

Referências

[1] **Caminos de Peso Mínimo** - http://personales.unican.es/ruizvc/scheme/grafos.alg/dijkstra.pdf

[2] Antonio C. Mariani, **Algoritmo de Dijkstra para cálculo do Caminho de Custo Mínimo** - http://www.inf.ufsc.br/grafos/temas/custo-minimo/dijkstra.html

[3] Wikipedia, **Dijkstra's algorithm** - http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

[4] **Animation of Dijkstra's algorithm** - http://www.cs.sunysb.edu/ skiena/combinatorica/animations/dijkstra.html