Analisando o Algoritmo de Yen

Micael Andrade Dos Santos

[1]Departamento de Computação (DCOMP) – Universidade Federal de Sergipe (UFS)
Av. Marechal Rondon, s/n – Jardim Rosa Elze – CEP 49100-000
São Cristóvão – SE – Brazil

kaell_andrade@academico.ufs.br, micael.santos@dcomp.ufs.br

Resumo. *Este relatório tem como objetivo fazer uma análise simples do algoritmo de Yen para o cálculo dos K's caminhos mais curtos em um grafo ponderado com arestas contendo pesos positivos.*

1. Sobre o Algoritmo

O algoritmo foi publicado pelo matemático Jin Y. Yen [Yen 1971] para determinar a sequência dos *K-1* caminhos mais curtos de forma eficiente. Ou seja, dado um grafo *G(E,V)* ponderado e, seja $\{i, j\}$ dois vértices quais quer de *G*, será calculado *K-1* caminhos mínimos entre *i* e *j*.

2. Como o Algoritmo funciona

Podemos dividir o algoritmo em duas partes cruciais para um melhor entendimento. A primeira delas seria definir o primeiro caminho mais curto entre *i, j* nessa primeira parte pode-se usar o algoritmo de Bellman-Ford [Goldberg and Radzik 1993] ou Dijkstra [Dijkstra 1968]. Qualquer um desses dois algoritmos podem ser usados de forma eficiente para determinar o primeiro caminho mais curto *K(1)*.

A segunda parte determina os outros caminhos mais curtos. Precisamos de duas listas, *A* e *B*. Sendo que a lista *A* irá armazenar os caminhos mais curtos encontrados pelo algoritmo de *Yen* e a lista *B* conterá todos possíveis caminhos mais curtos, quando dizemos possíveis é porque nem todos os caminhos encontrados são mais curtos até aquele momento, ou seja, o algoritmo encontra um determinado caminho curto *K*, mas pode haver outro caminho *K+1* que seja menor que o caminho *K* encontrado anteriormente.

Para entendermos melhor o funcionamento do algoritmo vamos analizar o seguinte pseudocódigo:

```
1  def yen(Grafo, I, J, K):
2      #Determinando o menor caminho e armazenando em A;
3      A = [Dijkstra(Grafo, I, J)];
4      B = []; #Aqui será armazenado os possíveis caminhos mínimios
5
6      #Começa em 1 pois já determinamos um caminho na linha 3;
7      for k in range(1, K):
8          # O nó de estimulo varia do primeiro nó para o próximo
9          # nó no caminho anterior do K-caminho mais curto.
10         for i  range(len(A[k-1] - 2):
11             #Recupera o nó de estimulo
```

```
12              noEstimulo = A[k-1][i];
13              #Sequencia de nós de I até o noEstimulo;
14              caminhoRaiz = A[k-1][:i+1];
15              for caminho in A:
16                  if(caminhoRaiz == caminho[:i+1]):
17                      #Define a restas como infinito,
18                      #pois agora estamos interessado em outro caminho;
19                      setInfinito(G, caminho[i], caminho[i+1]);
20              caminhoEstimulo = Dijkstra(Grafo, noEstimulo, J);
21              caminhoTotal = caminhoRaiz + caminhoEstimulo;
22              #Adiciona um possivel caminho min
23              if(caminhoTotal not in B):
24                  B.append(cainhoTotal);
25              #Precisamos recuperar todos os pesos das arestas
26              Grafo.restaurarTodasArestas();
27              if( len(B) == 0):
28                  break; #Não há mais possíveis caminhos;
29              B.sort(); #Ordena os potenciais caminhos mínimos;
30              #Adiciona um caminho minímo em A e retira de B
31              A.(B.pop(0));
32
33      return A;
```

A função *yen* recebe um *grafo* e dois vértices *i* e *j*, além disso temos um terceiro parâmetro *K* que define quantos caminhos mínimos estamos interessados. Na linha *3* será feito o cálculo do menor caminho com o algoritmo de Dijkstra, a lista *B* será responsável por armazenar todos os possíveis caminhos mínimos.

Como já calculamos o primeiro caminho mínimo, o laço da linha 7 garante que será calculado o *K-1* caminho mínimo. O laço da linha 10 nos permite utilizar os caminhos mínimos já calculado para determinar outros possíveis caminhos mínimos, para isso precisamos de um nó de estimulo que esteja incluso no primeiro caminho mínimos calculado como mostra a linha 12.

Utilizamos um lista na linha 14 para armazenar um caminhoRaiz que varia do primeiro nó do caminho raiz até o penúltimo nó contido no caminho mais curto.

Além disso, como estamos calculando *K's* caminhos mais curtos precisamso garantir de alguma forma que o algoritmo não leve em considerações caminhos que contenham a mesmas sequência de arestas. Para isso *setInfinito* na linha 19 garante que um novo caminho mínimo será calculado para cada iteração do algoritmo.

Dessa forma, na linha 20 será calculado um novo caminho mínimo agora utilizando o no de Estimulo, em seguida, caminhoTotal conterá um novo caminho mínimo calculado com base nos outros caminhos já descobertos, caso esse novo caminho não esteja na lista *B* o mesmo será adicionado.

Agora precisamos resetar os pesos de nossas arestas para a detecção de outros possíveis caminhos. A linha 27 garante a parada do algoritmo quando não encontramos possíveis caminhos ou quando todos os possíveis caminhos foram calculados.

Na linha 29 irá ordenar todos os possíveis caminhos mínimos que posteriormente será adcinada em *A*, para questão de eficiência poderiamos usar um Heap para armazenar os caminhos em *B* ao invés de cada iteração *K* precisar ordenar.

3. Exemplos

Vamos agora examinar três exemplos usados na função *Yen*. A figura 1 mostra como podemos usar o algoritmo para determinar $K = 3$ caminhos mínimos de C para H. O primeiro caminho mínimo encontrado será o caminho $([C - E - F - H], 5)$, após encontrarmos esse caminho precisamos garantir que ele não será mais encontrado pelo algoritmo, sendo assim cada aresta pertecente a ele será setado como infinita. Assim o caminho encontrado seria $([C - D - F - H], 8)$ pois a arestas *(C,E)* foi setada como infinita. Após todos os pesos das aresas serem resetadas o terceiro caminho encontrado seria $([C - E - G - H], 7)$.

Após B ser ordenado o vetor A irá conter os seguintes caminhos mais curtos $[[C, E, F, H], [C, E, G, H], [C, D, F, H]]$. Como pode ser notado a figura 2 e 3 segue a mesma lógica utilizada na figura 1.

Figura 1. Yen(C, H, 3)

A = [C, E, F, H], [C, E, G, H], [C, D, F, H]

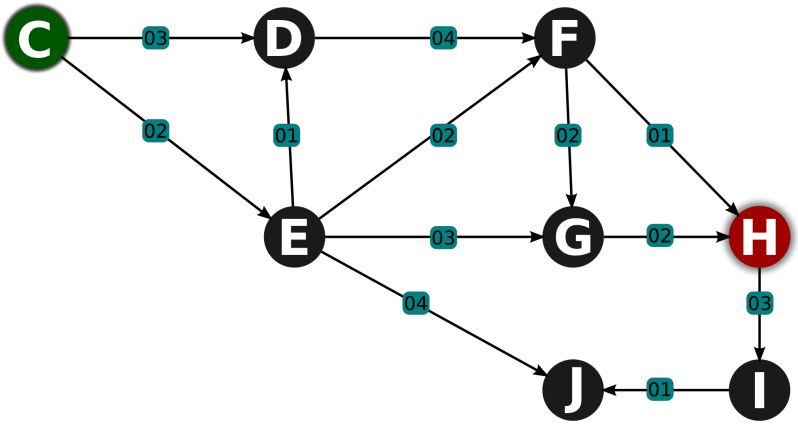


Figura 2. Yen(Aracaju, Jurubeba, 3)

A = [’Aracaju’, ’Boquim’, ’Lagarto’, ’Jurubeba’], [’Aracaju’, ’Boquim’, ’Simão Dias’, ’Jurubeba’], [’Aracaju’, ’Boquim’, ’Estância’, ’Lagarto’, ’Jurubeba’]

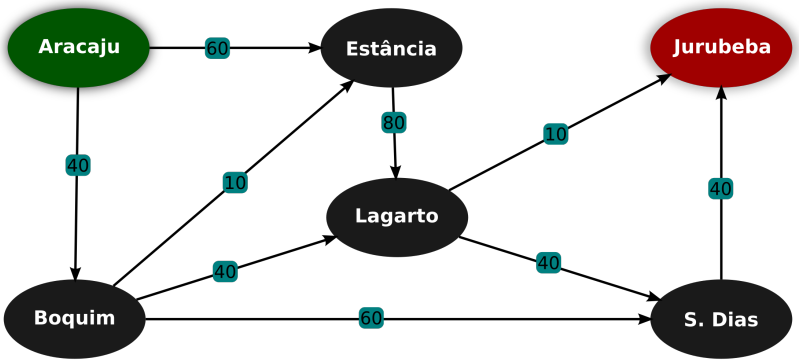
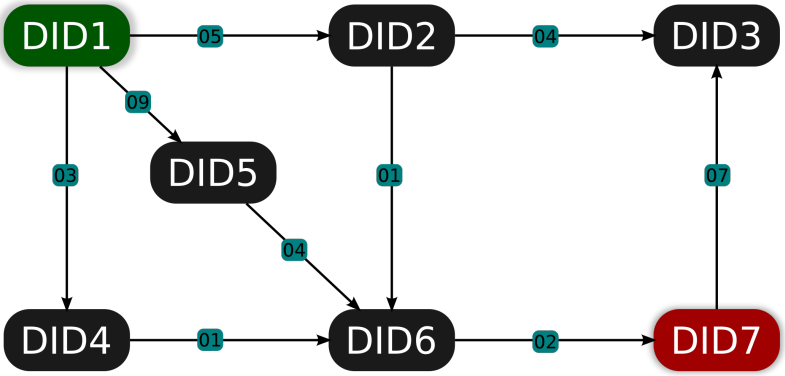


Figura 3. Yen(DID1, DID6, 3)

A = [’DID1’, ’DID4’, ’DID6’], [’DID1’, ’DID2’, ’DID6’], [’DID1’, ’DID5’, ’DID6’]



4. Complexidade

A eficiência do algoritmo de *Yen* depende de qual algoritmo será utilizado para o cáculo do primeiro caminho mínimo, o qual também será utilizado para o cálculo do nó de estimulo.

Sabemos que a implementação de Dijkstra utilizando Fibonacci Heap consome tempo de $O(M + NlogN)$ onde M será o número de arestas do grafo. Assim, *Yen* faz Kl chamadas a Dijkstra, onde l seria o tamanho do nó do *caminhoEstimulo* mencionado na linha 67 do pseudocódigo. Dessa forma, o tempo do algoritmo de *Yen* seria $O(KN(M + NLogN))$

Referências

[Dijkstra 1968] Dijkstra, E. W. (1968). A constructive approach to the problem of program correctness. *BIT Numerical Mathematics*, 8(3):174–186.

[Goldberg and Radzik 1993] Goldberg, A. and Radzik, T. (1993). A heuristic improvement of the bellman-ford algorithm. Technical report, STANFORD UNIV CA DEPT OF COMPUTER SCIENCE.

[Yen 1971] Yen, J. Y. (1971). Finding the k shortest loopless paths in a network. *management Science*, 17(11):712–716.