Relatório da implementação do algoritmo de Yen

Átila Bernardo Mota Sousa

Departamento de Computação
Universidade Federal de Sergipe (UFS) – Aracaju, SE – Brasil

Resumo. *Este relatório descreve a implementação do algoritmo de Yen em Python e suas validações. Também são mostrados os testes com diferentes grafos e suas respectivas saídas.*

1. Informações Gerais

O algoritmo de Yen é utilizado para determinar os *k-menores* caminhos entre dois vértices de um grafo. Para isso ele usa o algoritmo de *Dijkstra* (ou outro algoritmo de menor caminho) para mostrar os menores caminhos enquanto o algoritmo de Yen faz algumas modificações no grafo, como remover arestas ou vértices, determinando assim novos menores caminhos.

Nesse algoritmo são utilizados 2 conteiners *A* e *B*, *B* armazena os possíveis menores caminhos, enquanto o *A* armazena os menores caminhos em ordem crescente de custo. Também são usadas variáveis para armazenar o caminho raiz (*rootPath*), caminho atual (*spurPath*), vértice atual (*spurNode*). Também foi usado um grafo auxiliar feito através da cópia do grafo principal, dessa forma as arestas e vértices removidos sempre eram restaurados a cada iteração.

Foi implementada também uma função "*custo_caminho*" que recebe um vetor contendo o caminho e retorna o seu custo, analisando o peso das arestas. Essa função também tem a particularidade de evitar loops, pois caso verifique 2 vértices repetidos no caminho, ela retorna um custo infinito para o mesmo. Outra função foi a "*ler_saida*", que recebe o vetor A gerado pelo algoritmo de Yen e interpreta a saída com prints. Já a função "*menor_caminho*" recebe as saídas do *Dijkstra* (*rot* e *dt*) e retorna um vetor com o menor caminho.

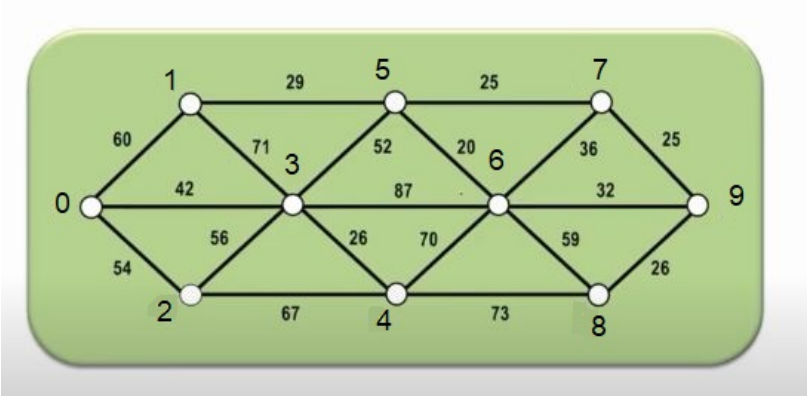
2. Resultados

2.1- Código do algoritmo de Yen em Python

```python
def yen(self, v1, v2, k):
    rot = self.dijkstra(v1)
    A = [{'cost': rot[0][v2], 'path': self.menor_distancia(v1, v2, rot[1])}]
    B = []

    for k in range(1, k):
        for i in range(0, len(A[-1]['path']) - 1) :
            grafo_aux = deepcopy(self)
            spurNode = A[-1]['path'][i]
            rootPath = A[-1]['path'][:i+1]

            for path in A:
                p = path['path']
                if rootPath == p[:i+1] and len(p) > i and len(grafo_aux.li[spurNode]) != 1:
                    grafo_aux.remove_aresta(p[i], p[i+1])

            saida2 = grafo_aux.dijkstra(spurNode)
            spurPath = grafo_aux.menor_distancia(spurNode, v2, saida2[1])
            total = rootPath[:-1] + spurPath
            totalPath = {'cost': self.custo_caminho(total), 'path': total}
            if totalPath not in B:
                B.append(totalPath)
        if len(B) == 0:
            break
        B = sorted(B, key = lambda x: x['cost'])
        A.append(B[0])
        B.pop(0)
    return A
```

2.2- Grafos testados

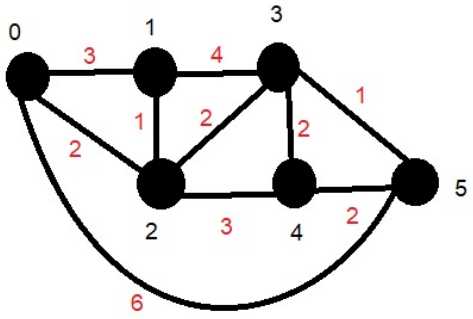
2.2.1- Grafo 1



Saída:

```
Grafo 1:
0 -> 1 -> 2 -> 3 -> NULL
1 -> 0 -> 3 -> 5 -> NULL
2 -> 0 -> 3 -> 4 -> NULL
3 -> 0 -> 1 -> 2 -> 5 -> 4 -> 6 -> NULL
4 -> 3 -> 2 -> 6 -> 8 -> NULL
5 -> 3 -> 1 -> 6 -> 7 -> NULL
6 -> 3 -> 5 -> 4 -> 7 -> 8 -> 9 -> NULL
7 -> 6 -> 5 -> 9 -> NULL
8 -> 6 -> 4 -> 9 -> NULL
9 -> 6 -> 7 -> 8 -> NULL
K = 4:
Caminhos mais curtos:
1- [0, 1, 5, 7, 9] custo: 139
2- [0, 1, 5, 6, 9] custo: 141
3- [0, 3, 5, 7, 9] custo: 144
4- [0, 3, 5, 6, 9] custo: 146
K = 5:
Caminhos mais curtos:
1- [0, 1, 5, 7, 9] custo: 139
2- [0, 1, 5, 6, 9] custo: 141
3- [0, 3, 5, 7, 9] custo: 144
4- [0, 3, 5, 6, 9] custo: 146
5- [0, 3, 6, 9] custo: 161
```

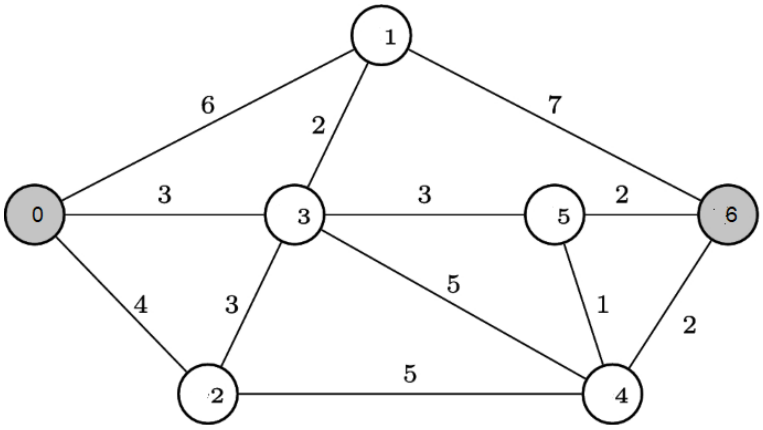
2.2.2- Grafo 2



Saída:

```
Grafo 2:
0 ->  1 ->  2 ->  5 ->  NULL
1 ->  0 ->  2 ->  3 ->  NULL
2 ->  0 ->  1 ->  3 ->  4 ->  NULL
3 ->  1 ->  2 ->  4 ->  5 ->  NULL
4 ->  2 ->  3 ->  5 ->  NULL
5 ->  3 ->  4 ->  0 ->  NULL
K = 3:
Caminhos mais curtos:
1- [0, 2, 3, 5] custo: 5
2- [0, 5] custo: 6
3- [0, 2, 4, 5] custo: 7
K = 4:
Caminhos mais curtos:
1- [0, 2, 3, 5] custo: 5
2- [0, 5] custo: 6
3- [0, 2, 4, 5] custo: 7
4- [0, 1, 2, 3, 5] custo: 7
```

2.2.3- Grafo 3



Saída:

```
Grafo 3:
0 ->  1 ->  2 ->  3 ->  NULL
1 ->  0 ->  3 ->  6 ->  NULL
2 ->  0 ->  3 ->  4 ->  NULL
3 ->  0 ->  1 ->  2 ->  5 ->  4 ->  NULL
4 ->  2 ->  3 ->  5 ->  6 ->  NULL
5 ->  3 ->  4 ->  6 ->  NULL
6 ->  1 ->  4 ->  5 ->  NULL
K = 4:
Caminhos mais curtos:
1- [0, 3, 5, 6] custo: 8
2- [0, 3, 5, 4, 6] custo: 9
3- [0, 3, 4, 6] custo: 10
4- [0, 2, 4, 6] custo: 11
K = 5:
Caminhos mais curtos:
1- [0, 3, 5, 6] custo: 8
2- [0, 3, 5, 4, 6] custo: 9
3- [0, 3, 4, 6] custo: 10
4- [0, 2, 4, 6] custo: 11
5- [0, 3, 4, 5, 6] custo: 11
```

Referências

Vídeo-aulas disponíveis no youtube do professor Renê Gusmão do Departamento de Computação da UFS. Disponível em:

<https://www.youtube.com/watch?v=54oAIViZChA&list=PLFmoA27GocLc87HfGQtGfsQveHlUIErXc&index=2&ab_channel=Prof.Ren%C3%AAGusm%C3%A3oProf.Ren%C3%AAGusm%C3%A3o>

Yen's Algorithm. Wikipedia. Disponível em:

<https://en.wikipedia.org/wiki/Yen%27s_algorithm>