Relatório da implementação do algoritmo de Dijkstra

Átila Bernardo Mota Sousa

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

Resumo. *Este relatório descreve a implementação do algoritmo de Dijkstra em Python e suas validações. Também são mostrados os testes com diferentes grafos e seus respectivos tempos de execução.*

1. Introdução

De início foi implementado o algoritmo de Dijkstra e utilizado um grafo mostrado em aula para validação. A saída da validação mostra os menores caminhos (verificados através do vetor rot) e as menores distâncias (vetor dt) de determinado vértice até o vértice de origem.

Depois disso foi implementado um algoritmo para criar grafos completos, que posteriormente foi utilizado para criar os grafos completos com a quantidade de vértices descritas na atividade do SIGAA, com pesos de arestas aleatórios no intervalo de 20 a 100. Também houve uma alteração na estrutura dos grafos, acrescentando a idéia dos pesos através de uma classe “aresta”. Depois de criados os grafos, foi aplicado o algoritmo de Djikstra e verificado seus respectivos tempos de execução.

2. Resultados

2.1- Código do algoritmo

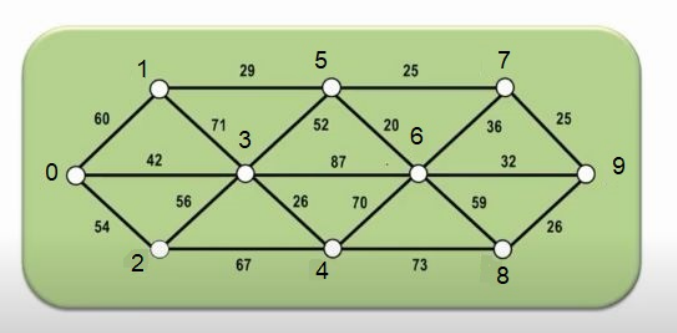
```python
def dijkstra(self, v):
    dt = []
    rot = []
    for i  in range(len(self.li)):
        dt.append(float('inf'))
        rot.append(0)
    dt[v] = 0
    rot[v] = v
    V = []
    for i in range(len(self.li)):
        V.append(i)
    A = deepcopy(V)
    F = []
    while sorted(F) != sorted(V):
        minimo = float('inf')
        for elemento in A:
            if dt[elemento] < minimo:
                minimo = elemento
        v = minimo
        F.append(v)
        A.remove(v)

        for u in self.li[v]:
            if dt[v] + u.peso < dt[u.x]:
                dt[u.x] = dt[v] + u.peso
                rot[u.x] = v

    return dt, rot
```

2.2- Validação do algoritmo

O seguinte grafo foi usado para validar o algoritmo de Dijkstra:



Foi obtida a seguinte saída:

```
Teste do algoritmo:
Grafo:
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

Menor distancia entre 0 e 1 = 60
Menor distancia entre 0 e 2 = 54
Menor distancia entre 0 e 3 = 42
Menor distancia entre 0 e 4 = 68
Menor distancia entre 0 e 5 = 89
Menor distancia entre 0 e 6 = 109
Menor distancia entre 0 e 7 = 114
Menor distancia entre 0 e 8 = 141
Menor distancia entre 0 e 9 = 139

Menor caminho entre 0 e 1: 1 - 0
Menor caminho entre 0 e 2: 2 - 0
Menor caminho entre 0 e 3: 3 - 0
Menor caminho entre 0 e 4: 4 - 3 - 0
Menor caminho entre 0 e 5: 5 - 1 - 0
Menor caminho entre 0 e 6: 6 - 5 - 1 - 0
Menor caminho entre 0 e 7: 7 - 5 - 1 - 0
Menor caminho entre 0 e 8: 8 - 4 - 3 - 0
Menor caminho entre 0 e 9: 9 - 7 - 5 - 1 - 0
```

2.3. Tempos de execução

Criando os grafos completos com um número arbitrário de vértices e aplicando Dijkstra:

```
start_time = time.time()

grafo_teste = Grafo()
grafo_teste.init_completo(100)
saida1 = grafo_teste.dijkstra(0)


end_time = time.time()
print((end_time - start_time))
```

Tempos obtidos com cada grafo:

V = 100, tempo = 0.0320 segundos

V = 200, tempo = 0.1489 segundos

V = 500, tempo = 1.1899 segundos

V = 1000, tempo = 4.4440 segundos

As demais quantidades de vértices (10000, 50000, 100000) acabaram travando o computador, por falta de processamento e memória.

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>