**ALGORITMOS I**

**Trabalho Prático 1**

Gustavo Amaral Bernardino – 2021432674

Departamento de Ciência da Computação

Universidade Federal de Minas Gerais

(DCC / UFMG)

Belo Horizonte – MG – Brasil

gustavo.amaral@dcc.ufmg.br

# INTRODUÇÃO

A situação-problema a ser solucionada pelo presente TP consiste em uma breve narrativa que, em termos práticos, corresponde ao desafio de construir um programa capaz de encontrar o caminho mais curto entre dois vértices de um grafo não-direcionado, considerando duas restrições peculiares:

1. O caminho não pode passar por nenhuma aresta com peso ímpar;
2. A quantidade de arestas contida no caminho mais curto deve ser par.

# modelagem

A estrutura de dados escolhida para representar o grafo foi uma matriz de adjacência, cujos índices representam os vértices e os valores representam os pesos das arestas entre tais vértices, sendo igual a zero quando não existe aresta entre eles. A opção por tal estrutura se deu principalmente pela maior facilidade de implementação quando comparado com outros tipos de implementação, apesar da tendência de maior custo de espaço.

Com a estrutura de dados principal definida, o próximo passo foi a definição da abordagem do problema em si. Em condições normais, a obtenção da menor distância entre quaisquer dois pontos de um grafo pode ser obtida com a implementação do algoritmo de Dijkstra, conforme visto em sala de aula. No entanto, as duas restrições precisariam ser ponderadas.

A restrição que impede o uso de arestas de peso ímpar foi abordada no algoritmo de construção da matriz de adjacência, de forma simples. Para isso, foi inserido no método de adição de arestas uma condição que permite que somente as arestas de tamanho par podem ser inseridas na matriz, de forma que tais arestas efetivamente nem existem na matriz final.

Enquanto isso, a segunda restrição, que obriga que os caminhos contenham um número par de estradas, exigiu maior cuidado. Esse tipo de problema pode ser resolvido com a duplicação de todos os vértices do grafo original, de forma que para todo vértice sejam criados os vértices e . Em seguida, para toda aresta , deve ser criadas as arestas e . Por fim, a menor distância com número par de arestas no grafo original entre quaisquer dois vértices pode ser encontrada executando o algoritmo de Dijkstra nos dois equivalentes “pares” de tais vértices no novo grafo.

Isto posto, a matriz de adjacência passou a ser construída com dimensões duas vezes maiores que o número de vértices original, ou seja, um grafo com vértices dá origem a uma matriz de dimensão . Dessa forma, passou a ser possível criar as duplicatas de cada um dos vértices e arestas originais, criando seus “pares” e “ímpares”. Nesse sentido, vale ressaltar que o código foi escrito de forma que o vértice “par” corresponde ao índice “esperado” dentro da matriz, enquanto o vértice “ímpar” corresponde a este índice acrescido do valor .

Com isso, as duas restrições resultaram em uma matriz de adjacências que não corresponde exatamente ao grafo original, mas que retorna o resultado buscado quando executado o algoritmo de Dijkstra. Para exemplificar a “transformação” do grafo original na matriz em que efetivamente o algoritmo de caminho mais curto é executado, as Imagens 1, 2 e 3 foram inseridas ao final desta documentação. Vale ressaltar que o vértice 1 corresponde ao índice 0 da matriz, o vértice 2 ao índice 1, e assim por diante.

Dessa forma, o último ponto a ser resolvido pelo algoritmo diz respeito ao algoritmo de Dijkstra propriamente dito. Para isso, a estrutura de dados utilizada foi o vetor, implementado de forma a armazenar pares de inteiros, correspondendo ao índice de um vértice e à sua distância da origem. Nesse formato foram criados dois vetores separados, sendo um chamado de “Explorados” e outro de “Fronteira”.

Os vértices Explorados são todos aqueles para os quais o algoritmo já encontrou a menor distância da origem. Enquanto isso, os vértices contidos na Fronteira são aqueles que, em um dado momento, podem ser alcançados por algum outro vértice já explorado. Assim, o algoritmo funciona basicamente avaliando sempre qual o vértice da fronteira que possui a menor distância total da origem e, assim que o encontra, adiciona esse vértice dentre os Explorados e o remove da Fronteira. Além disso, quando um vértice é Explorado, a Fronteira é expandida, pois é preciso verificar todos os vértices que esse vértice alcança.

Dessa forma, é importante ressaltar que a implementação foi feita de forma que a distância de um vértice até a origem é registrada somente dentro dos vetores e , diferente do algoritmo apresentado em sala de aula. Nesse sentido, algumas implementações de Dijkstra usam a distância de até a origem como um atributo de . Porém, o uso dos dois vetores funciona essencialmente da mesma maneira.

# apêndice

|  |
| --- |
| **Imagem 1 – Grafo Original** |
| DESTINO  ORIGEM |

|  |
| --- |
| **Imagem 2 – Matriz de Adjacência Construída** |
| Tabela  Descrição gerada automaticamente |
| As posições em vermelho representam as arestas com peso impar, que na verdade não foram efetivamente inseridas na matriz. |

|  |
| --- |
| **Imagem 3 – Grafo Equivalente à Matriz de Adjacência (Onde o Algoritmo de Caminho Mais Curto Será Rodado)** |
| Diagrama  Descrição gerada automaticamente  DESTINO  ORIGEM | |
| Arestas marcadas com X correspondem às arestas com peso ímpar, que existiam no grafo original, mas não foram adicionadas na matriz de adjacência. | |