# Trabalho Prático 2

# Algoritmos I

**Igor Roiz Teixeira**

Universidade Federal de Minas Gerais (UFMG)

Belo Horizonte - MG - Brasil

r01z@ufmg.br

## 1.Introdução

O problema proposto pede que seja desenvolvido um programa capaz de dizer como será construída uma ciclovia entre pontos turísticos de uma cidade com o menor custo possível. Serão inseridos a quantidade de pontos turísticos, junto de um número que representa o valor turístico de cada ponto, e também serão passadas as conexões possíveis entre os pontos junto de seus respectivos custos.

O problema se trata então de um grafo não direcionado onde os pontos turísticos representam os nós e as ciclovias são as arestas. Para saber qual construção terá o menor custo basta obter a Árvore Geradora Mínima (AGM) desse grafo onde o custo de cada aresta equivale a seu peso. Em caso de arestas com pesos iguais, a soma dos valores turísticos dos pontos que cada uma conecta será usado como critério de comparação, prevalecendo o maior.

## 2.Implementação

O programa foi desenvolvido em linguagem C++ e compilado com o compilador G++ da GNU Compiler Collection. Foram usadas estruturas de dados e ordenações desenvolvidas na disciplina de Estruturas de Dados. Para solucionar esse problema foi usado o algoritmo *Reverse-Delete* para obtenção da árvore geradora mínima. Essa solução foi escolhida porque é especificado na descrição do trabalho que o grafo será inserido informando uma lista de arestas, e a lista de arestas que compõem a árvore geradora mínima deverá ser apresentada ao final. Por meio desse método, com o grafo completo já pronto, só é preciso remover as arestas que não farão parte da árvore, facilitando o trabalho com as estruturas de dados.

### 2.1. Modelagem e estruturas

No começo do código é criado um vetor *nos* da estrutura *No*. Cada posição dessa estrutura representa um nó no grafo e contém o valor turístico (VT) do ponto e uma lista de inteiros que guarda quais outros nós estão conectados naquele. Por se tratar de um grafo não direcionado, quando houver uma aresta entre um nó *A* e um *B*, *B* será registrado na lista de adjacência do nó *A* e vice-versa.

Também é criada uma lista encadeada *arestas* que guarda cada uma das arestas contidas no grafo. Para cada aresta é registrado os nós que ela conecta, o seu custo e o VT somado dos nós. Uma segunda lista, *arvoreGM*, é criada e ao final ela estará armazenando as arestas que compõem a AGM.

### 2.2. Obtenção da AGM

Para obter a AGM por meio do método *reverse-delete* é necessário primeiro ordenar a lista de arestas com base no custo de cada uma, da menor para a maior, como dito anteriormente, em caso de duas arestas com o mesmo custo são comparados os VTs de cada uma, sendo o maior aquele que virá primeiro.

Uma vez que a lista está ordenada é chamada a função *reverseDelete* que fará o processo e obtenção da árvore. Nessa função cada aresta é conferida a partir da de maior custo até a menor. Os nós que fazem parte da aresta são apagados de suas respectivas listas de adjacência em seguida o vetor de nós é passado para outra função que confere se o grafo continua conectado usando o caminhamento de grafo DFS. Se o grafo desconectar significa que a aresta faz parte da AGM, então os nós são inseridos de volta nas listas de adjacência, os custos e VTs totais dessa árvore são atualizados e a aresta é copiada para lista *arvoreGM*.

Abaixo o pseudocódigo da implementação da função *reverseDelete*, ao final dessa função, a lista *arvoreGM* terá todas as arestas que compõem a AGM e as listas de adjacência de cada terá apenas aqueles nós que se conectam na árvore.

```
Ordena a lista de arestas do menor custo para o maior;
Para cada posição da lista de arestas (da última a primeira):
    Obter os nós v,u que compoem a aresta;
    Remover v da lista de adjcência de u;
    Remover u da lista de adjcência de v;
    Testa se o grafo está conectado
```

```
Se grafo estiver desconectado:
  Insere v na lista de adjacência de u;
  Insere u na lista de adjacência de v;
  Incrementa custo total da árvore com o custo da aresta;
  Incrementa o VT total da árvore com o VT da aresta;
  Insere a aresta atual na lista arvoreGM;
```

## 3.Análise Assintótica de Tempo

O tempo de execução do algoritmo depende principalmente do número de arestas *E* e do número de nós *N*.

O primeiro passo é a ordenação da lista de arestas que foi feita antes da execução da função *reverseDelete*, como foi usado *bubblesort*, a ordenação tem tempo $O(E^2)$.

Em seguida é chamada a função para gerar a árvore que irá percorrer todos os *E* elementos da lista de arestas. A cada iteração é feita a chamada da função que confere se o grafo está conectado. Essa função usa o algoritmo *DFS* de custo $O(E+N)$ para obter a situação da conectividade do grafo e depois executa um *loop* $O(N)$ para conferir a conectividade.

O tempo de execução então é $O(E\ (E+N))$. Essa análise se aplica ao código desenvolvido para esse trabalho, o algoritmo *Reverse Delete* pode ainda ser melhorado para obter o tempo $O(E \log V(\log \log V)^3)$.