UFMG

# Trabalho Prático JII

*Bicicletário de Belleville*


## Eduardo Capanema

Matrícula 2020041515

Aluno de Graduação

Matemática Computacional

Universidade Federal de Minas Gerais


**Fevereiro, 2021**

*Submetido para Avaliação: Disciplina de Algoritmos I (Semestre 2020/2).*

# 1

# Documentação

**O trabalho prático II da disciplina de Algoritmos I, do segundo semestre letivo do ano de 2020, nos solicitou que implementássemos, utilizando a linguagem de programação c ou c++, uma solução para um problema de ordem prática. Resumidamente, o problema consiste em determinar, a partir de uma entrada padronizada, uma ciclovia que minimiza o custo de sua produção e, ao mesmo tempo, maximiza o valor turístico ao se percorrer tal ciclovia, dados valores fornecidos para tais valores.**

## 1.1 | Modelagem Computacional

A modelagem computacional que julgamos adequada para a solução do problema consistiu em utilizar Grafos Direcionados Ponderados (com pesos), (uma vez que as arestas representam o valor de construção de tal trecho caso ele seja construído).

A partir do padrão de entrada dos dados, criamos uma classe *Graph* responsável por fornecer um nível de abstração que aumenta a facilidade de leitura e compreensão do código proposto. A medida que lemos a entrada padrão, populamos a classe com os vértices determinados e seus pesos.

## 1.2 | Estruturas de Dados

Em nossa implementação, utilizamos algumas estruturas de dados bastante comuns. Além dos grafos direcionados ponderados, que já mencionamos acima, nosso código utilizou vetores para ordenamento dos vértices dos gra-

fos, vetores (*vector*) para manipulação de entradas (e.g. *sort*, *push_back*, *pop*, dentre outras).

### 1.2.1 | Lista de Prioridades

Utilizamos, em nossa implementação, uma lista de prioridades, para ordenar, segundo o melhor caso de construção da ciclovia para a representação de nossas possíveis ciclovias presentes no grafo utilizado o que se mostrou extremamente útil no problema em questão por se tratar de uma abordagem gulosa (greedy).

"Um algoritmo guloso escolhe, em cada iteração, o objeto mais apetitoso que vê pela frente. (A definição de apetitoso é estabelecida *a priori*, antes da execução do algoritmo.) O objeto escolhido passa a fazer parte da solução que o algoritmo constrói." *IME - USP - Algoritmos Gulosos*

## 1.3 | Algoritmos

O principal algoritmo utilizado em nossa implementação foi o Algoritmo de Prim, que é útil para encontrar uma Árvore Geradora Mínima (*Minimum Spanning Tree*).

"Um exemplo de uso de uma árvore de extensão mínima seria a instalação de fibras óticas num campus de uma faculdade. Cada trecho de fibra ótica entre os prédios possui um custo associado (isto é, o custo da fibra, somado ao custo da instalação da fibra, mão de obra, etc). Com esses dados em mãos (os prédios e os custos de cada trecho de fibra ótica entre todos os prédios), podemos construir uma árvore de extensão que nos diria um jeito de conectarmos todos os prédios sem redundância." Wiki

### 1.3.1 | Pseudocodigo de Prim

Nossa implementação do algoritmo de Prim realizou-se da seguinte forma:

```
T = vazio;
U = { 1 };
while (U diff V)
    let (u, v) be the lowest cost edge such
    that u in U and v in V - U;
    T = T in {(u, v)}
    U = U in {v}
```

O limitador, opera conforme o pseudocódigo destacado abaixo:

## 1.4 | Complexidade

Ao final de nossa implementação, realizamos a análise da complexidade assintótica, tanto do tempo quanto do espaço (uma vez que, conforme já mencionado, utilizamos o Algoritmo de Kruskal).

Nossa implementação mostrou-se prática uma vez que sua execução, em tempo e espaço, é linear, i.e. $O(V + E)$, onde $V$ representa o número de vértices e $E$ o número de arestas.

É importante destacar que a Busca em Profundidade (DFS) possui este caráter linear o que o torna um algoritmo considerado bastante prático e pouco custoso do ponto de vista de seu custo de tempo. Contudo, este algoritmo utiliza-se largamente de uma função recursiva em sua implementação, podendo se tornar custoso no que se refere ao uso de memória nos sistemas que o executam.

## 1.5 | Conclusão

Nosso trabalho foi capaz de solucionar, para as entradas padrão fornecidas e outras que fomos capazes de testar, o encaminhamento adequado ao longo do Grafo Direcionado sem peso que modelamos. Sua execução é linear, permitindo que afirmemos tratar-se de uma implementação prática e segura. As estruturas de dados são simples e comuns, tornando sua compilação descomplicada e acessível. O algoritmo utilizado em sua implementação, i.e. o algoritmo de Kruskal condiz com o objeto de estudo da disciplina nesta fase

do aprendizado e se presta excelentemente bem à missiva. Finalmente, gostaríamos de ressaltar a simplicidade e concisão do código proposto que toma algo em torno de 180 linhas e que apresenta boa lógica de abstração (facilitando sua leitura e compreensão), além de estar devidamente comentado e organizado.