
Miguel Gewehr de Oliveira
Áurea Santos de Oliveira
Gabriel Zuany

# Travelling Salesman Problem

Vitória, ES
2023

# 1 Introdução

Este documento apresenta o *Travelling Salesman Problem* (TSP). O Problema do Caixeiro-Viajante (PCV) é um problema que tenta determinar a menor rota para percorrer uma série de cidades (visitando uma única vez cada uma delas), retornando à cidade de origem. Diversos algoritmos e heurísticas foram desenvolvidos para abordar o TSP e encontrar uma *Minimum Spanning Tree (MST)*. Neste trabalho utilizaremos o algoritmo de Kruskal para tal.

Além desta introdução, este documento está organizado da seguinte forma: a Seção 2 apresenta os objetivos propostos para o trabalho, o que buscamos alcançar; a Seção 3 apresenta a metodologia utilizada nesta atividade, junto com o por que dessa escolha;por fim, a Seção 4 apresenta os resultados obtidos do trabalho.

# 2 Objetivos

O trabalho tem como objetivo aprofundar o estudo de Técnicas de Busca e Ordenação, além de apresentar estruturas de dados adequadas, como o *Union-find*, num contexto aplicado a grafos, o TSP. Para isso, foi utilizado o algoritmo deKruskal para gerar uma árvore geradora mínima, e em seguida foi feito uma *Busca Primeiro em Profundidade* (DFS) na MST para encontrar um Tour válido.

# 3 Metodologia

A metodologia abordada para o trabalho foi tratar as cidades do problema como vétices de um grafo ponderado, onde os pesos das arestas são as distâncias euclidiana entre dois vétices. Foi implementado um *U*nion-find genérico que armazena os vértices e uma fila de prioridade para armazenar as arestas para o cálculo da MST com o algoritmo de Kruskal.

## 3.1 Union Find

A estrutura consiste em um vetor que armazena, em cada posição, o ID do ponto e o ID do ponto-pai. Foram implementadas as funções *union* para conectar um ponto ao outro e *find* para conferir se dois pontos estavam conectados, isto é, fazem parte de um

mesmo componente.

## 3.2 árvore geradora mínima (MST)

Considerando que cada ponto é ligado a todos os outros, o vetor das arestas existentes cresce conforme o triângulo de Pascal, comumente usado para calcular combinações. O vetor é ordenado com a função para que, seguindo o algoritmo de Kruskal, sejam analisadas sempre as menores distancias primeiro, afim de encontrar a combinação de arestas que geram a árvore geradora de menor peso.

## 3.3 Tour

Para criação do tour, foi armazenada a lista de adjacências dos vertices da MST (complexidade: lgN) e depois realizada uma DFS (complexidade: O(v+e)), onde as cidade visitadas foram armazenadas em um array e depois escrita em um arquivo de saída (complexida: O(N) em relação ao número de pontos).

# 4 Resultados e Avaliação

## 4.1 Análise Teórica

Durante o planejamento e oraganização das ideias pelo time, foram debatidas as Estrutuuras de Dados mais adequadas, complexidades para as operações previstas, performance e memória para cada etapa da resolução do problema.

Separamos o problema em 6 partes fundamentais: Leitura dos Dados, Construção de um Grafo Não Dirigido Completo, Ordenação das Arestas, Construção da MST e Tour.

Para cada uma delas, estimamos a sua complexidade e como isso impactaria no desempenho do programa como um todo.

### 4.1.1 Leitura dos Dados

- Complexidade: O(N) . Linear, cresce de modo proporcional ao número de cidades existentes.

### 4.1.2 Construção do Grafo Não Dirigido Completo

- Complexidade: O($N^2$). Quadratico, para cada vértice, conecta este com todos os demais. O tamanho alocado segue uma aritmetica de elementos centrais do Triangulo de Pascal ($N * (N - 1)/2$).

### 4.1.3 Ordenação das Arestas (*qsort*)

- Complexidade no melhor caso $O(n * log(n))$ - Complexidade no pior caso $O(n^2)$

### 4.1.4 Construção da MST

- Complexidade: $O(n * log(n))$. Linerar devido ao tamanho de cidades e log para juntar dois vértices na estrutura QuickUnionFind.

### 4.1.5 Tour

- Complexidade: O(Vértices + Arestas). Complexidade do DFS.

## 4.2 Análise Empírica

### 4.2.1 Geração da Amostras (casos teste)

Foi escrito um script em Python para gerar amostras aleatórias contendo grafos de tamanhos diversos, a fim de realizar um profiling mais preciso e ter um estudo mais assertivo acerca das complexidades.

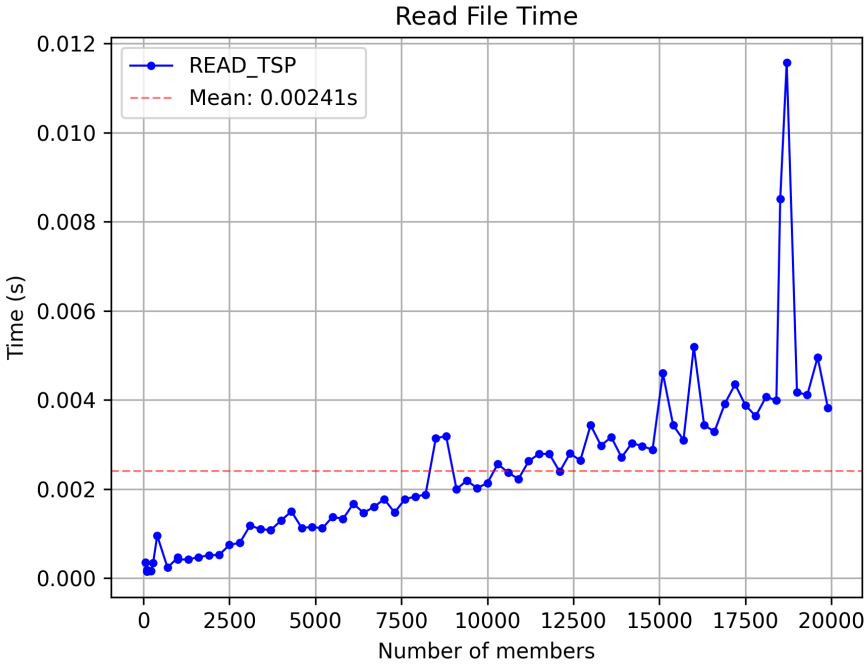


Figura 1 – Leitura dos Dados

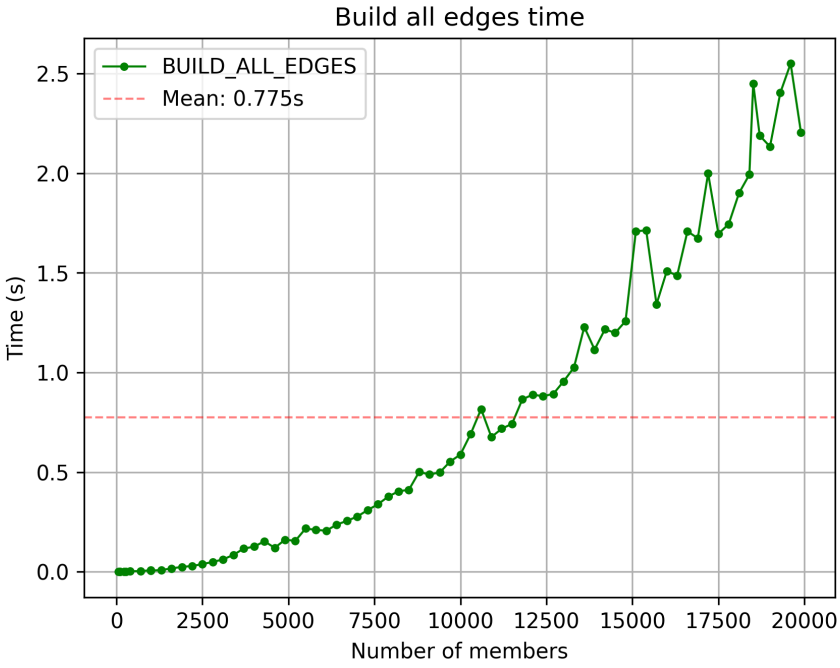


Figura 2 – Construção do Grafo Não Dirigido Completo

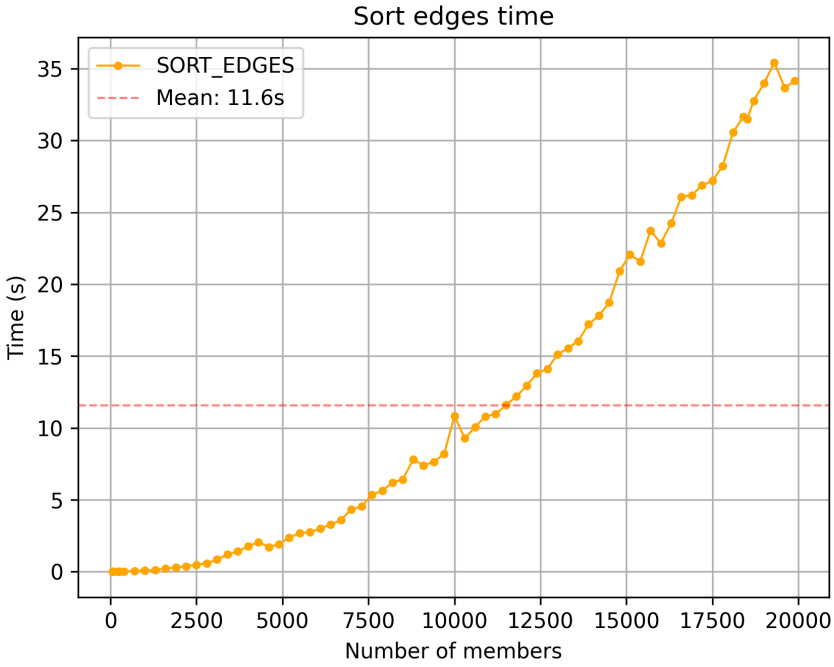


Figura 3 – Ordenação das Arestas

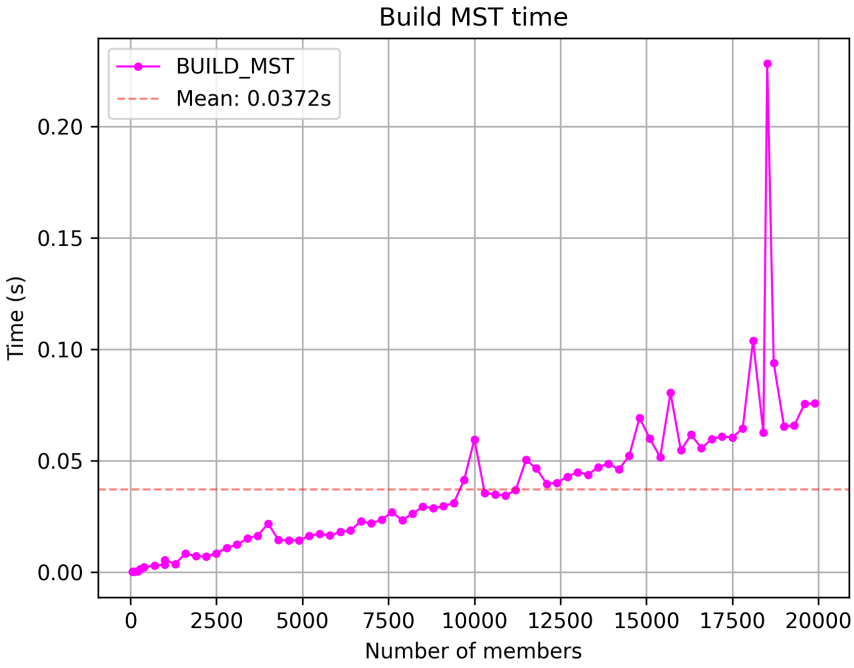


Figura 4 – Construção da MST

# Referências