# Trabalho Prático AEDs II
# Caixeiro Viajante

**André Garcia Vieira**
**Matrícula 2016060403**

[1]Departamento de ciência da computação – Universidade Federal de Mina Gerais(UFMG)

andregarcia@dcc.ufmg.br

***Resumo.*** *Este trabalho prático remete-se a disciplina de algoritmo e estrutura de dados ministrada em 2017/2, onde o objetivo é desenvolver um código que visa solucionar o problema do caixeiro viajante.*

## 1. Introdução

O problema do caixeiro viajante (PCV) é um problema matématico sem solução analítica que visa determinar a menor rota possível que o caixeiro deve percorrer para visitar todas as instâncias, retornando a cidade de origem.
O PCV se enquadra no conjunto de problemas NP-Difícil. Isso significa dizer que o não pode ser resolvido em um tempo polinomial de acordo com o número de entradas.

## 2. Proposta

Criar um TAD, que implemente a solução ótima para o PCV euclidiano. Será fornecido um arquivo de entrada contendo o número de cidades a ser visitadas o nome da cidade com suas coordenadas. O nome da cidade é também um número.

### 2.1. Parte 1

primera parte do trabalho consiste em gerar um arquivo de saída contendo:

- O número de soluções possíveis.
- Todas as rotas possíveis.

#### 2.1.1. O Numero de Soluções

O número de soluções do PCV pode ser determinado como:

$$S = N!$$

onde S é Número de soluções e N é o número de instâncias a serem visitadas.
Podemos notar, através da Tabela 1, que o número de soluções cresce muito rápido de acordo com o número de entradas.

#### 2.1.2. Rotas

Para gerar as rotas foi usado um algorítmo de gerar permutações sem repetições. Esse algorítmo foi desenvolvido no TP0 da mesma disciplina. Após gerar as rotas, essas rotas devem ser impressas em arquivo contendo, simplismente as coordenadas de cada instância no formato (X,Y).

| Número de Entradas | Número de Soluções |
|---|---|
| 3 | 6 |
| 4 | 24 |
| 6 | 720 |
| 10 | 3628800 |
| 14 | 87178291200 |

**Tabela 1. Tabela de número de Soluções**

### 2.1.3. Dificuldades encontradas

Ao começar a implementar o algorítimo de permutação deparei com vários problemas. o algorítmo consistia em gerar um vetor de tamanho N e iniciar o vetor com todas as posições igual a 1. Após isso ia percorrendo o vetor verificando se já existia aquela instância no vetor, se existisse incrementava 1 e testava de novo.
Tal forma de inicialização se mostrou bastante ineficiente. A primeira forma de otimização pensada foi iniciar cada posição do vetor com uma instância diferente, essa maneira de inicialização ja fez com que o tempo de execução do programa já reduzisse drasticamente.
O tempo de execução do programa de acordo com o número de entradas na Figura 1.

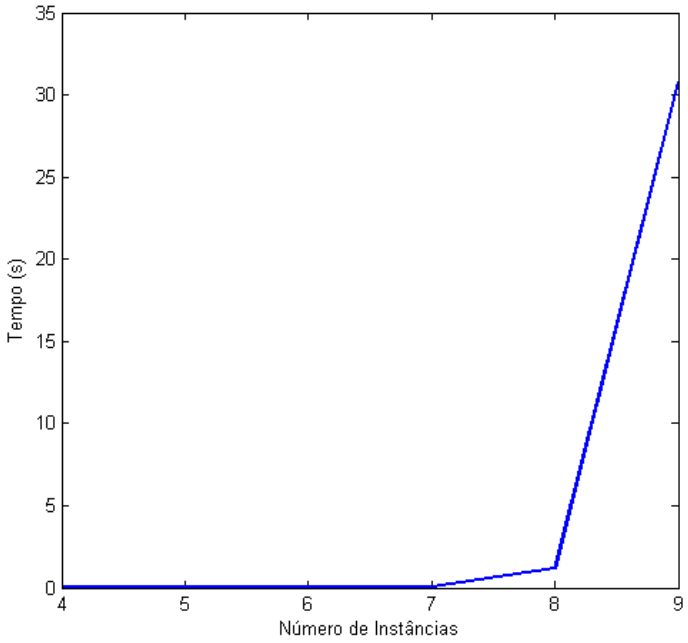


**Figura 1. Tempo de execução por número de instâncias**

Foi rodado uma única vez a aplicação para 10 instâncias e o tempo de execução foi superior a 15 minutos.
Foi rodado também uma vez para 14 instâncias, conforme orientações do TP, porem, o tempo de execução já estava superior a 1 dia e a execução foi interrompida.

### 2.2. Parte 2

A segunda parte consiste em encontrar a solução ótima. Para tal solução é necessário calcular a distância de todas as rotas possíveis e comparar todas exibindo a menor distância resultado.

#### 2.2.1. Metodologia

Para otimizar as buscas de distâncias foi utilizado uma matriz de adjacência. A matriz de adjacêcias consiste em que um dado elemento aij da matriz, representa a distância entre i e j.

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0.000000 | 45.486263 | 13.152946 | 25.059929 |
| 2 | 45.486263 | 0.000000 | 55.172459 | 20.615528 |
| 3 | 13.152946 | 55.172459 | 0.000000 | 34.655445 |
| 4 | 25.059929 | 20.615528 | 34.655445 | 0.000000 |

**Tabela 2. Matriz de Adjacência**

## 3. Conclusão

Durante o desenvolvimento do trabalho encarei várias dificudades. Essas dificudade demandaram, de mim, mais tempo do que o esperado, mas consegui terminar o código com razoável êxito.