DESCOBRINDO AS MENORES DISTÂNCIAS ENTRE DIVERSAS CIDADES COM PYTHON.

BETANIA S. C. CAMPELLO

ORIENTADOR: WASHINGTON A. OLIVEIRA

CO ORIENTADORA: CARLA T. L. S. GHIDINI

Introdução

O QUE É A PESQUISA OPERACIONAL?
PROBLEMA DO CAIXEIRO VIAJANTE.

O QUE É A PESQUISA OPERACIONAL?

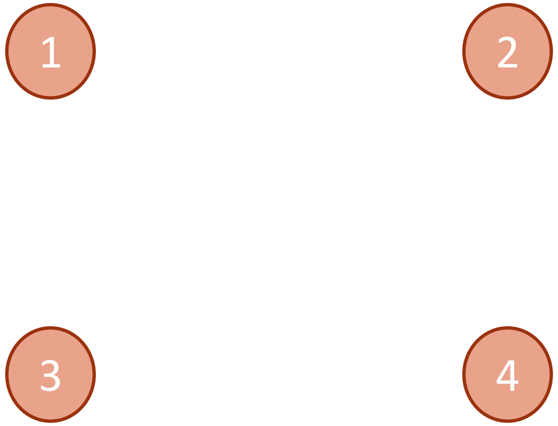
- ✓ *É um ramo interdisciplinar da **matemática aplicada.***
- ✓ *Faz uso de **modelos matemáticos**, estatísticos e de algoritmos **na ajuda à tomada de decisão.***
- ✓ *Busca **otimizar um processo**, isto é, visa encontrar o melhor resultado a partir de determinadas características do problema.*

O PROBLEMA DO CAIXEIRO VIAJANTE

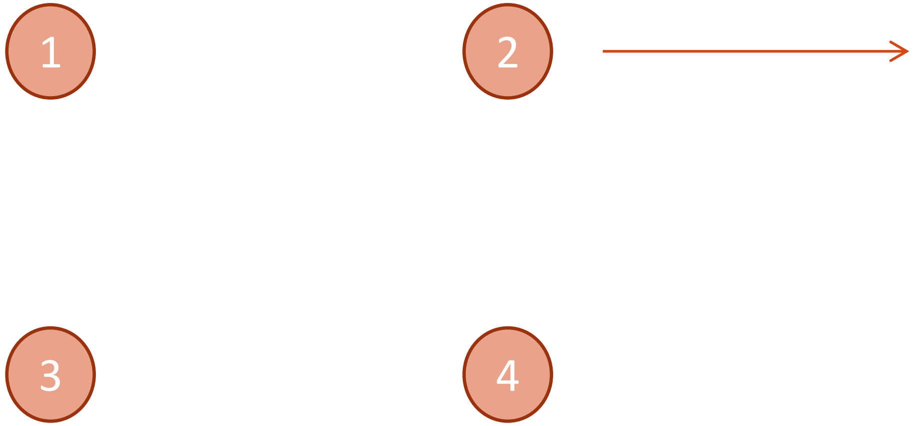
Por exemplo:

- ✓ *Problema do caixeiro viajante – busca de uma rota para visitar todas as cidades sem visitar a mesma cidade duas vezes com o mínimo custo possível.*

O PROBLEMA DO CAIXEIRO VIAJANTE

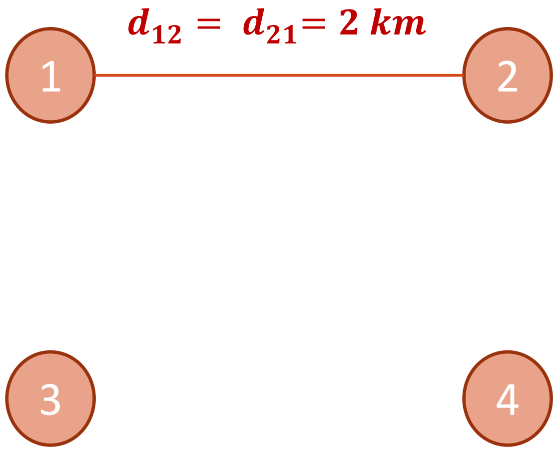


O PROBLEMA DO CAIXEIRO VIAJANTE



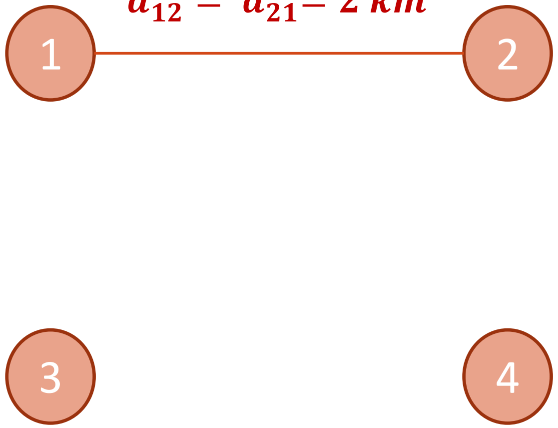
*As cidades são representadas por **NÓS** .*

O PROBLEMA DO CAIXEIRO VIAJANTE



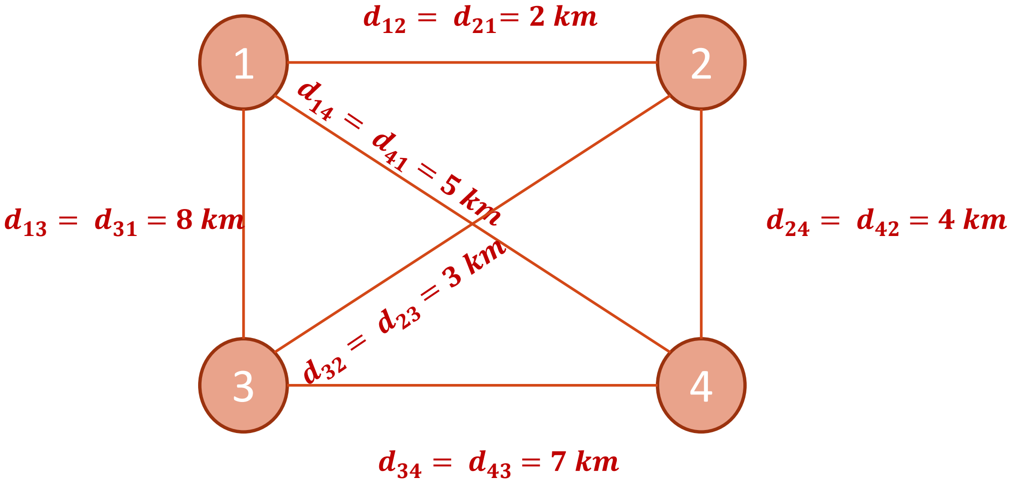
O PROBLEMA DO CAIXEIRO VIAJANTE

$d_{12} = d_{21} = 2\ km$

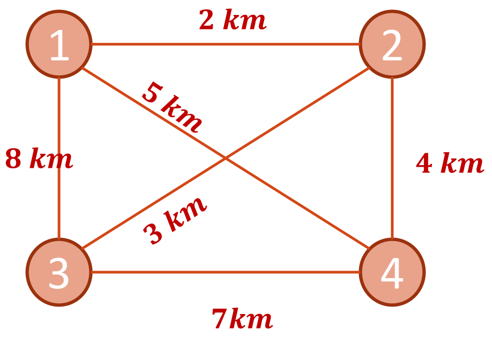


Os possíveis caminhos são representados por arestas

O PROBLEMA DO CAIXEIRO VIAJANTE



O PROBLEMA DO CAIXEIRO VIAJANTE



Matriz de distâncias:

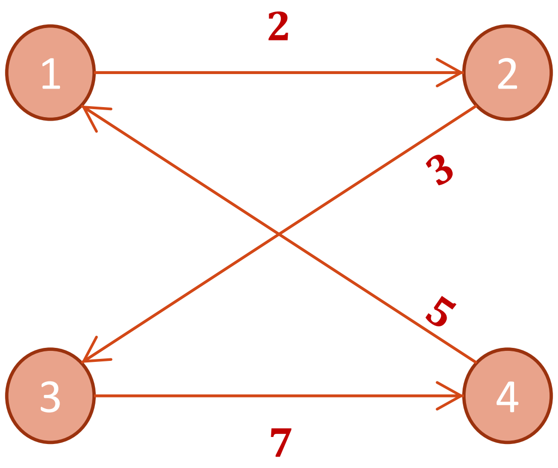
nó	**1**	**2**	**3**	**4**
1	∞	2	8	5
2	2	∞	3	4
3	8	3	∞	7
4	5	4	7	∞

O PROBLEMA DO CAIXEIRO VIAJANTE

Existem (n-1)! possíveis trajetos:
Neste exemplo: (4-1)! = 3! = 6

O PROBLEMA DO CAIXEIRO VIAJANTE

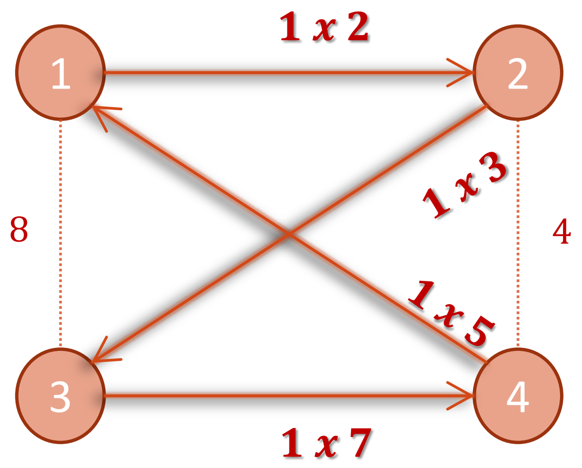
1º exemplo de trajeto



TRAJETO:	DISTÂNCIA:
✓ 1 - 2 - 3 - 4 - 1	2 + 3 + 7 + 5 = 17

O PROBLEMA DO CAIXEIRO VIAJANTE

1º

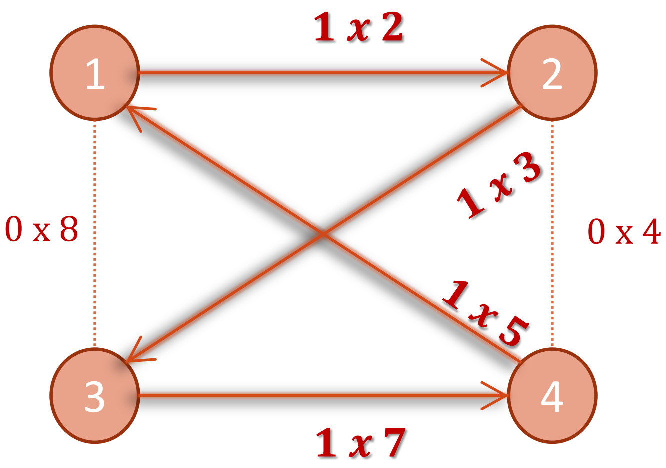


As arestas em destaque são as arestas ou caminhos ***ATIVOS****.*

Posso multiplicar as distâncias das arestas ***ativas*** *pelo número* ***1****.*

O PROBLEMA DO CAIXEIRO VIAJANTE

1º

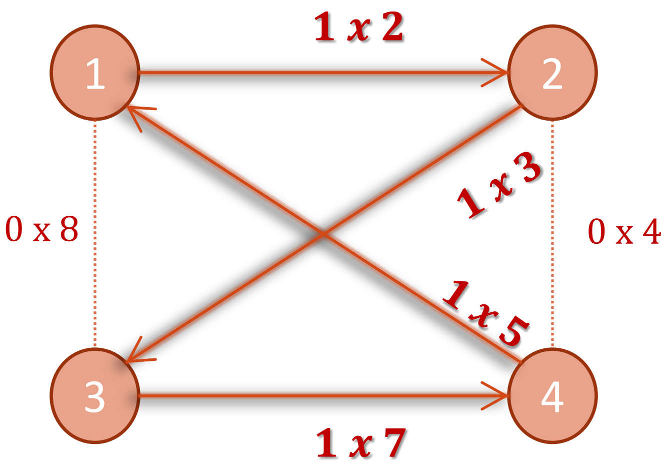


*As arestas não destacadas são arestas ou caminhos **INATIVOS**. Indicam que aquele caminho **NÃO** deve ser feito.*

*Posso multiplicar as distâncias das arestas **inativas** pelo número **0.***

O PROBLEMA DO CAIXEIRO VIAJANTE

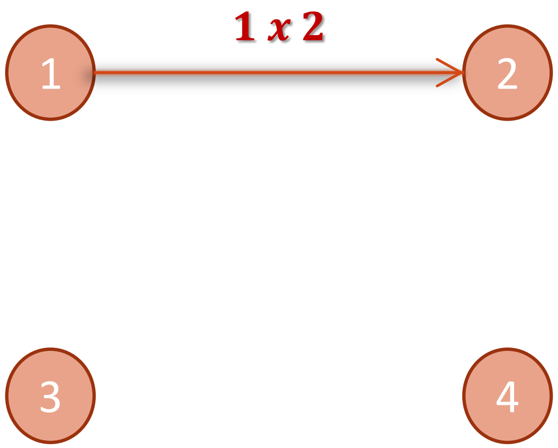
1º



Depois basta somar todas as distâncias das arestas para encontrar a distância total do trajeto.

O PROBLEMA DO CAIXEIRO VIAJANTE

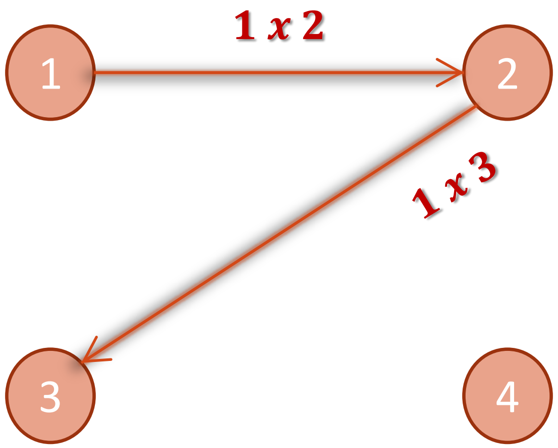
1º



Distância:
(1 x 2) + ...

O PROBLEMA DO CAIXEIRO VIAJANTE

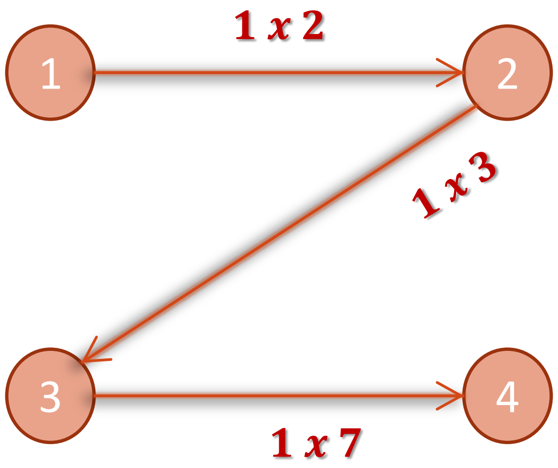
1º



Distância:
(1 x 2) + (1 x 3) + ...

O PROBLEMA DO CAIXEIRO VIAJANTE

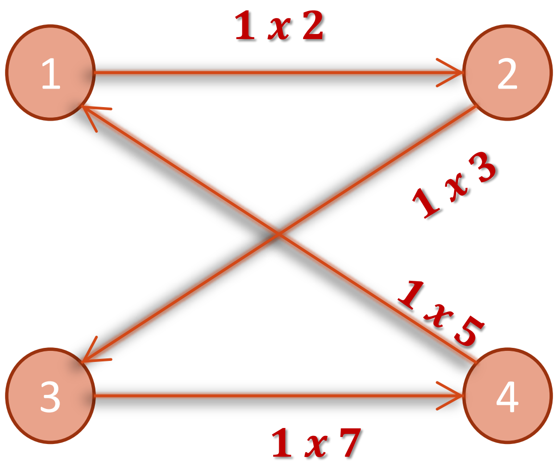
1º



Distância:
(1 x 2) + (1 x 3) + (1 x 7) + ...

O PROBLEMA DO CAIXEIRO VIAJANTE

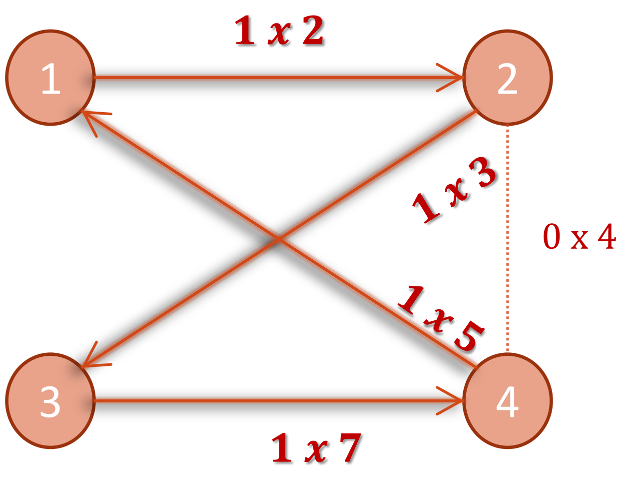
1º



Distância:
(1 x 2) + (1 x 3) + (1 x 7) + (1 x 5) + ...

O PROBLEMA DO CAIXEIRO VIAJANTE

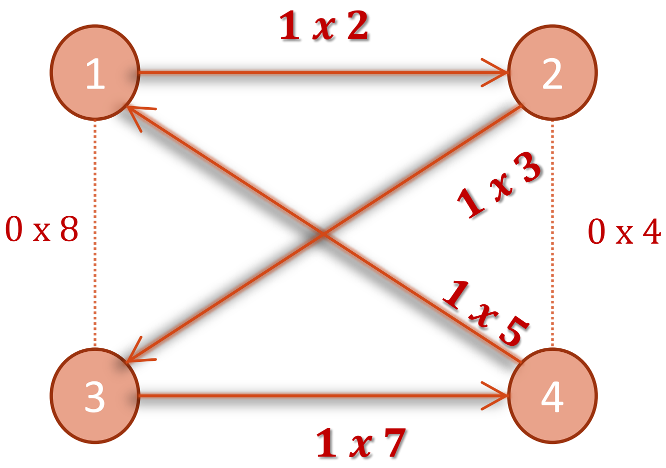
1º



Distância:
(1 x 2) + (1 x 3) + (1 x 7) + (1 x 5) + (0 x 4) + ...

O PROBLEMA DO CAIXEIRO VIAJANTE

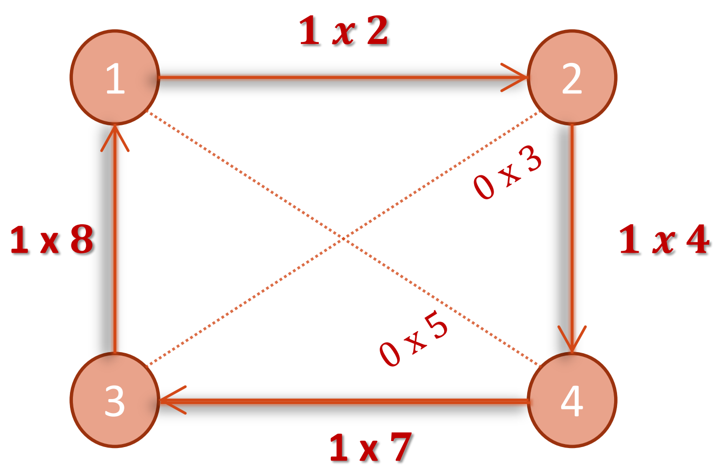
1º



Distância:
(1 x 2) + (1 x 3) + (1 x 7) + (1 x 5) + (0 x 4) + (0 x 8) = 17

O PROBLEMA DO CAIXEIRO VIAJANTE

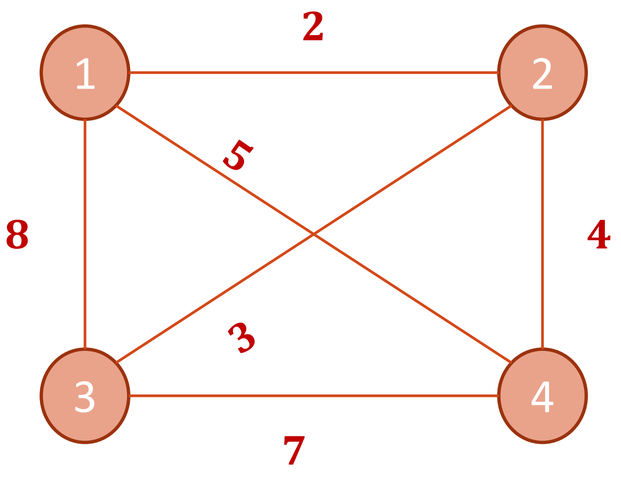
2º exemplo de trajeto



- *TRAJETO:*
  1 - 2 - 4 - 3 - 1

- *DISTÂNCIA:*
  (1x 2) + (1x 4) + (1 x 7) + (1x 8) + (0 x 5) + (0 x 3) = 21

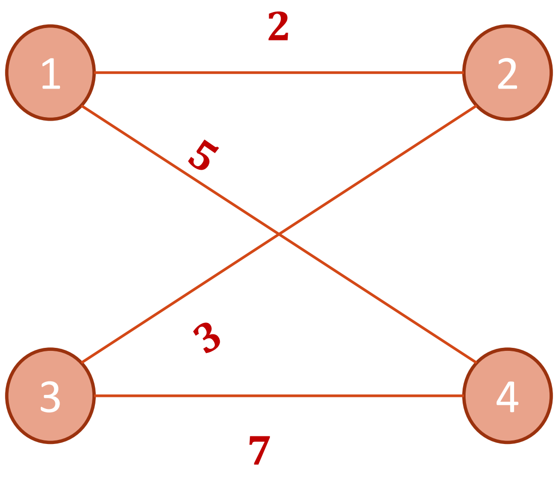
O PROBLEMA DO CAIXEIRO VIAJANTE



Existem (n-1)! possíveis caminhos: (4-1)! = 3! = 6

	TRAJETO:	*DISTÂNCIA:*
1.	*1 - 2 - 3 - 4 - 1*	*2 + 3 + 7 + 5 = 17*
2.	*1 - 2 - 4 - 3 - 1*	*2 + 4 + 7 + 8 = 21*
3.	*1 - 3 - 2 - 4 - 1*	*8 + 3 + 4 + 5 = 20*
4.	*1 - 3 - 4 - 2 -1*	*8 + 7 + 4 + 2 = 21*
5.	*1 - 4 - 2 - 3 -1*	*5 + 4 + 3 + 8 = 20*
6.	*1 - 4 - 3- 2 -1*	*5 + 7 + 3 + 2 = 17*

O PROBLEMA DO CAIXEIRO VIAJANTE

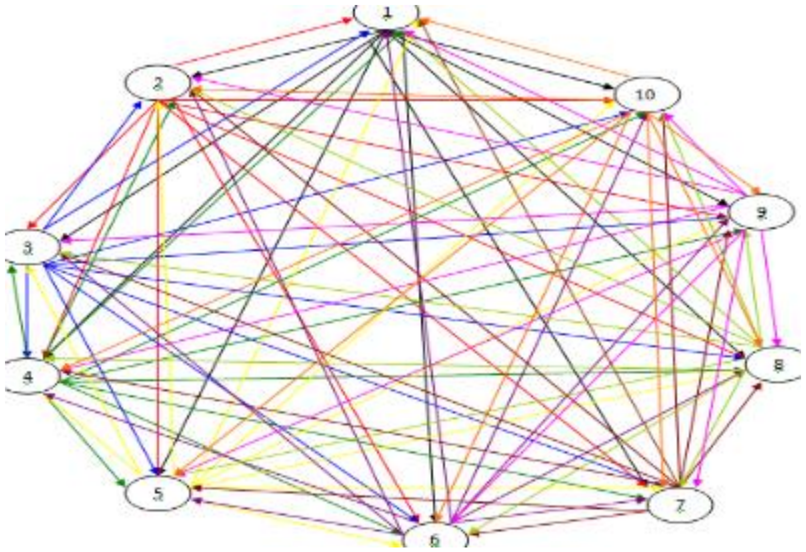


Existem (n-1)! possíveis caminhos: (4-1)! = 3! = 6

	TRAJETO:	*DISTÂNCIA:*
1.	***1 - 2 - 3 - 4 - 1***	***2 + 3 + 7 + 5 = 17***
2.	*1 - 2 - 4 - 3 - 1*	*2 + 4 + 7 + 8 = 21*
3.	*1 - 3 - 2 - 4 - 1*	*8 + 3 + 4 + 5 = 20*
4.	*1 - 3 - 4 - 2 -1*	*8 + 7 + 4 + 2 = 21*
5.	*1 - 4 - 2 - 3 -1*	*5 + 4 + 3 + 8 = 20*
6.	***1 - 4 - 3- 2 -1***	***5 + 7 + 3 + 2 = 17***

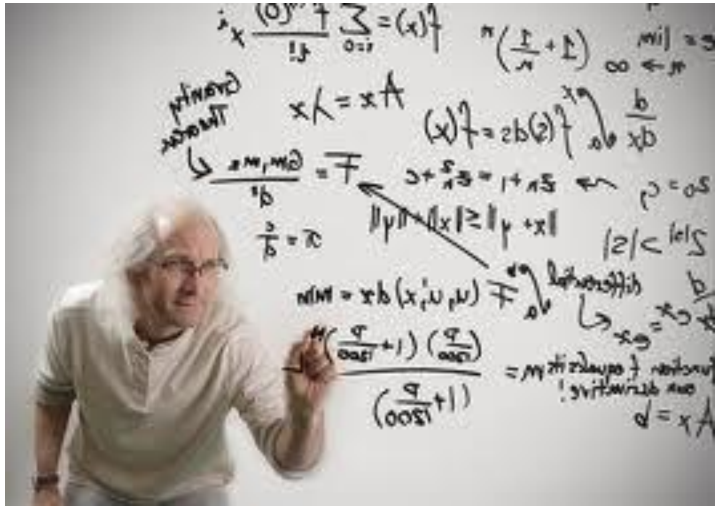
O PROBLEMA DO CAIXEIRO VIAJANTE

Para problemas pequenos, é possível obter a solução ótima por busca exaustiva (força bruta). Porém, imagine uma situação onde temos 200 ou 1.000 cidades a serem percorridas, isso significa (1.000 - 1)! caminhos possíveis.



O PROBLEMA DO CAIXEIRO VIAJANTE

Não é necessário fazer os cálculos por força bruta. Existem algoritmos que encontram a melhor solução a partir de um modelo matemático.



O PROBLEMA DO CAIXEIRO VIAJANTE

Resultado deste problema com algoritmos que encontram a melhor solução a partir de um modelo matemático.

```
Trajeto:  [0, 1, 2, 3, 0]
Distancia total:  17
Tempo de execucao: 0:00:00.022000
```

O PROBLEMA DO CAIXEIRO VIAJANTE

Modelo matemático consiste em:

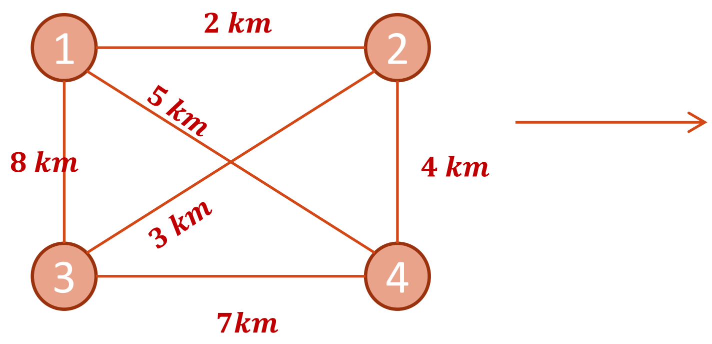
- ✓ ***PARÂMETROS DO MODELO***

O PROBLEMA DO CAIXEIRO VIAJANTE

Modelo matemático consiste em:

✓ ***PARÂMETROS DO MODELO***

Matriz de distâncias:



nó	1	2	3	4
1	∞	2	8	5
2	2	∞	3	4
3	8	3	∞	7
4	5	4	7	∞

O PROBLEMA DO CAIXEIRO VIAJANTE

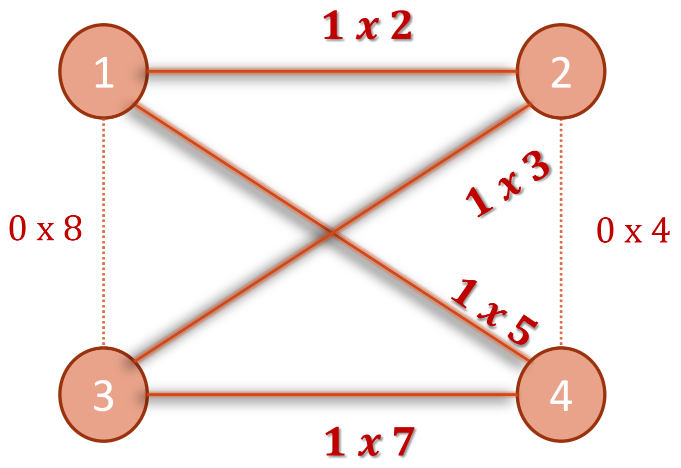
Modelo matemático consiste em:

- ✓ *PARÂMETROS DO MODELO*
- ✓ ***VARIÁVEIS DE DECISÃO***

O PROBLEMA DO CAIXEIRO VIAJANTE

Modelo matemático consiste em:

- ✓ *PARÂMETROS DO MODELO*
- ✓ ***VARIÁVEIS DE DECISÃO***



O PROBLEMA DO CAIXEIRO VIAJANTE

Modelo matemático consiste em:

- ✓ *PARÂMETROS DO MODELO*
- ✓ *VARIÁVEIS DE DECISÃO*
- ✓ ***OBJETIVO***

O PROBLEMA DO CAIXEIRO VIAJANTE

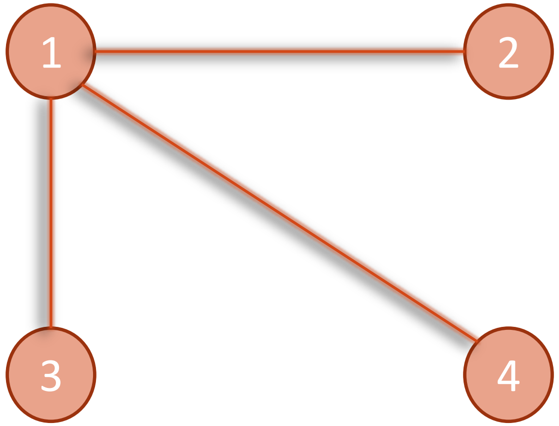
Modelo matemático consiste em:

- ✓ *PARÂMETROS DO MODELO*
- ✓ *VARIÁVEIS DE DECISÃO*
- ✓ *OBJETIVO*
- ✓ ***RESTRIÇÕES***

O PROBLEMA DO CAIXEIRO VIAJANTE

Modelo matemático consiste em:

- ✓ *PARÂMETROS DO MODELO*
- ✓ *VARIÁVEIS DE DECISÃO*
- ✓ *OBJETIVO*
- ✓ ***RESTRIÇÕES***



Programas para Otimização Usando Linguagem Python

SOLVER: GLPK, CPLEX

Solver

- ✓ *É um programa para otimização.*
- ✓ *Usa algoritmos eficientes para a busca da solução ótima dado o modelo matemático.*

GLPK

- ✓ *Open Sourse*
- ✓ *Fácil de instalar*
- ✓ *Limitado*

https://github.com/coin-or/pulp

CPLEX

- ✓ *Versão gratuita para estudantes*
- ✓ *Versão gratuita para provar*
- ✓ *Bastante robusto*

https://www.ibm.com/developerworks/community/blogs/jfp/entry/cplex_studio_in_ibm_academic_initiative?lang=en

https://www.ibm.com/developerworks/community/blogs/jfp/entry/Solving_Sudoku_In_Python_With_DOcplex?lang=en

Programação do Problema do Caixeiro Viajante em Python

Programação do Problema do Caixeiro Viajante em Python

Programação do Problema do Caixeiro Viajante em Python

✓ *Passo 1: Criando o objeto*

Objeto 1
Parâmetros 2
Variáveis de Decisão 3
Função Objetivo 4
Restrições 5
Resolução 6
Solução 7


```
from docplex.mp.context import DOcloudContext
from docplex.mp.model import Model


docloud_context = DOcloudContext()
model           = Model('Final33', docloud_context=docloud_context)
```

Programação do Problema do Caixeiro Viajante em Python

Objeto 1
Parâmetros 2
Variáveis de Decisão 3
Função Objetivo 4
Restrições 5
Resolução 6
Solução 7

✓ *Passo 2: Definindo os parâmetros - Matriz de distâncias*

```
#MATRIZ DE DISTANCIA
d_ij = [[1000000000, 625,        1734,       2791,       2040,       2702,       2121,       2510,       3382       ],
        [625,        1000000000, 1273,       2330,       1580,       2242,       1659,       1952,       2922       ],
        [1734,       1273,       1000000000, 1060,       308,        1030,       654,        1422,       1637       ],
        [2791,       2330,       1060,       1000000000, 775,        346,        854,        1512,       566        ],
        [2040,       1580,       308,        775,        1000000000, 898,        656,        1484,       1300       ],
        [2702,       2242,       1030,       346,        898,        1000000000, 699,        1303,       677        ],
        [2121,       1659,       654,        854,        656,        699,        1000000000,853,         1330       ],
        [2510,       1952,       1422,       1512,       1484,       1303,       853,        1000000000, 1792       ],
        [3382,       2922,       1637,       566,        1300,       677,        1330,       1792,       1000000000 ]]
```

Programação do Problema do Caixeiro Viajante em Python

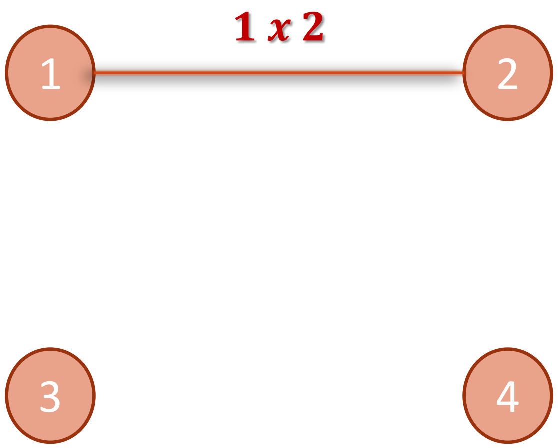
Objeto 1
Parâmetros 2
Variáveis de Decisão 3
Função Objetivo 4
Restrições 5
Resolução 6
Solução 7

✓ *Passo 3: Definindo as variáveis de decisão*

Relembrando:

Programação do Problema do Caixeiro Viajante em Python

Definindo a VARIÁVEL:



Distância:

(1 x 2) + ...

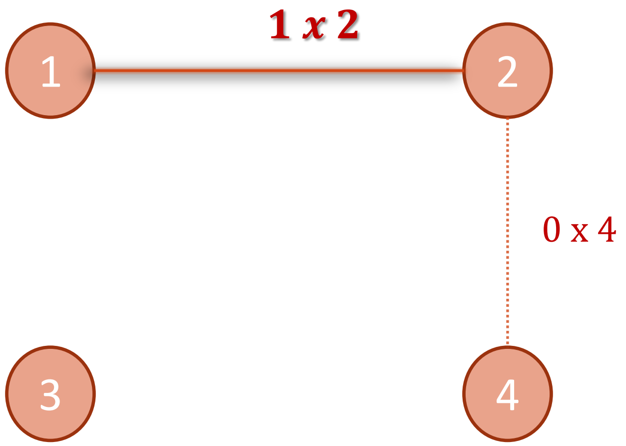
1 se a aresta está ativa, ou seja, se o percurso (i , j) deve ser feito...

Objeto 1
Parâmetros 2
Variáveis de Decisão 3
Função Objetivo 4
Restrições 5
Resolução 6
Solução 7

Programação do Problema do Caixeiro Viajante em Python

Objeto 1
Parâmetros 2
Variáveis de Decisão 3
Função Objetivo 4
Restrições 5
Resolução 6
Solução 7

Definindo a VARIÁVEL:



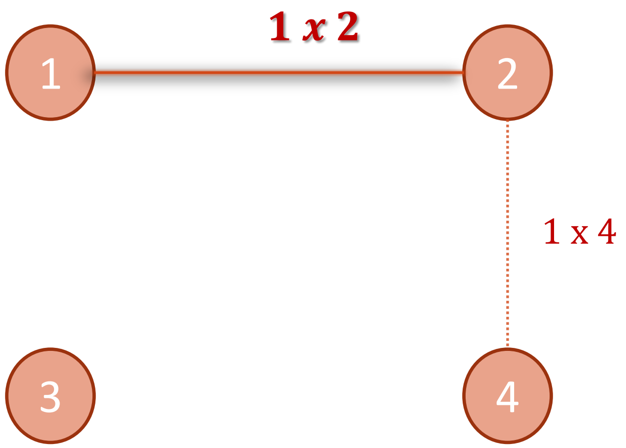
Distância:
(1 x 2) + (0 x 4)

e zero caso a aresta não seja ativa, ou seja, o caminho (i,j) não deve ser feito.

Programação do Problema do Caixeiro Viajante em Python

Objeto 1
Parâmetros 2
Variáveis de Decisão 3
Função Objetivo 4
Restrições 5
Resolução 6
Solução 7

Definindo a VARIÁVEL:



Distância:

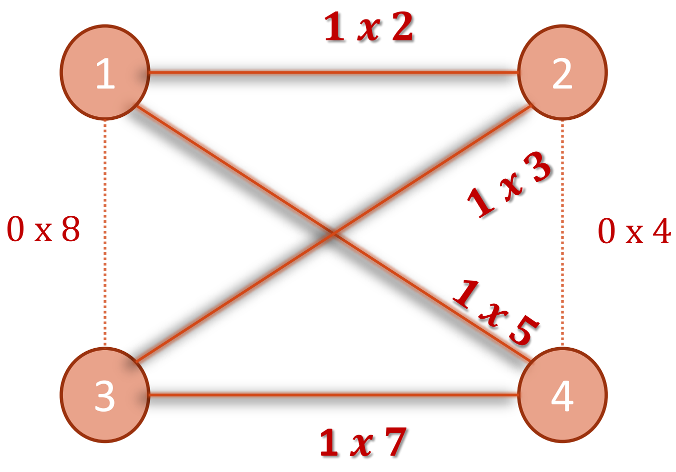
$(\underbrace{1}\ x\ \ 2) + (\underbrace{0}\ x\ \ 4)$

Este valor (zero ou um) será a variável que o solver deverá calcular. Chamamos esta variável de $\boldsymbol{X_{ij}}$

Programação do Problema do Caixeiro Viajante em Python

Objeto 1
Parâmetros 2
Variáveis de Decisão 3
Função Objetivo 4
Restrições 5
Resolução 6
Solução 7

Definindo a VARIÁVEL:



Distância:

$\underbrace{(1 \times 2) + (1 \times 3) + (1 \times 7) + (1 \times 5)} + \underbrace{(0 \times 4) + (0 \times 8)} = 17$

O solver deverá retornar 1 para todos os caminhos (i,j) ativos

E zero para todos os caminhos (i,j) não ativos

Programação do Problema do Caixeiro Viajante em Python

Objeto 1
Parâmetros 2
Variáveis de Decisão 3
Função Objetivo 4
Restrições 5
Resolução 6
Solução 7

Representando em Python as VARIÁVEIS DE DECISÃO:
CUIDADO! Em DOCPLEX não é possível representar as variáveis de decisão com bibliotecas tipo Numpy

```
#VARIAVEL DE DECISAO
x_ij = [[model.binary_var(name='x_ij_%d_%d' %(I,J))
                         for J in range(0, num_cidades)]
                         for I in range(0, num_cidades)]
```

Programação do Problema do Caixeiro Viajante em Python

✓ *Passo 4:*

Definindo o OBJETIVO: Nosso objetivo é fazer o percurso com a menor distância possível, isto é, o somatório de todas as arestas (ativas e inativas) multiplicado pela correspondente distância deve ser o menor valor possível:

Minimize: $\sum_{i=0}^{N} \sum_{j=0}^{N} X_{ij} d_{ij}$

Programação do Problema do Caixeiro Viajante em Python

Objeto 1
Parâmetros 2
Variáveis de Decisão 3
Função Objetivo 4
Restrições 5
Resolução 6
Solução 7

Representando em Python a FUNÇÃO OBJETIVO:

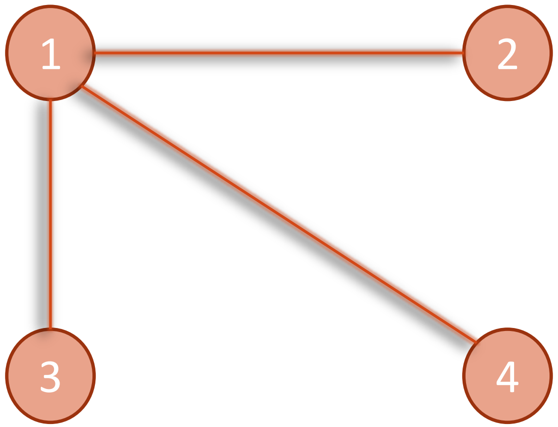
Minimize: $\sum_{i=0}^{N} \sum_{j=0}^{N} X_{ij} d_{ij}$

```
#OBJETIVO
model.minimize(model.sum([x_ij[I][J] * d_ij[I][J] for I in range(0, num_cidades)
                                                  for J in range(0, num_cidades)]))
```

Programação do Problema do Caixeiro Viajante em Python

Objeto 1
Parâmetros 2
Variáveis de Decisão 3
Função Objetivo 4
Restrições 5
Resolução 6
Solução 7

- *Passo 5: Definindo as restrições*



Programação do Problema do Caixeiro Viajante em Python

Definindo as restrições:

Para todo i:

$$\sum_{J=0}^{N}(x_{ij}) = 1$$

Para todo j:

$$\sum_{i=0}^{N}(x_{ij}) = 1$$

Objeto 1
Parâmetros 2
Variáveis de Decisão 3
Função Objetivo 4
Restrições 5
Resolução 6
Solução 7

Programação do Problema do Caixeiro Viajante em Python

Objeto 1
Parâmetros 2
Variáveis de Decisão 3
Função Objetivo 4
Restrições 5
Resolução 6
Solução 7

Para todo i:

$$\sum_{J=0}^{N} (x_{ij}) = 1$$

```
for I in range(0, num_cidades):
    model.add_constraint(model.sum([x_ij[I][J] for J in range(0,num_cidades)]) == 1)
```

Programação do Problema do Caixeiro Viajante em Python

Objeto 1
Parâmetros 2
Variáveis de Decisão 3
Função Objetivo 4
Restrições 5
Resolução 6
Solução 7

Para todo j:

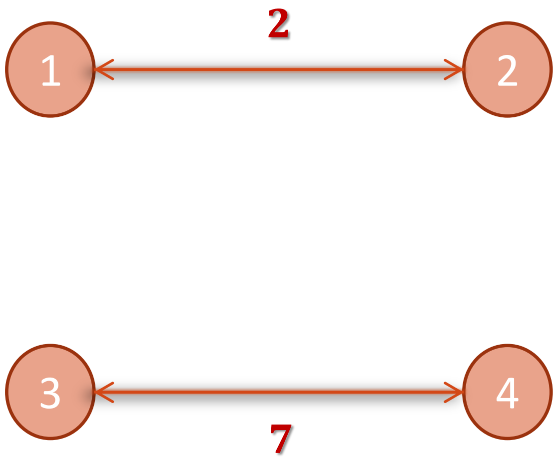
$$\sum_{i=0}^{N}(x_{ij}) = 1$$

```
for J in range(0, num_cidades):
    model.add_constraint(model.sum([x_ij[I][J] for I in range(0, num_cidades)]) == 1)
```

Programação do Problema do Caixeiro Viajante em Python

Objeto 1
Parâmetros 2
Variáveis de Decisão 3
Função Objetivo 4
Restrições 5
Resolução 6
Solução 7

Sub rotas:



Programação do Problema do Caixeiro Viajante em Python

Objeto 1
Parâmetros 2
Variáveis de Decisão 3
Função Objetivo 4
Restrições 5
Resolução 6
Solução 7

Evitando Sub rotas:

```
#RESTRICAO DE SUBROTA
for subrota in range(3, num_cidades / 2 + 1):
    combinacao =  list(itertools.permutations(range(num_cidades), subrota - 1))
    for item in combinacao:
        arestas = list(itertools.permutations(item, 2))
        model.add_constraint(model.sum([x_ij[c[0]][c[1]]for c in arestas]) <= (subrota - 2))
```

Programação do Problema do Caixeiro Viajante em Python

Objeto 1
Parâmetros 2
Variáveis de Decisão 3
Função Objetivo 4
Restrições 5
Resolução 6
Solução 7

- ✓ *Passo 6: Resolvendo o modelo*

```
model.solve()
```

Programação do Problema do Caixeiro Viajante em Python

Objeto 1
Parâmetros 2
Variáveis de Decisão 3
Função Objetivo 4
Restrições 5
Resolução 6
Solução 7

- ✓ *Passo 7: Obtendo a solução das variáveis.*

```
#OBTENDO A SOLUCAO
solucao = [[x_ij[I][J].solution_value
                    for J in range(0, num_cidades)]
                    for I in range(0, num_cidades)]
```

Programação do Problema do Caixeiro Viajante em Python

Objeto 1
Parâmetros 2
Variáveis de Decisão 3
Função Objetivo 4
Restrições 5
Resolução 6
Solução 7

- ✓ *Obtendo a solução da função objetivo.*

```
#OBTENDO A FUNCAO OBJETIVO
model.print_solution()
```

Programação do Problema do Caixeiro Viajante em Python

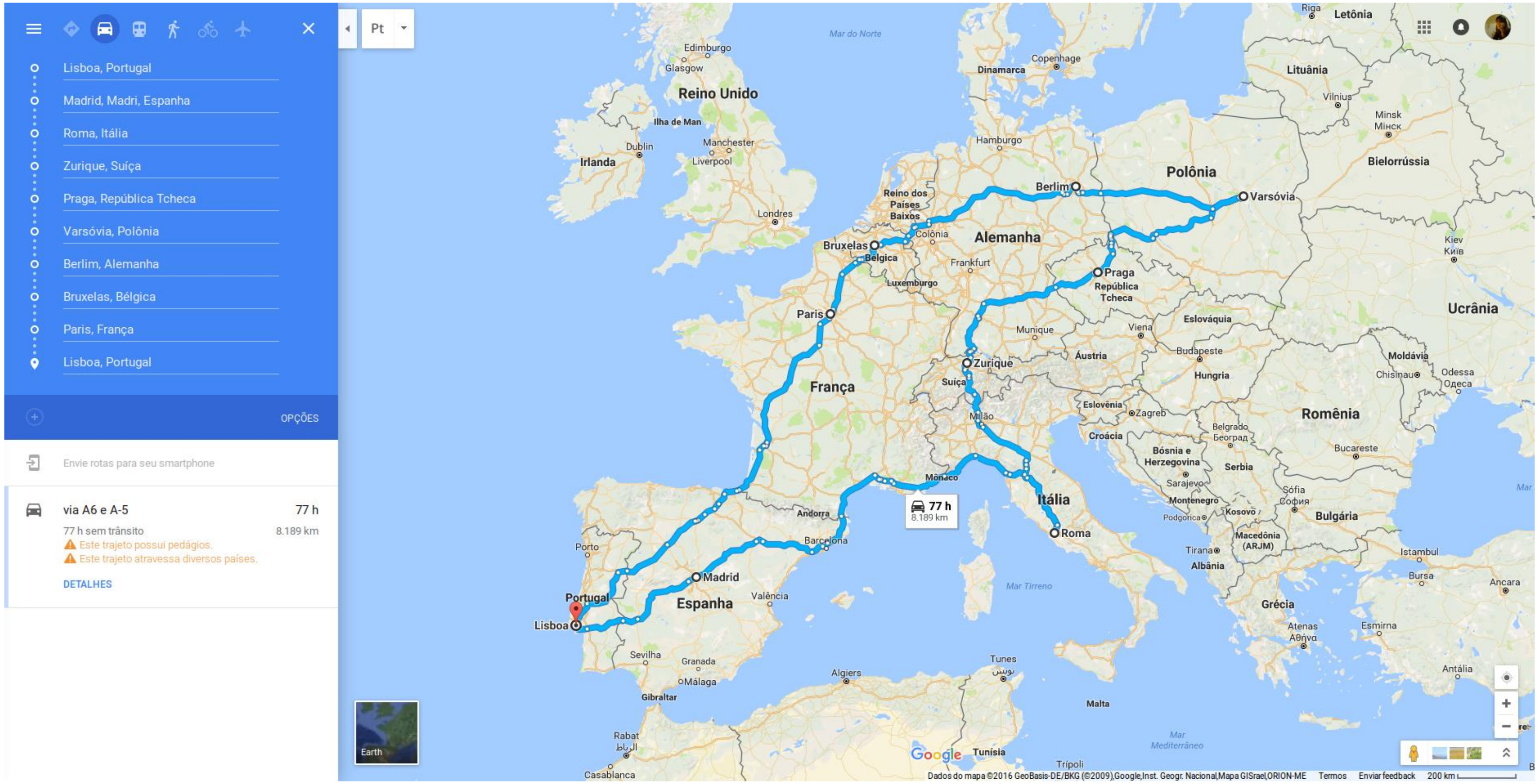
- ✓ *Resultado final:*

```
objective: 8189
  x_ij_5_8=1
  x_ij_4_2=1
  x_ij_1_7=1
  x_ij_6_5=1
  x_ij_0_1=1
  x_ij_8_3=1
  x_ij_7_6=1
  x_ij_2_0=1
  x_ij_3_4=1


Solucao Matriz de caminhos
[[0, 1, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 1, 0],
[1, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 1, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 1],
[0, 0, 0, 0, 0, 1, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 1, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0, 0]]
```

```
Trajeto:  [0, 1, 7, 6, 5, 8, 3, 4, 2, 0]
Tempo de execucao: 0:00:00.074000
```

Programação do Problema do Caixeiro Viajante em Python



Programação do Problema do Caixeiro Viajante em Python

- ✓ *Problemas de entrega de mercadorias.*
- ✓ *Fabricação de placas de circuito eletrônico*
- ✓ *Raio - X cristalográfico (análise da estrutura dos cristais)*
- ✓ *Recolhimento de lixo*
- ✓ *Serialização em arqueologia*

Referências

[1] Tópicos em otimização combinatória - Prof. Dra. Priscila Rampazzo

[2] O problema do caixeiro viajante, teoria e aplicações. Conte, Nelson, 2002.

DÚVIDAS?

OBRIGADA!

Betania S. C. Campello – betania.campello@fca.unicamp.br
washington A. Oliveira - washington.oliveira@fca.unicamp.br
Carla T. L. S. Ghidini - carla.ghidini@fca.unicamp.br