**Uma solução para o "Problema do Caixeiro Viajante"**

**Mário Leite**

**...**

Um dos problemas mais estudados e mais interessantes da Matemática Aplicada é, sem dúvida o **"Problema do Caixeiro Viajante"** (conhecido pela sigla TSP), é um clássico da área de otimização de processos na "Teoria dos Grafos"; por tanto, um assunto bem complexo. Resumidamente, se trata da seguinte situação: "***Dado um conjunto de cidades e as distâncias entre cada par de cidades, o objetivo é encontrar o melhor caminho (mais curto) que o caixeiro pode visitar cada cidade uma só vez e retornar à cidade de origem***". Deste modo o objetivo é "encontrar a rota mais eficiente para minimizar a distância total percorrida". Então, o problema pode ser formulado de maneira mais formal usando um grafo, onde as cidades são **nós** e as distâncias entre elas as **arestas**. Deste modo deve ser encontrado um circuito hamiltoniano (um ciclo que visita cada nó exatamente uma vez) com o menor comprimento possível. Sem entrar nos detalhes mais complexos e resumidamente do ponto de vista matemático, a formulação da solução pode ser apresentada na forma: **G = (V,C)**, onde **V** é o conjunto de vértices (cidades) e **C** o conjunto de arestas (distâncias entre as cidades), reduzindo o problema a encontrar uma permutação **P** dos vértices de maneira que a soma das distâncias seja a menor possível.

De um modo geral, o "Problema do Caixeiro Viajante" é conhecido por ser do tipo **NP** *(o problema "**P** versus **NP**" é o principal problema aberto da Ciência da Computação*) **– difícil**; o que significa que não existe um algoritmo eficiente conhecido que resolva todas as instâncias do problema em tempo polinomial. Assim, abordagens exatas como *"força bruta"* podem se tornar impraticáveis para um grande número de cidades, mesmo com algoritmos heurísticos para encontrar soluções aceitáveis em tempo razoável

O programa **"CaixeiroViajante"**, codificado em Python, é uma solução bem simples para um exemplo simples sobre esse assunto; mas dá uma visão geral de como resolver o problema, que se resume no seguinte algoritmo:

1) O grafo é representado por uma matriz, onde a variável-lista **LstMatGrafo[i][j]** armazena o custo de viajar da cidade **i** para a cidade **j**.
2) Geração de Permutações: O programa gera todas as permutações possíveis das cidades, representando diferentes ordens de visita.
3) Cálculo do Custo: Para cada permutação o programa calcula o custo total da rota, somando os custos de viajar de uma cidade para a próxima.
4) Atualização da Melhor Rota: Se uma rota atual tiver um custo menor do que a melhor rota conhecida até o momento a melhor rota e seu custo são atualizados.
5) Saída de Resultados: No final, o programa imprime a melhor rota encontrada e o custo total.

A abordagem de "força bruta" testa todas as possíveis permutações, garantindo que não se perca a melhor solução. No entanto, esta abordagem pode ser computacionalmente cara para um grande número de cidades; existem algoritmos mais eficientes para lidar com instâncias...
A **figura 1** mostra um exemplo de grafo, e a **figura 2** a saída do programa para o exemplo de grafo dado como uma matriz-lista na variável lista **LstMatGrafo[i][j]**.

---

**Nota**: Apenas como referência didática, as variáveis com prefixo **Lst** são todas variáveis indexadas do tipo **lista**. As outras que não têm esse prefixo são variáveis simples

---

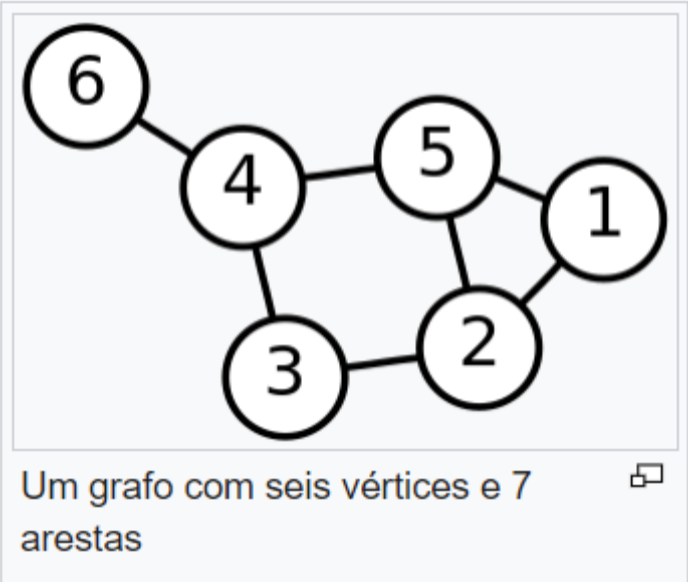


**Figura 1 - Exemplo de um grafo simples**
**Fonte: Wikipédia**

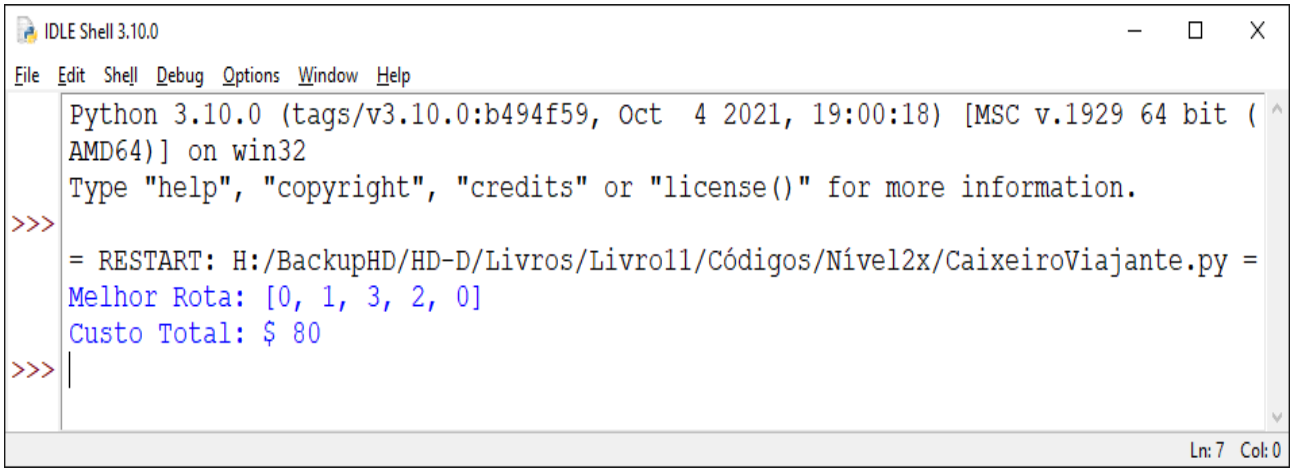


**Figura 2 - Saída do programa "CaixeiroViajante"**

```
'''
CaixeiroViajante.py
-------------------------------------------------------------------------------
Cria uma solução simples para o "Problema do Caixeiro Viajante", indicando
a melhor rota e o custo total associado.
-------------------------------------------------------------------------------
'''
import itertools

def CalcularCusto(LstMatGrafo, LstRota):
    #Função para calcular o custo total
    custo = 0
    for i in range(len(LstRota)-1):
        custo += LstMatGrafo[LstRota[i]][LstRota[i+1]]
    custo += LstMatGrafo[LstRota[-1]][LstRota[0]]  #retorna ao ponto inicial
    return custo
#-------------------------------------------------------------------------------
def Permutar(LstMatGrafo,LstRotaAtual,inicio,fim,LstMelhorRota,LstMelhorCusto):
    if(inicio == fim):
        custoAtual = CalcularCusto(LstMatGrafo, LstRotaAtual)
        if(custoAtual < LstMelhorCusto[0]):
            LstMelhorCusto[0] = custoAtual
            LstMelhorRota[0] = LstRotaAtual.copy()
    else:
        for i in range(inicio, fim+1):
            LstRotaAtual[inicio], LstRotaAtual[i] = LstRotaAtual[i],
            LstRotaAtual[inicio]
            Permutar(LstMatGrafo, LstRotaAtual, inicio+1, fim, LstMelhorRota,
            LstMelhorCusto)
            LstRotaAtual[inicio], LstRotaAtual[i] = LstRotaAtual[i],
            LstRotaAtual[inicio]  #retorna à configuração original

#-------------------------------------------------------------------------------
def ExecutarTSP(LstMatGrafo):
    #Função para executar a solução "tsp" do Caixeiro Viajante
    vertices = len(LstMatGrafo)
    LstMelhorRota = [None]
    LstMelhorCusto = [float('inf')]
    LstRotaInicial = list(range(vertices))  #cria a lista da rota inicial
    Permutar(LstMatGrafo,LstRotaInicial,0,vertices-1,LstMelhorRota, LstMelhorCusto)
    print("Melhor Rota:", LstMelhorRota[0] + [LstMelhorRota[0][0]])
    print(f"Custo Total: $ {LstMelhorCusto[0]}")

#===============================================================================
#Programa principal
if __name__ == "__main__":
    LstMatGrafo = [
        [0, 10, 15, 20],
        [10, 0, 35, 25],
        [15, 35, 0, 30],
        [20, 25, 30, 0]
    ]

    ExecutarTSP(LstMatGrafo)
#Fim do programa "CaixeiroViajante"-----------------------------------===----
```