

# Circuito Hamiltoniano na Teoria dos Grafos

---

Disciplina: Projeto de Análise de Algoritmos

---

Doscente: Leonardo Nogueira Matos

Discente: Caio Vasconcelos Silva Andrade

# Introdução

## Circuitos Hamiltonianos

Um Circuito Hamiltoniano em um grafo é um ciclo que visita cada vértice exatamente uma vez.

- **William Rowan Hamilton** (1805-1865) criou o "Icosian Game" em 1857, propondo um caminho que visitasse cada vértice de um dodecaedro uma única vez.
- **Definições básicas:**
  - **Caminho Hamiltoniano:** caminho que visita todos os vértices exatamente uma vez.
  - **Ciclo Hamiltoniano (ou circuito):** caminho Hamiltoniano que retorna ao vértice inicial.



# Introdução

## Aplicações Práticas dos Circuitos Hamiltonianos

- **Logística e Entregas:** Otimização de rotas de entrega, minimizando distâncias percorridas e maximizando eficiência operacional em sistemas de distribuição urbana.
- **Rotas Aéreas:** Planejamento de itinerários que conectem múltiplas cidades, otimizando combustível e tempo de voo em redes de transporte aéreo.

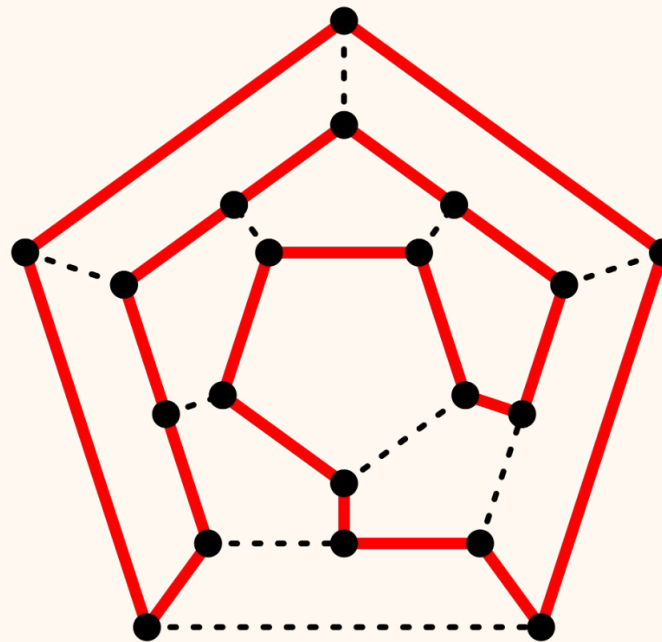
# Apresentação do Problema

- **Abordagem por Força Bruta:** Complexidade  **$O(n!)$** 
  - Testar todas as rotas possíveis tem um custo fatorial.
  - $n=10 \rightarrow \sim 180$  mil rotas |  $n=20 \rightarrow$  Um numero bem grande (Acho que 60. quatrilhoes) de rotas
  - **Conclusão:** Impraticável
- **Classificação:** NP-Completo
- **Não existe** um algoritmo eficiente conhecido ( $O(n)$  /  $O(n^2)$  /  $O(\log(n))$  ) para a solução exata!

# Apresentação do Problema

## Exemplos Clássicos e aplicações de Grafos Hamiltonianos

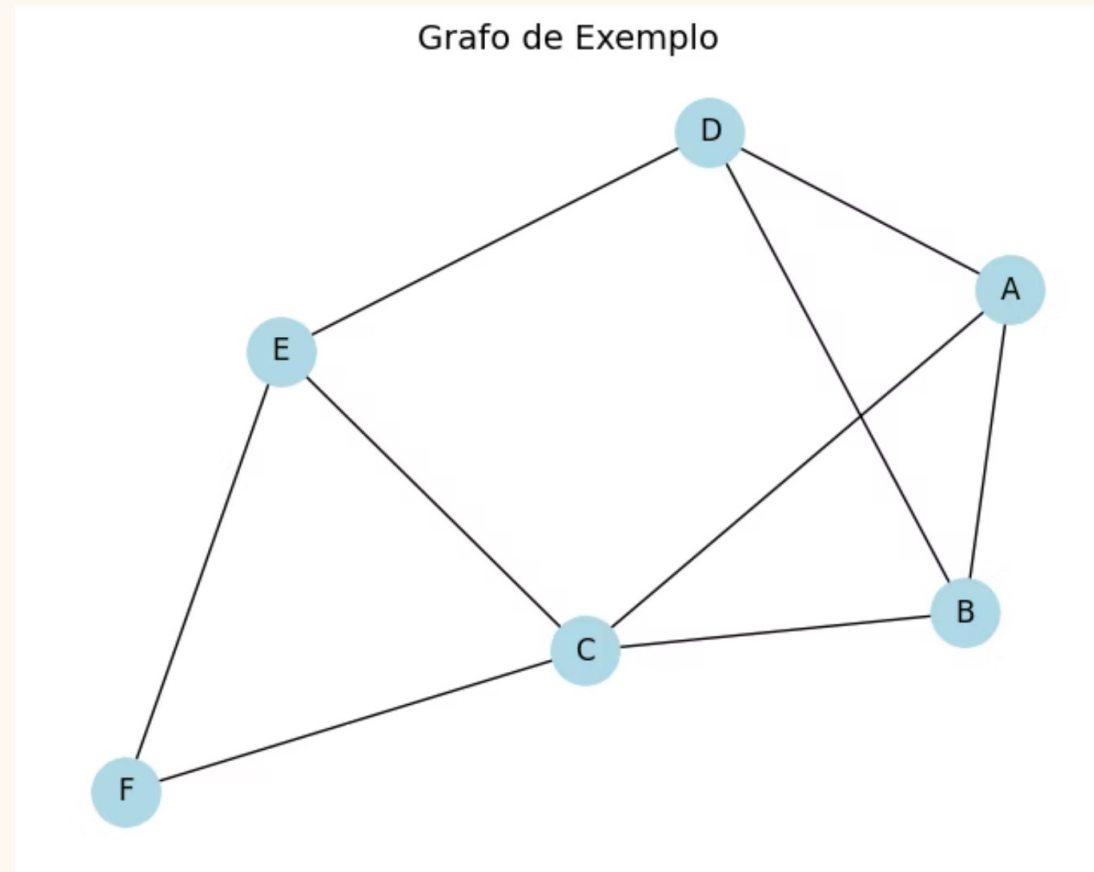
- **Sólidos Platônicos:** Todos os cinco sólidos platônicos (tetraedro, cubo, octaedro, dodecaedro, icosaedro) podem ser representados como grafos Hamiltonianos.



# Apresentação do Problema

## Exemplos

- OBS: Os exemplos foram criados em python! Em uma library chamada **networkx**

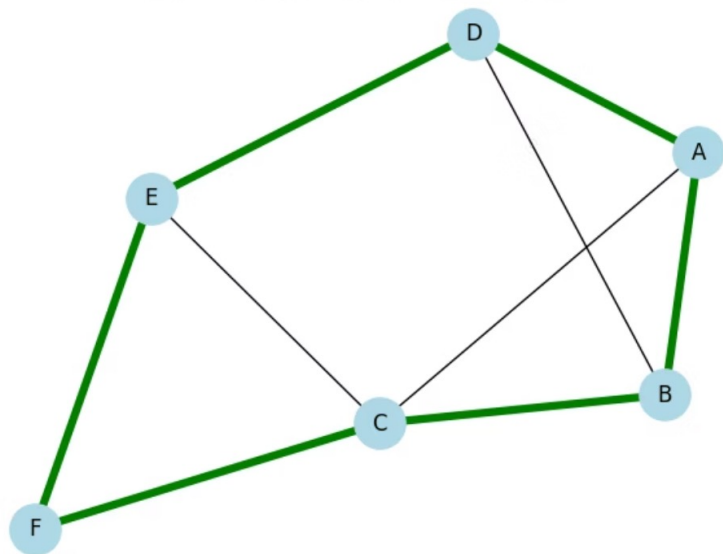


# Apresentação do Problema

## Exemplos

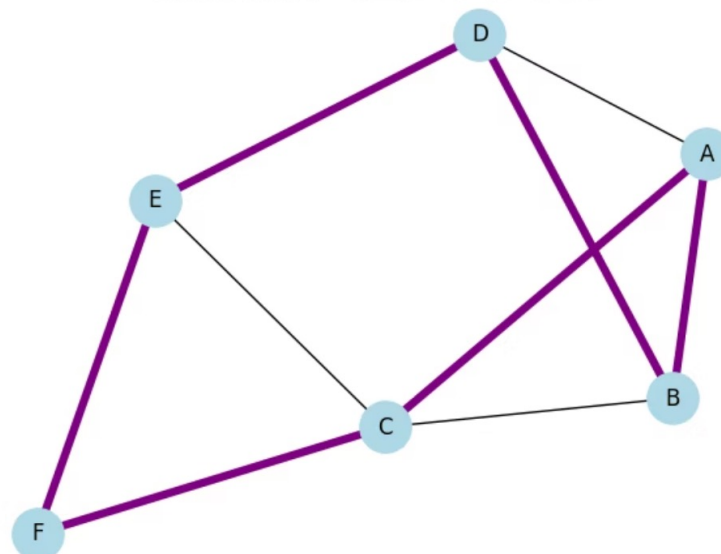
Exemplo-1 de Circuito Hamiltoniano (Válido)

Caminho Válido: A -> B -> C -> F -> E -> D -> A

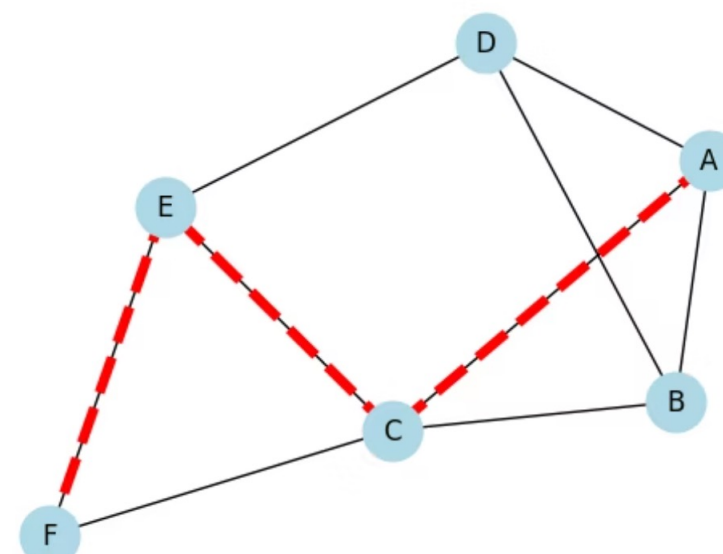


Exemplo-2 de Circuito Hamiltoniano (Válido)

Caminho Válido: A -> B -> D -> E -> F -> C -> A



Exemplo de Caminho Não Hamiltoniano (Inválido)



# Algoritmo de Backtracking

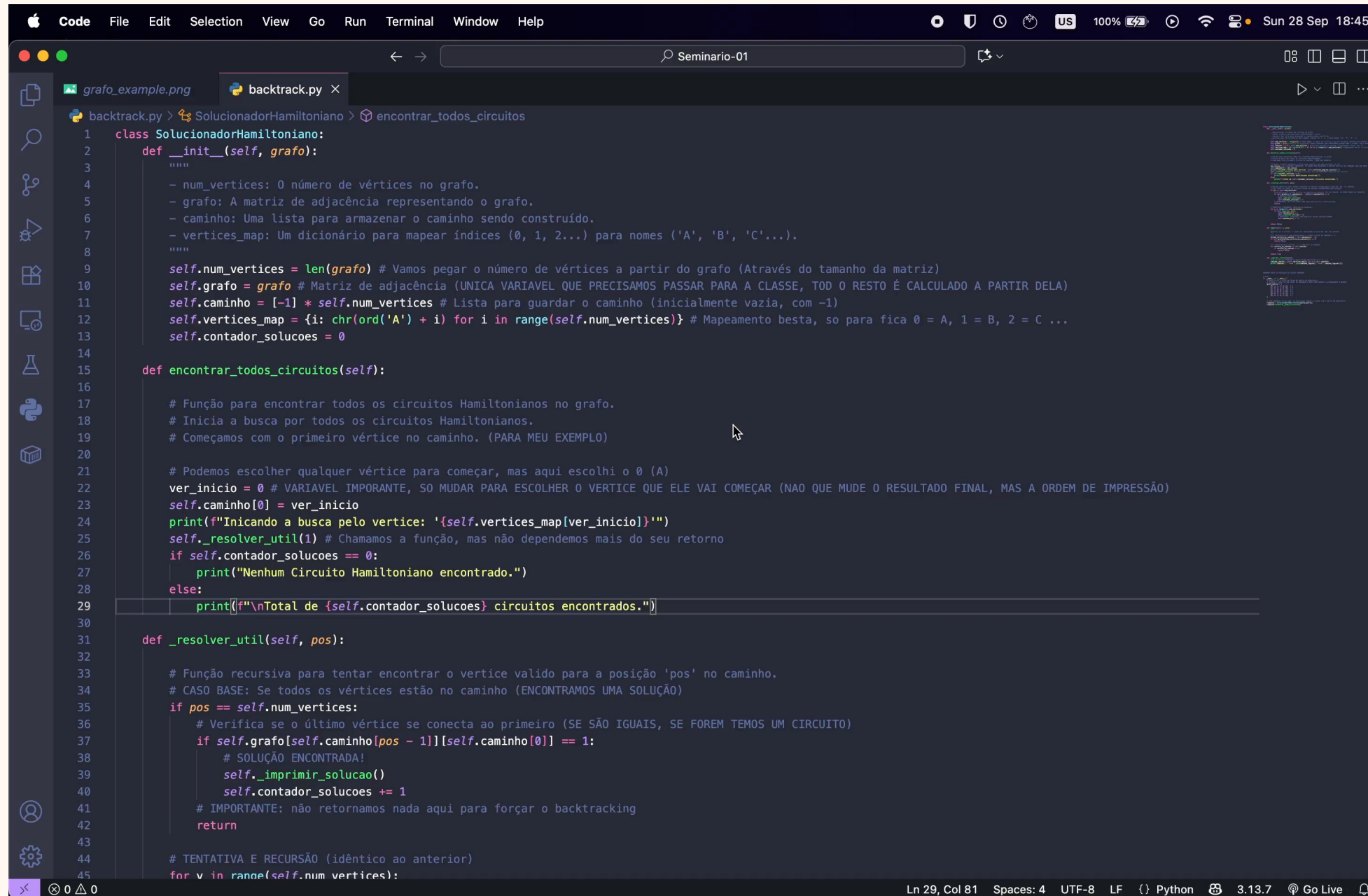
## A Lógica do Backtracking

O algoritmo de backtracking resolve o problema construindo o caminho vértice por vértice, seguindo uma estratégia simples e de tentativa e erro.

- **Início:** A partir do ponto atual, avance para um vizinho ainda não visitado.
- **Exploração:** Se o passo foi válido, repita o processo a partir do novo ponto, **construindo o caminho passo a passo**.
- **Impasse:** Se não houver nenhum caminho novo e válido para seguir, chegamos a um beco sem saída.
- **Backtrack:** Ao encontrar um impasse, ele não para. Ele volta para o ponto de decisão anterior e tenta uma rota diferente que ainda não foi explorada! E caso essa rota diferente já tenha sido toda explorada, voltamos outro ponto!



# Demonstração do algoritmo



```
backtrack.py > SolucionadorHamiltoniano > encontrar_todos_circuitos
1 class SolucionadorHamiltoniano:
2     def __init__(self, grafo):
3         """
4         - num_vertices: 0 número de vértices no grafo.
5         - grafo: A matriz de adjacência representando o grafo.
6         - caminho: Uma lista para armazenar o caminho sendo construído.
7         - vertices_map: Um dicionário para mapear índices (0, 1, 2...) para nomes ('A', 'B', 'C'...).
8         """
9         self.num_vertices = len(grafo) # Vamos pegar o número de vértices a partir do grafo (Através do tamanho da matriz)
10        self.grafo = grafo # Matriz de adjacência (UNICA VARIÁVEL QUE PRECISAMOS PASSAR PARA A CLASSE, TOD O RESTO É CALCULADO A PARTIR DELA)
11        self.caminho = [-1] * self.num_vertices # Lista para guardar o caminho (inicialmente vazia, com -1)
12        self.vertices_map = {i: chr(ord('A') + i) for i in range(self.num_vertices)} # Mapeamento besta, so para fica 0 = A, 1 = B, 2 = C ...
13        self.contador_solucoes = 0
14
15    def encontrar_todos_circuitos(self):
16
17        # Função para encontrar todos os circuitos Hamiltonianos no grafo.
18        # Inicia a busca por todos os circuitos Hamiltonianos.
19        # Começamos com o primeiro vértice no caminho. (PARA MEU EXEMPLO)
20
21        # Podemos escolher qualquer vértice para começar, mas aqui escolhi o 0 (A)
22        ver_inicio = 0 # VARIÁVEL IMPORANTE, SO MUDAR PARA ESCOLHER O VERTICE QUE ELE VAI COMEÇAR (NAO QUE MUDE O RESULTADO FINAL, MAS A ORDEM DE IMPRESSÃO)
23        self.caminho[0] = ver_inicio
24        print(f"Iniciando a busca pelo vertice: '{self.vertices_map[ver_inicio]}')")
25        self._resolver_util(1) # Chamamos a função, mas não dependemos mais do seu retorno
26        if self.contador_solucoes == 0:
27            print("Nenhum Circuito Hamiltoniano encontrado.")
28        else:
29            print(f"\nTotal de {self.contador_solucoes} circuitos encontrados.")
30
31    def _resolver_util(self, pos):
32
33        # Função recursiva para tentar encontrar o vertice valido para a posição 'pos' no caminho.
34        # CASO BASE: Se todos os vértices estão no caminho (ENCONTRAMOS UMA SOLUÇÃO)
35        if pos == self.num_vertices:
36            # Verifica se o último vértice se conecta ao primeiro (SE SÃO IGUAIS, SE FOREM TEMOS UM CIRCUITO)
37            if self.grafo[self.caminho[pos - 1]][self.caminho[0]] == 1:
38                # SOLUÇÃO ENCONTRADA!
39                self._imprimir_soluciao()
40                self.contador_solucoes += 1
41            # IMPORTANTE: não retornamos nada aqui para forçar o backtracking
42            return
43
44        # TENTATIVA E RECURSÃO (idêntico ao anterior)
45        for v in range(self.num_vertices):
```

# Conclusão

- O Circuito Hamiltoniano é um desafio fundamental sobre encontrar um "tour completo" em um grafo.
- É um problema **NP-Completo**, o que significa que não há solução eficiente conhecida para grandes escalas, tornando a força bruta inviável.
- O **Backtracking** oferece uma solução que explora o espaço de busca de forma "sagaz" com uma estratégia de "analisar, passos para frente e para trás"