AED2 - Algoritmos e Estr. de Dados II

Aula 26: Caminhos Eulerianos e Hamiltonianos

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br

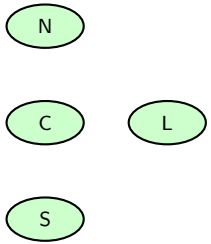
CEFET-MG - Campus Timoteo
Dep. Engenharia de Computação

Junho de 2026

Sumário

1. O Problema Histórico (1736)

A cidade de Königsberg (hoje Kaliningrado, Rússia) era cortada pelo rio Pregel e possuía **sete pontes**.

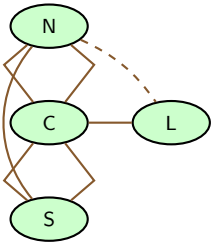


1. O Problema Histórico (1736)

A cidade de Königsberg (hoje Kaliningrado, Rússia) era cortada pelo rio Pregel e possuía **sete pontes**.

O Desafio

É possível fazer um passeio pela cidade cruzando **cada ponte exatamente uma vez**?



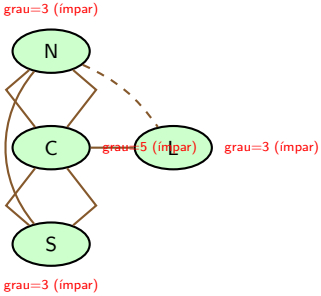
1. O Problema Histórico (1736)

A cidade de Königsberg (hoje Kaliningrado, Rússia) era cortada pelo rio Pregel e possuía **sete pontes**.

O Desafio

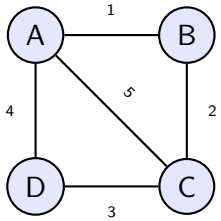
É possível fazer um passeio pela cidade cruzando **cada ponte exatamente uma vez**?

O matemático **Leonhard Euler** (1736) foi o primeiro a **provar matematicamente** que é **impossível** — e ao fazer isso, **criou a Teoria dos Grafos**.



Caminho Euleriano

Caminho que percorre **cada aresta exatamente uma vez**. Não precisa voltar ao início.



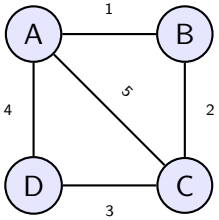
2. Definições: Caminho e Ciclo Euleriano

Caminho Euleriano

Caminho que percorre **cada aresta exatamente uma vez**. Não precisa voltar ao início.

Ciclo Euleriano

Caminho Euleriano que **começa e termina no mesmo vértice**. Percorre todas as arestas exatamente uma vez e fecha o laço.



Ciclo: A-B-C-A-D-C

2. Definições: Caminho e Ciclo Euleriano

Caminho Euleriano

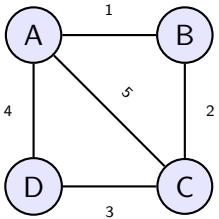
Caminho que percorre **cada aresta exatamente uma vez**. Não precisa voltar ao início.

Ciclo Euleriano

Caminho Euleriano que **começa e termina no mesmo vértice**. Percorre todas as arestas exatamente uma vez e fecha o laço.

Diferença crucial com Hamiltoniano

Euleriano $\Rightarrow$ todas as **arestas** uma vez.
Hamiltoniano $\Rightarrow$ todos os **vértices** uma vez.



Ciclo: A-B-C-A-D-C

3. Condição de Existência: O Teorema de Euler

Euler provou as condições **necessárias e suficientes**:

Ciclo Euleriano (grafo não-direcionado)

Caminho Euleriano (grafo não-direcionado)

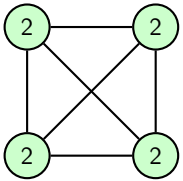
3. Condição de Existência: O Teorema de Euler

Euler provou as condições **necessárias e suficientes**:

Ciclo Euleriano (grafo não-direcionado)

Existe $\Leftrightarrow$ o grafo é **conexo** E **todos os vértices têm grau par**.

Caminho Euleriano (grafo não-direcionado)



Ciclo Euleriano ✓

3. Condição de Existência: O Teorema de Euler

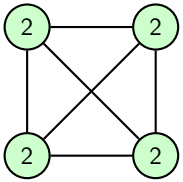
Euler provou as condições **necessárias e suficientes**:

Ciclo Euleriano (grafo não-direcionado)

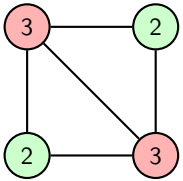
Existe $\Leftrightarrow$ o grafo é **conexo** E **todos os vértices têm grau par**.

Caminho Euleriano (grafo não-direcionado)

Existe $\Leftrightarrow$ o grafo é **conexo** E tem **exatamente 0 ou 2 vértices de grau ímpar**.



Ciclo Euleriano ✓



Caminho Euleriano ✓

3. Condição de Existência: O Teorema de Euler

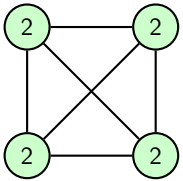
Euler provou as condições **necessárias e suficientes**:

Ciclo Euleriano (grafo não-direcionado)

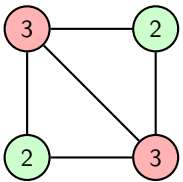
Existe $\Leftrightarrow$ o grafo é **conexo** E **todos os vértices têm grau par**.

Caminho Euleriano (grafo não-direcionado)

Existe $\Leftrightarrow$ o grafo é **conexo** E tem **exatamente 0 ou 2 vértices de grau ímpar**.
(Com 2 vértices ímpares, o caminho começa em um e termina no outro.)



Ciclo Euleriano ✓



Caminho Euleriano ✓

3. Condição de Existência: O Teorema de Euler

Euler provou as condições **necessárias e suficientes**:

Ciclo Euleriano (grafo não-direcionado)

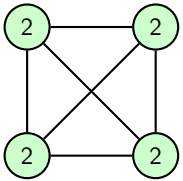
Existe $\Leftrightarrow$ o grafo é **conexo** E **todos os vértices têm grau par**.

Caminho Euleriano (grafo não-direcionado)

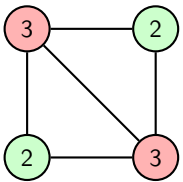
Existe $\Leftrightarrow$ o grafo é **conexo** E tem **exatamente 0 ou 2 vértices de grau ímpar**.
(Com 2 vértices ímpares, o caminho começa em um e termina no outro.)

Verificação em $O(V + E)$

Basta checar conectividade (BFS/DFS) e contar



Ciclo Euleriano ✓



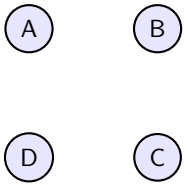
Caminho Euleriano ✓

4. Grafos Direcionados: Condição para Euler

Em grafos **direcionados** (dígrafos), a condição muda:

Circuito Euleriano (dígrafo)

Caminho Euleriano (dígrafo)



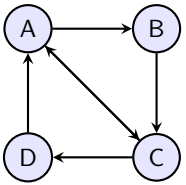
4. Grafos Direcionados: Condição para Euler

Em grafos **direcionados** (dígrafos), a condição muda:

Circuito Euleriano (dígrafo)

Existe $\Leftrightarrow$ o grafo é **fortemente conexo** E para cada vértice: grau-entrada $=$ grau-saída.

Caminho Euleriano (dígrafo)



in=out para todos ✓

4. Grafos Direcionados: Condição para Euler

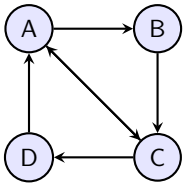
Em grafos **direcionados** (dígrafos), a condição muda:

Circuito Euleriano (dígrafo)

Existe $\Leftrightarrow$ o grafo é **fortemente conexo** E para cada vértice: grau-entrada $=$ grau-saída.

Caminho Euleriano (dígrafo)

Existe $\Leftrightarrow$ há um único vértice com saída $-$ entrada $= 1$ (origem) e um com entrada $-$ saída $= 1$ (destino), e os demais são equilibrados.



in=out para todos ✓

4. Grafos Direcionados: Condição para Euler

Em grafos **direcionados** (dígrafos), a condição muda:

Circuito Euleriano (dígrafo)

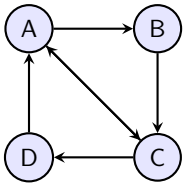
Existe $\Leftrightarrow$ o grafo é **fortemente conexo** E para cada vértice: grau-entrada $=$ grau-saída.

Caminho Euleriano (dígrafo)

Existe $\Leftrightarrow$ há um único vértice com saída $-$ entrada $= 1$ (origem) e um com entrada $-$ saída $= 1$ (destino), e os demais são equilibrados.

Aplicação clássica

Sequências de De Bruijn: encontrar a menor

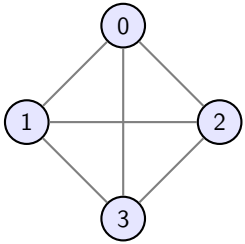


in=out para todos ✓

5. Algoritmo de Hierholzer: Ideia Central

Uma vez que sabemos que o ciclo Euleriano **existe**, como **encontrá-lo** em $O(E)$?

Hierholzer (1873) — Ideia

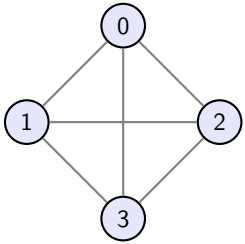


5. Algoritmo de Hierholzer: Ideia Central

Uma vez que sabemos que o ciclo Euleriano **existe**, como **encontrá-lo** em $O(E)$?

Hierholzer (1873) — Ideia

1. Comece de um vértice qualquer v.

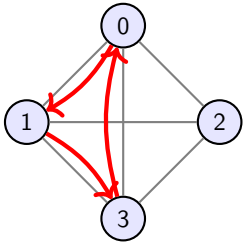


5. Algoritmo de Hierholzer: Ideia Central

Uma vez que sabemos que o ciclo Euleriano **existe**, como **encontrá-lo** em $O(E)$?

Hierholzer (1873) — Ideia

1. Comece de um vértice qualquer v.
2. Siga arestas **arbitrariamente**, removendo-as, até voltar a v (sub-ciclo).



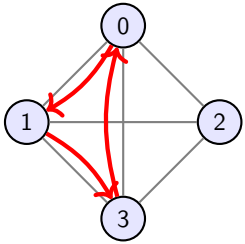
Sub-ciclo: 0-1-3-0

5. Algoritmo de Hierholzer: Ideia Central

Uma vez que sabemos que o ciclo Euleriano **existe**, como **encontrá-lo** em $O(E)$?

Hierholzer (1873) — Ideia

1. Comece de um vértice qualquer v.
2. Siga arestas **arbitrariamente**, removendo-as, até voltar a v (sub-ciclo).
3. Se sobrar arestas não visitadas, elas estão conectadas a algum vértice do ciclo atual. Encontre esse vértice e expanda um novo sub-ciclo a partir dele.



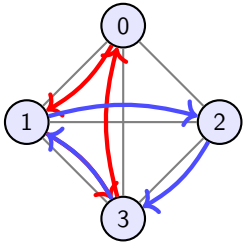
Sub-ciclo: 0-1-3-0

5. Algoritmo de Hierholzer: Ideia Central

Uma vez que sabemos que o ciclo Euleriano **existe**, como **encontrá-lo** em $O(E)$?

Hierholzer (1873) — Ideia

1. Comece de um vértice qualquer v.
2. Siga arestas **arbitrariamente**, removendo-as, até voltar a v (sub-ciclo).
3. Se sobrar arestas não visitadas, elas estão conectadas a algum vértice do ciclo atual. Encontre esse vértice e expanda um novo sub-ciclo a partir dele.
4. **Funda** os sub-ciclos ao ciclo principal.



Sub-ciclo: 0-1-3-0

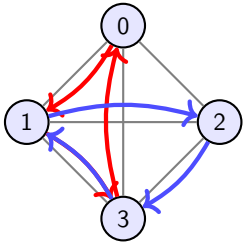
Expande em 1

5. Algoritmo de Hierholzer: Ideia Central

Uma vez que sabemos que o ciclo Euleriano **existe**, como **encontrá-lo** em $O(E)$?

Hierholzer (1873) — Ideia

1. Comece de um vértice qualquer v.
2. Siga arestas **arbitrariamente**, removendo-as, até voltar a v (sub-ciclo).
3. Se sobrar arestas não visitadas, elas estão conectadas a algum vértice do ciclo atual. Encontre esse vértice e expanda um novo sub-ciclo a partir dele.
4. **Funda** os sub-ciclos ao ciclo principal.
5. Repita até não sobrar arestas.



Sub-ciclo: 0-1-3-0

Expande em 1

6. Implementação do Hierholzer em Java

```java
import java.util.*;

public class Hierholzer {
    // adj: lista de adjacencia usando indice de aresta
    static List<int[]>[] adj; // adj[u] = lista de {v, id_aresta}
    static boolean[] usado;   // aresta usada?
    static LinkedList<Integer> ciclo = new LinkedList<>();

    static void dfs(int u) {
        while (!adj[u].isEmpty()) {
            int[] aresta = adj[u].remove(adj[u].size() - 1);
            int v = aresta[0], id = aresta[1];
            if (usado[id]) continue; // aresta ja usada
            usado[id] = true;
            dfs(v);
        }
        ciclo.addFirst(u); // insere na frente ao retornar
    }

    public static List<Integer> eulerianCircuit(int V, int[][] arestas) {
        adj = new List[V];
        for (int i = 0; i < V; i++) adj[i] = new ArrayList<>();
```

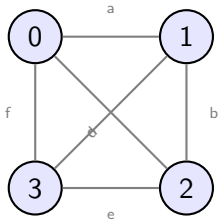
6. Implementação do Hierholzer em Java

```java
import java.util.*;

public class Hierholzer {
    // adj: lista de adjacencia usando indice de aresta
    static List<int[]>[] adj; // adj[u] = lista de {v, id_aresta}
    static boolean[] usado;   // aresta usada?
    static LinkedList<Integer> ciclo = new LinkedList<>();

    static void dfs(int u) {
        while (!adj[u].isEmpty()) {
            int[] aresta = adj[u].remove(adj[u].size() - 1);
            int v = aresta[0], id = aresta[1];
            if (usado[id]) continue; // aresta ja usada
            usado[id] = true;
            dfs(v);
        }
        ciclo.addFirst(u); // insere na frente ao retornar
    }

    public static List<Integer> eulerianCircuit(int V, int[][] arestas) {
        adj = new List[V];
        for (int i = 0; i < V; i++) adj[i] = new ArrayList<>();
```

7. Hierholzer – Trace Animado

Grafo com 4 vértices e 6 arestas (todos de grau par $\Rightarrow$ ciclo Euleriano existe).

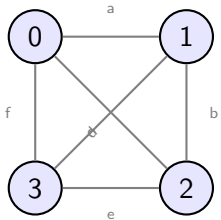
Passo	Ação



7. Hierholzer – Trace Animado

Grafo com 4 vértices e 6 arestas (todos de grau par $\Rightarrow$ ciclo Euleriano existe).

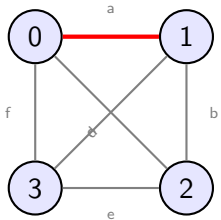
Passo	**Ação**
Init	Começa em **0**, pilha = [0]



7. Hierholzer – Trace Animado

Grafo com 4 vértices e 6 arestas (todos de grau par $\Rightarrow$ ciclo Euleriano existe).

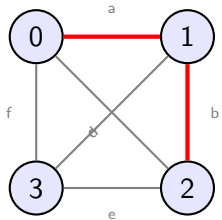
Passo	Ação
Init	Começa em **0**, pilha = [0]
1	$0 \rightarrow 1$, pilha = [0,1]



7. Hierholzer – Trace Animado

Grafo com 4 vértices e 6 arestas (todos de grau par $\Rightarrow$ ciclo Euleriano existe).

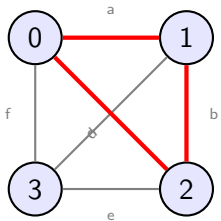
Passo	Ação
Init	Começa em **0**, pilha = [0]
1	$0 \rightarrow 1$, pilha = [0,1]
2	$1 \rightarrow 2$, pilha = [0,1,2]



7. Hierholzer – Trace Animado

Grafo com 4 vértices e 6 arestas (todos de grau par $\Rightarrow$ ciclo Euleriano existe).

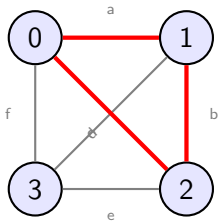
Passo	Ação
Init	Começa em **0**, pilha = [0]
1	$0 \rightarrow 1$, pilha = [0,1]
2	$1 \rightarrow 2$, pilha = [0,1,2]
3	$2 \rightarrow 0$, pilha = [0,1,2,0] **(sub-ciclo!)**



7. Hierholzer – Trace Animado

Grafo com 4 vértices e 6 arestas (todos de grau par $\Rightarrow$ ciclo Euleriano existe).

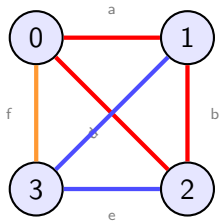
Passo	Ação
Init	Começa em **0**, pilha = [0]
1	$0 \rightarrow 1$, pilha = [0,1]
2	$1 \rightarrow 2$, pilha = [0,1,2]
3	$2 \rightarrow 0$, pilha = [0,1,2,0] **(sub-ciclo!)**
4	Retorna a 1 (tem mais arestas)



7. Hierholzer – Trace Animado

Grafo com 4 vértices e 6 arestas (todos de grau par $\Rightarrow$ ciclo Euleriano existe).

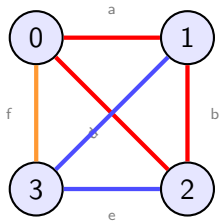
Passo	Ação
Init	Começa em **0**, pilha = [0]
1	$0 \to 1$, pilha = [0,1]
2	$1 \to 2$, pilha = [0,1,2]
3	$2 \to 0$, pilha = [0,1,2,0] **(sub-ciclo!)**
4	Retorna a 1 (tem mais arestas)
5	$1 \to 3$, $3 \to 2$, $2 \to 1$ **(funde)**



7. Hierholzer – Trace Animado

Grafo com 4 vértices e 6 arestas (todos de grau par $\Rightarrow$ ciclo Euleriano existe).

Passo	Ação
Init	Começa em **0**, pilha = [0]
1	$0 \rightarrow 1$, pilha = [0,1]
2	$1 \rightarrow 2$, pilha = [0,1,2]
3	$2 \rightarrow 0$, pilha = [0,1,2,0] **(sub-ciclo!)**
4	Retorna a 1 (tem mais arestas)
5	$1 \rightarrow 3$, $3 \rightarrow 2$, $2 \rightarrow 1$ **(funde)**
Fim	Ciclo: **0-1-3-2-1-2-0**

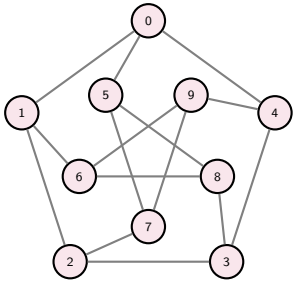


Resultado

Todas as **6 arestas** percorridas exatamente uma vez!

8. O Jogo Icosiano de Hamilton (1857)

William Rowan Hamilton (matemático irlandês) propôs em 1857 o "Icosian Game":

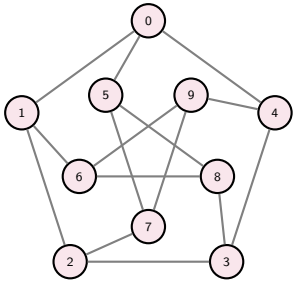


8. O Jogo Icosiano de Hamilton (1857)

William Rowan Hamilton (matemático irlandês) propôs em 1857 o "Icosian Game":

Desafio

Num dodecaedro (grafo de 20 vértices representando cidades), encontrar um ciclo que passe por **todos os vértices exatamente uma vez**.



8. O Jogo Icosiano de Hamilton (1857)

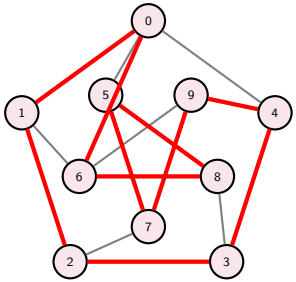
William Rowan Hamilton (matemático irlandês) propôs em 1857 o "Icosian Game":

Desafio

Num dodecaedro (grafo de 20 vértices representando cidades), encontrar um ciclo que passe por **todos os vértices exatamente uma vez**.

Definição: Ciclo Hamiltoniano

Ciclo que visita **cada vértice exatamente uma vez** e retorna ao início.
Caminho Hamiltoniano: sem obrigação de retornar.



8. O Jogo Icosiano de Hamilton (1857)

William Rowan Hamilton (matemático irlandês) propôs em 1857 o "Icosian Game":

Desafio

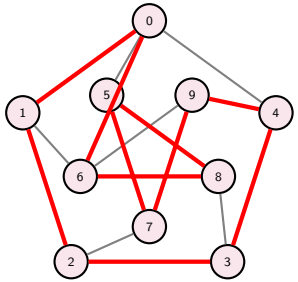
Num dodecaedro (grafo de 20 vértices representando cidades), encontrar um ciclo que passe por **todos os vértices exatamente uma vez**.

Definição: Ciclo Hamiltoniano

Ciclo que visita **cada vértice exatamente uma vez** e retorna ao início.
Caminho Hamiltoniano: sem obrigação de retornar.

O Grande Abismo

Enquanto Euleriano é $O(E)$, verificar se um grafo tem



9. Por Que é Tão Difícil? Euleriano vs. Hamiltoniano

Característica	Euleriano	Hamiltoniano

9. Por Que é Tão Difícil? Euleriano vs. Hamiltoniano

Característica	**Euleriano**	**Hamiltoniano**
O que percorre	Cada **aresta** 1x	Cada **vértice** 1x

9. Por Que é Tão Difícil? Euleriano vs. Hamiltoniano

Característica	**Euleriano**	**Hamiltoniano**
O que percorre	Cada **aresta** 1x	Cada **vértice** 1x
Condição de existência	Sim (graus)	**Não (geral)**

9. Por Que é Tão Difícil? Euleriano vs. Hamiltoniano

Característica	**Euleriano**	**Hamiltoniano**
O que percorre	Cada **aresta** 1x	Cada **vértice** 1x
Condição de existência	Sim (graus)	**Não (geral)**
Complexidade (encontrar)	$O(E)$ (Hierholzer)	$O(2^N \cdot N)$ (DP)

9. Por Que é Tão Difícil? Euleriano vs. Hamiltoniano

Característica	**Euleriano**	**Hamiltoniano**
O que percorre	Cada **aresta** 1x	Cada **vértice** 1x
Condição de existência	Sim (graus)	**Não (geral)**
Complexidade (encontrar)	$O(E)$ (Hierholzer)	$O(2^N \cdot N)$ (DP)
Classe do problema	P	**NP-Completo**

9. Por Que é Tão Difícil? Euleriano vs. Hamiltoniano

Característica	Euleriano	Hamiltoniano
O que percorre	Cada **aresta** 1x	Cada **vértice** 1x
Condição de existência	Sim (graus)	**Não (geral)**
Complexidade (encontrar)	$O(E)$ (Hierholzer)	$O(2^N \cdot N)$ (DP)
Classe do problema	P	**NP-Completo**
Critério suficiente	Grau par (todos)	Ore, Dirac...

9. Por Que é Tão Difícil? Euleriano vs. Hamiltoniano

Característica	Euleriano	Hamiltoniano
O que percorre	Cada **aresta** 1x	Cada **vértice** 1x
Condição de existência	Sim (graus)	**Não (geral)**
Complexidade (encontrar)	$O(E)$ (Hierholzer)	$O(2^N \cdot N)$ (DP)
Classe do problema	P	**NP-Completo**
Critério suficiente	Grau par (todos)	Ore, Dirac...
Aplicação típica	Rotas, DNA	TSP, agendamento

9. Por Que é Tão Difícil? Euleriano vs. Hamiltoniano

Característica	Euleriano	Hamiltoniano
O que percorre	Cada **aresta** 1x	Cada **vértice** 1x
Condição de existência	Sim (graus)	**Não (geral)**
Complexidade (encontrar)	$O(E)$ (Hierholzer)	$O(2^N \cdot N)$ (DP)
Classe do problema	P	**NP-Completo**
Critério suficiente	Grau par (todos)	Ore, Dirac...
Aplicação típica	Rotas, DNA	TSP, agendamento

A Lição Fundamental

Uma pequena mudança na definição do problema pode mudar sua classe de complexidade de **polinomial** para **NP-Completo**. Reconhecer esta diferença é essencial em algoritmos!

10. Condições Suficientes (mas Não Necessárias)

Apesar de não existir um critério "se e somente se" geral, existem **condições suficientes**:

Teorema de Dirac (1952)

Teorema de Ore (1960)

10. Condições Suficientes (mas Não Necessárias)

Apesar de não existir um critério "se e somente se" geral, existem **condições suficientes**:

Teorema de Dirac (1952)

Se G tem $n \geq 3$ vértices e **todo vértice tem grau** $\geq n/2$, então G tem Ciclo Hamiltoniano.

Teorema de Ore (1960)

10. Condições Suficientes (mas Não Necessárias)

Apesar de não existir um critério "se e somente se" geral, existem **condições suficientes**:

Teorema de Dirac (1952)

Se G tem $n \geq 3$ vértices e **todo vértice tem grau** $\geq n/2$, então G tem Ciclo Hamiltoniano.

Teorema de Ore (1960)

Se para todo par de vértices **não-adjacentes** u, v: $\deg(u) + \deg(v) \geq n$, então existe Ciclo Hamiltoniano.

(Dirac é caso especial de Ore: $n/2 + n/2 = n$.)

10. Condições Suficientes (mas Não Necessárias)

Apesar de não existir um critério "se e somente se" geral, existem **condições suficientes**:

Teorema de Dirac (1952)

Se G tem $n \geq 3$ vértices e **todo vértice tem grau** $\geq n/2$, então G tem Ciclo Hamiltoniano.

Teorema de Ore (1960)

Se para todo par de vértices **não-adjacentes** u, v: $\deg(u) + \deg(v) \geq n$, então existe Ciclo Hamiltoniano.

(Dirac é caso especial de Ore: $n/2 + n/2 = n$.)

Cuidado!

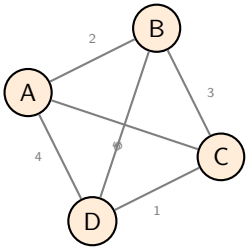
Estas condições são **suficientes, não necessárias**: um grafo pode ter Ciclo Hamiltoniano mesmo sem satisfazê-las. Um grafo com n vértices e n arestas em ciclo é Hamiltoniano mas não satisfaz Dirac.

11. O Problema do Caixeiro Viajante (TSP)

O **TSP (Traveling Salesman Problem)** é a versão otimizada do Hamiltoniano:

Problema

Dado um grafo completo com pesos nas arestas, encontrar o **Ciclo Hamiltoniano de menor peso total**.



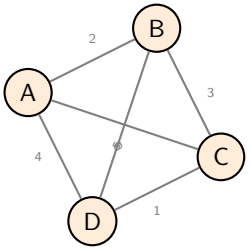
11. O Problema do Caixeiro Viajante (TSP)

O **TSP (Traveling Salesman Problem)** é a versão otimizada do Hamiltoniano:

Problema

Dado um grafo completo com pesos nas arestas, encontrar o **Ciclo Hamiltoniano de menor peso total**.

Por que é hard? Há $(N-1)!/2$ ciclos distintos. Para $N = 20$: $\approx 6 \times 10^{16}$ possibilidades. Força bruta é inviável!



11. O Problema do Caixeiro Viajante (TSP)

O **TSP (Traveling Salesman Problem)** é a versão otimizada do Hamiltoniano:

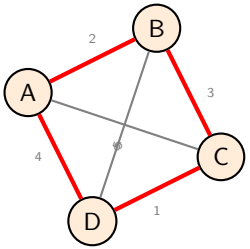
Problema

Dado um grafo completo com pesos nas arestas, encontrar o **Ciclo Hamiltoniano de menor peso total**.

Por que é hard? Há $(N-1)!/2$ ciclos distintos. Para $N = 20$: $\approx 6 \times 10^{16}$ possibilidades. Força bruta é inviável!

Abordagens por tamanho de N

- $N \leq 20$: **DP com Bitmask** — $O(2^N \cdot N^2)$
- $N \leq 12$: Backtracking com poda
- N grande: Heurísticas (2-OPT, Ant Colony



Custo: 2+3+1+4 = 10

12. DP com Bitmask: A Ideia

A ideia genial: usar um **inteiro como máscara de bits** para representar o conjunto de cidades visitadas.

Estado

12. DP com Bitmask: A Ideia

A ideia genial: usar um **inteiro como máscara de bits** para representar o conjunto de cidades visitadas.

Estado

`dp[mask][u]` = menor custo para visitar exatamente as cidades indicadas por `mask`, terminando na cidade `u`.

Exemplo com N=3

`mask = 110` (binário)
$\Rightarrow$ cidades 1 e 2 visitadas.

`dp[110][1]` = menor custo visitando {1,2} terminando em 1.

`dp[110][2]` = menor custo visitando {1,2} terminando em 2.

12. DP com Bitmask: A Ideia

A ideia genial: usar um **inteiro como máscara de bits** para representar o conjunto de cidades visitadas.

Estado

`dp[mask][u]` = menor custo para visitar exatamente as cidades indicadas por `mask`, terminando na cidade `u`.

Transição

$$dp[\text{mask}][u] = \min_{v \in \text{mask}, v \neq u} \left(dp[\text{mask} \setminus \{u\}][v] + w(v, u) \right)$$

Exemplo com N=3

`mask = 110` (binário)
$\Rightarrow$ cidades 1 e 2 visitadas.

`dp[110][1]` = menor custo visitando {1,2} terminando em 1.

`dp[110][2]` = menor custo visitando {1,2} terminando em 2.

12. DP com Bitmask: A Ideia

A ideia genial: usar um **inteiro como máscara de bits** para representar o conjunto de cidades visitadas.

Estado

`dp[mask][u]` = menor custo para visitar exatamente as cidades indicadas por `mask`, terminando na cidade `u`.

Transição

$$dp[\text{mask}][u] = \min_{v \in \text{mask}, v \neq u} \left(dp[\text{mask} \setminus \{u\}][v] + w(v, u) \right)$$

Exemplo com N=3

`mask = 110` (binário)
$\Rightarrow$ cidades 1 e 2 visitadas.

`dp[110][1]` = menor custo visitando {1,2} terminando em 1.

`dp[110][2]` = menor custo visitando {1,2} terminando em 2.

Complexidade

Estados: $2^N \times N$

Transições: $O(N)$ por estado

13. Bitmask: Operações Essenciais

Para implementar a DP, precisamos manipular bits:

Operações Fundamentais

Operação	Java

13. Bitmask: Operações Essenciais

Para implementar a DP, precisamos manipular bits:

Operações Fundamentais

Operação	**Java**
Bit *i* ativo?	`(mask & (1<<i)) != 0`

13. Bitmask: Operações Essenciais

Para implementar a DP, precisamos manipular bits:

Operações Fundamentais

Operação	**Java**
Bit *i* ativo?	`(mask & (1<<i)) != 0`
Ativar bit *i*	`mask | (1<<i)`

13. Bitmask: Operações Essenciais

Para implementar a DP, precisamos manipular bits:

Operações Fundamentais

Operação	**Java**
Bit *i* ativo?	`(mask & (1<<i)) != 0`
Ativar bit *i*	`mask | (1<<i)`
Desativar bit *i*	`mask ^ (1<<i)`

13. Bitmask: Operações Essenciais

Para implementar a DP, precisamos manipular bits:

Operações Fundamentais

Operação	**Java**
Bit *i* ativo?	`(mask & (1<<i)) != 0`
Ativar bit *i*	`mask | (1<<i)`
Desativar bit *i*	`mask ^ (1<<i)`
Máscara completa	`(1<<N) - 1`

13. Bitmask: Operações Essenciais

Para implementar a DP, precisamos manipular bits:

Operações Fundamentais

Operação	**Java**
Bit i ativo?	`(mask & (1<<i)) != 0`
Ativar bit i	`mask | (1<<i)`
Desativar bit i	`mask ^ (1<<i)`
Máscara completa	`(1<<N) - 1`
Contar bits ativos	`Integer.bitCount(mask)`

13. Bitmask: Operações Essenciais

Para implementar a DP, precisamos manipular bits:

Operações Fundamentais

Operação	Java
Bit *i* ativo?	`(mask & (1<<i)) != 0`
Ativar bit *i*	`mask | (1<<i)`
Desativar bit *i*	`mask ^ (1<<i)`
Máscara completa	`(1<<N) - 1`
Contar bits ativos	`Integer.bitCount(mas`

Trace para N=3, mask=101

- `mask = 5` (binário: 101)
- Cidades visitadas: $\{0, 2\}$
- Cidade 1 visitada? `5 & 2 = 0` $\Rightarrow$ Não
- Remover cidade 2: `5 ^ 4 = 1` (001)
- ALL = `(1<<3)-1 = 7` (111)

14. Implementação Java: TSP com DP Bitmask

```java
public class TSP {
    static final int INF = Integer.MAX_VALUE / 2;

    public static int solveTSP(int[][] dist) {
        int N = dist.length;
        int ALL = (1 << N) - 1;
        int[][] dp = new int[1 << N][N];

        for (int[] row : dp) Arrays.fill(row, INF);
        dp[1][0] = 0; // comeca na cidade 0, mask=0...01

        for (int mask = 1; mask <= ALL; mask++) {
            for (int u = 0; u < N; u++) {
                if ((mask & (1 << u)) == 0) continue; // u nao esta em mask
                if (dp[mask][u] == INF) continue;

                // Tenta ir para cidade v ainda nao visitada
                for (int v = 0; v < N; v++) {
                    if ((mask & (1 << v)) != 0) continue; // v ja visitado
                    int newMask = mask | (1 << v);
                    dp[newMask][v] = Math.min(dp[newMask][v],
                                              dp[mask][u] + dist[u][v]);
                }
            }
        }
        // Fecha o ciclo voltando a cidade 0
        int ans = INF;
        for (int u = 1; u < N; u++)
            if (dp[ALL][u] != INF)
                ans = Math.min(ans, dp[ALL][u] + dist[u][0]);
        return ans;
    }
}
```

14. Implementação Java: TSP com DP Bitmask

```java
public class TSP {
    static final int INF = Integer.MAX_VALUE / 2;

    public static int solveTSP(int[][] dist) {
        int N = dist.length;
        int ALL = (1 << N) - 1;
        int[][] dp = new int[1 << N][N];

        for (int[] row : dp) Arrays.fill(row, INF);
        dp[1][0] = 0; // comeca na cidade 0, mask=0...01

        for (int mask = 1; mask <= ALL; mask++) {
            for (int u = 0; u < N; u++) {
                if ((mask & (1 << u)) == 0) continue; // u nao esta em mask
                if (dp[mask][u] == INF) continue;

                // Tenta ir para cidade v ainda nao visitada
                for (int v = 0; v < N; v++) {
                    if ((mask & (1 << v)) != 0) continue; // v ja visitado
                    int newMask = mask | (1 << v);
                    dp[newMask][v] = Math.min(dp[newMask][v],
                                              dp[mask][u] + dist[u][v]);
                }
            }
        }
        // Fecha o ciclo voltando a cidade 0
        int ans = INF;
        for (int u = 1; u < N; u++)
            if (dp[ALL][u] != INF)
                ans = Math.min(ans, dp[ALL][u] + dist[u][0]);
        return ans;
    }
}
```

Memória

$2^{20} \times 20 \times 4$ bytes ≈ 80 MB para $N = 20$. Cuidado com o limite de memória!

15. Reconstruindo o Caminho no TSP

Para **imprimir o caminho** (não só o custo), guardamos o vetor de "pais":

```
int[][] pai = new int[1 << N][N];
for (int[] row : pai) Arrays.fill(row, -1);
// ... (mesma DP, mas ao atualizar dp[newMask][v], salva pai[newMask][v] = u)

// Reconstrucao: começa do ultimo vertice e volta
List<Integer> caminho = new ArrayList<>();
int mask = ALL, cur = melhorUltimo; // melhorUltimo = u que minimizou a resposta
while (cur != -1) {
    caminho.add(cur);
    int prev = pai[mask][cur];
    mask ^= (1 << cur);
    cur = prev;
}
Collections.reverse(caminho);
caminho.add(0); // fecha o ciclo
```

15. Reconstruindo o Caminho no TSP

Para **imprimir o caminho** (não só o custo), guardamos o vetor de "pais":

```
int[][] pai = new int[1 << N][N];
for (int[] row : pai) Arrays.fill(row, -1);
// ... (mesma DP, mas ao atualizar dp[newMask][v], salva pai[newMask][v] = u)

// Reconstrucao: começa do ultimo vertice e volta
List<Integer> caminho = new ArrayList<>();
int mask = ALL, cur = melhorUltimo; // melhorUltimo = u que minimizou a resposta
while (cur != -1) {
    caminho.add(cur);
    int prev = pai[mask][cur];
    mask ^= (1 << cur);
    cur = prev;
}
Collections.reverse(caminho);
caminho.add(0); // fecha o ciclo
```

Trace para $N = 4$

Suponha que a resposta seja $0 \to 2 \to 3 \to 1 \to 0$. A reconstrução começa em 1 com `mask=1111`, segue `pai[1111][1]=3`, depois `pai[1011][3]=2`, `pai[0011][2]=0`.

16. Quando N é Grande: Heurísticas e Aproximações

Para instâncias grandes do TSP ($N > 20$), usamos heurísticas:

Vizinho Mais Próximo (Greedy)

2-OPT (Melhoria local)

16. Quando N é Grande: Heurísticas e Aproximações

Para instâncias grandes do TSP ($N > 20$), usamos heurísticas:

Vizinho Mais Próximo (Greedy)

1. Comece numa cidade qualquer.
2. Sempre vá para a cidade mais próxima ainda não visitada.
3. Volte ao início.

Garantia: Nenhuma. Pode ser bem ruim.
Complexidade: $O(N^2)$.

2-OPT (Melhoria local)

16. Quando N é Grande: Heurísticas e Aproximações

Para instâncias grandes do TSP ($N > 20$), usamos heurísticas:

Vizinho Mais Próximo (Greedy)

1. Comece numa cidade qualquer.
2. Sempre vá para a cidade mais próxima ainda não visitada.
3. Volte ao início.

Garantia: Nenhuma. Pode ser bem ruim.
Complexidade: $O(N^2)$.

2-OPT (Melhoria local)

- Parte de qualquer ciclo Hamiltoniano.
- Testa trocas: remove 2 arestas e reconecta de outro jeito.
- Repete até não melhorar mais.

Garantia: Ótimo local (não global).
Complexidade: $O(N^2)$ por iteração.

16. Quando N é Grande: Heurísticas e Aproximações

Para instâncias grandes do TSP ($N > 20$), usamos heurísticas:

Vizinho Mais Próximo (Greedy)

1. Comece numa cidade qualquer.
2. Sempre vá para a cidade mais próxima ainda não visitada.
3. Volte ao início.

Garantia: Nenhuma. Pode ser bem ruim.
Complexidade: $O(N^2)$.

2-OPT (Melhoria local)

- Parte de qualquer ciclo Hamiltoniano.
- Testa trocas: remove 2 arestas e reconecta de outro jeito.
- Repete até não melhorar mais.

Garantia: Ótimo local (não global).
Complexidade: $O(N^2)$ por iteração.

Aproximação com Garantia: Algoritmo de Christofides

Para TSP métrico (com desigualdade triangular), Christofides garante solução no máximo $1.5\times$ o ótimo. Usa MST + matching de peso mínimo. *(Complexidade: $O(N^3)$)*

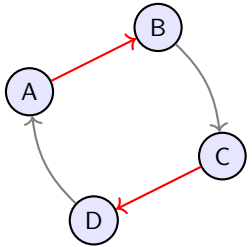
Operação 2-OPT

Dadas arestas (A, B) e (C, D) no ciclo atual:

Antes: $\ldots A \to B \ldots C \to D \ldots$

Depois: $\ldots A \to C \ldots B \to D \ldots$

Troca se:
$d(A, C) + d(B, D) < d(A, B) + d(C, D)$



Antes (cruzando)

17. Visualizando 2-OPT

Operação 2-OPT

Dadas arestas (A, B) e (C, D) no ciclo atual:

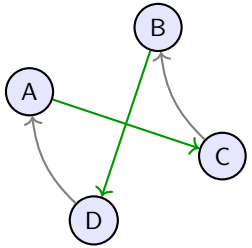
Antes: $\ldots A \to B \ldots C \to D \ldots$

Depois: $\ldots A \to C \ldots B \to D \ldots$

Troca se:
$d(A, C) + d(B, D) < d(A, B) + d(C, D)$

Implementação

- Dois loops: $O(N^2)$ pares de arestas.
- Se melhora: reverte o segmento entre B e C.
- Repete enquanto houver melhora.



Depois (sem cruzamento)

18. Aplicações de Euleriano: Exemplos Reais

1. Roteiro de Inspeção de Ruas

3. Planejamento de Trilhas

2. Sequenciamento de DNA

4. Traçado de Circuitos (PCB)

18. Aplicações de Euleriano: Exemplos Reais

1. Roteiro de Inspeção de Ruas

Um gari deve varrer todas as ruas de um bairro exatamente uma vez. Modelagem: ruas = arestas, cruzamentos = vértices. Se existe Circuito Euleriano, perfeito. Senão, encontrar o **Caminho Chinês do Carteiro** (duplicar arestas para equilibrar graus).

2. Sequenciamento de DNA

3. Planejamento de Trilhas

4. Traçado de Circuitos (PCB)

18. Aplicações de Euleriano: Exemplos Reais

1. Roteiro de Inspeção de Ruas

Um gari deve varrer todas as ruas de um bairro exatamente uma vez. Modelagem: ruas = arestas, cruzamentos = vértices. Se existe Circuito Euleriano, perfeito. Senão, encontrar o **Caminho Chinês do Carteiro** (duplicar arestas para equilibrar graus).

2. Sequenciamento de DNA

Fragment Assembly: fragmentos de DNA se sobrepõem. Modelar como grafo de De Bruijn (k-mers como vértices, sobreposições como arestas) e encontrar Circuito Euleriano = montar o genoma!

3. Planejamento de Trilhas

4. Traçado de Circuitos (PCB)

18. Aplicações de Euleriano: Exemplos Reais

1. Roteiro de Inspeção de Ruas

Um gari deve varrer todas as ruas de um bairro exatamente uma vez. Modelagem: ruas = arestas, cruzamentos = vértices. Se existe Circuito Euleriano, perfeito. Senão, encontrar o **Caminho Chinês do Carteiro** (duplicar arestas para equilibrar graus).

2. Sequenciamento de DNA

Fragment Assembly: fragmentos de DNA se sobrepõem. Modelar como grafo de De Bruijn (k-mers como vértices, sobreposições como arestas) e encontrar Circuito Euleriano = montar o genoma!

3. Planejamento de Trilhas

Um parque quer que os visitantes percorram todos os caminhos sem repetição. Verificar existência do Circuito Euleriano e, se não existir, encontrar o conjunto mínimo de trechos a repetir.

4. Traçado de Circuitos (PCB)

18. Aplicações de Euleriano: Exemplos Reais

1. Roteiro de Inspeção de Ruas

Um gari deve varrer todas as ruas de um bairro exatamente uma vez. Modelagem: ruas = arestas, cruzamentos = vértices. Se existe Circuito Euleriano, perfeito. Senão, encontrar o **Caminho Chinês do Carteiro** (duplicar arestas para equilibrar graus).

2. Sequenciamento de DNA

Fragment Assembly: fragmentos de DNA se sobrepõem. Modelar como grafo de De Bruijn (k-mers como vértices, sobreposições como arestas) e encontrar Circuito Euleriano = montar o genoma!

3. Planejamento de Trilhas

Um parque quer que os visitantes percorram todos os caminhos sem repetição. Verificar existência do Circuito Euleriano e, se não existir, encontrar o conjunto mínimo de trechos a repetir.

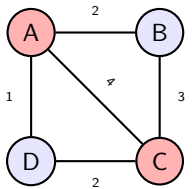
4. Traçado de Circuitos (PCB)

Máquinas de perfuração devem visitar todos os furos de uma placa eletrônica minimizando o deslocamento da broca $\Rightarrow$ variante do TSP!

19. O Problema do Carteiro Chinês

Quando o grafo **não é Euleriano** (tem vértices de grau ímpar), ainda queremos percorrer todas as arestas com o **menor custo total**:

Algoritmo



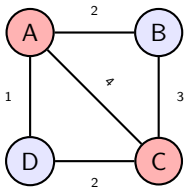
A e C: grau ímpar

19. O Problema do Carteiro Chinês

Quando o grafo **não é Euleriano** (tem vértices de grau ímpar), ainda queremos percorrer todas as arestas com o **menor custo total**:

Algoritmo

1. Encontre todos os vértices de **grau ímpar** (deve haver número par deles).



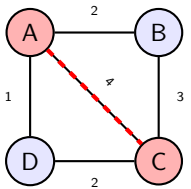
A e C: grau ímpar

19. O Problema do Carteiro Chinês

Quando o grafo **não é Euleriano** (tem vértices de grau ímpar), ainda queremos percorrer todas as arestas com o **menor custo total**:

Algoritmo

1. Encontre todos os vértices de **grau ímpar** (deve haver número par deles).
2. Encontre o **emparelhamento de peso mínimo** entre eles.



A e C: grau ímpar

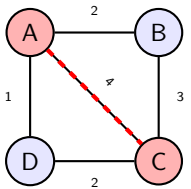
Duplica aresta A-C

19. O Problema do Carteiro Chinês

Quando o grafo **não é Euleriano** (tem vértices de grau ímpar), ainda queremos percorrer todas as arestas com o **menor custo total**:

Algoritmo

1. Encontre todos os vértices de **grau ímpar** (deve haver número par deles).
2. Encontre o **emparelhamento de peso mínimo** entre eles.
3. **Duplique** as arestas nos caminhos mínimos entre cada par emparelhado.



A e C: grau ímpar

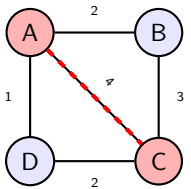
Duplica aresta A-C

19. O Problema do Carteiro Chinês

Quando o grafo **não é Euleriano** (tem vértices de grau ímpar), ainda queremos percorrer todas as arestas com o **menor custo total**:

Algoritmo

1. Encontre todos os vértices de **grau ímpar** (deve haver número par deles).
2. Encontre o **emparelhamento de peso mínimo** entre eles.
3. **Duplique** as arestas nos caminhos mínimos entre cada par emparelhado.
4. Agora todos os vértices têm grau par $\Rightarrow$ aplique Hierholzer!

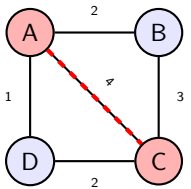


19. O Problema do Carteiro Chinês

Quando o grafo **não é Euleriano** (tem vértices de grau ímpar), ainda queremos percorrer todas as arestas com o **menor custo total**:

Algoritmo

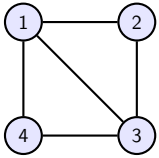
1. Encontre todos os vértices de **grau ímpar** (deve haver número par deles).
2. Encontre o **emparelhamento de peso mínimo** entre eles.
3. **Duplique** as arestas nos caminhos mínimos entre cada par emparelhado.
4. Agora todos os vértices têm grau par $\Rightarrow$ aplique Hierholzer!



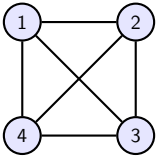
Complexidade

20. Exercício 1: Existe Ciclo Euleriano?

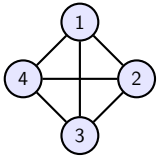
Para cada grafo abaixo, determine se existe **Ciclo Euleriano**, **Caminho Euleriano** ou **nenhum**.



Grafo A



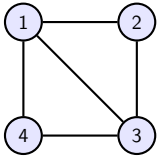
Grafo B



Grafo C

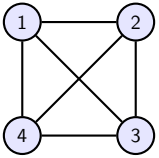
20. Exercício 1: Existe Ciclo Euleriano?

Para cada grafo abaixo, determine se existe **Ciclo Euleriano**, **Caminho Euleriano** ou **nenhum**.



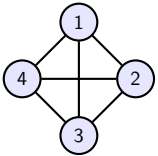
Grafo A

Graus: 3,2,3,2. **Caminho Euleriano (2 ímpares)**



Grafo B

Graus: 3,3,3,3. **Nenhum (4 ímpares)**



Grafo C

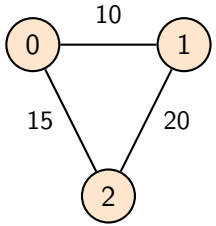
Graus: 4,4,4,4 — pares! **Ciclo Euleriano!**

21. Exercício 2: Trace do TSP com Bitmask ($N = 3$)

Considere 3 cidades com distâncias: $d(0,1) = 10$, $d(0,2) = 15$, $d(1,2) = 20$. Encontre o ciclo ótimo.

Preencha a tabela DP (`dp[mask][u]`):

mask	**u=0**	**u=1**	**u=2**

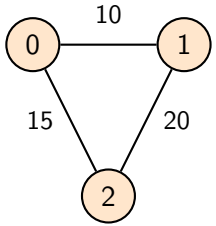


21. Exercício 2: Trace do TSP com Bitmask ($N = 3$)

Considere 3 cidades com distâncias: $d(0,1) = 10$, $d(0,2) = 15$, $d(1,2) = 20$. Encontre o ciclo ótimo.

Preencha a tabela DP (`dp[mask][u]`):

mask	**u=0**	**u=1**	**u=2**
001	0	∞	∞

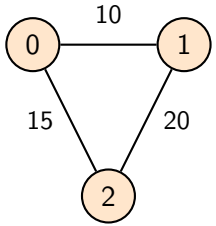


21. Exercício 2: Trace do TSP com Bitmask ($N = 3$)

Considere 3 cidades com distâncias: $d(0,1) = 10$, $d(0,2) = 15$, $d(1,2) = 20$. Encontre o ciclo ótimo.

Preencha a tabela DP (`dp[mask][u]`):

mask	u=0	u=1	u=2
001	0	∞	∞
011	∞	10	∞

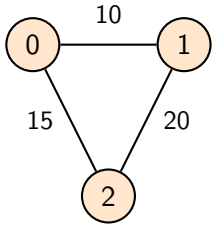


21. Exercício 2: Trace do TSP com Bitmask ($N = 3$)

Considere 3 cidades com distâncias: $d(0,1) = 10$, $d(0,2) = 15$, $d(1,2) = 20$. Encontre o ciclo ótimo.

Preencha a tabela DP (`dp[mask][u]`):

mask	u=0	u=1	u=2
001	0	∞	∞
011	∞	10	∞
101	∞	∞	15

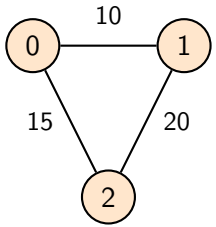


21. Exercício 2: Trace do TSP com Bitmask ($N = 3$)

Considere 3 cidades com distâncias: $d(0,1) = 10$, $d(0,2) = 15$, $d(1,2) = 20$. Encontre o ciclo ótimo.

Preencha a tabela DP (`dp[mask][u]`):

mask	u=0	u=1	u=2
001	0	∞	∞
011	∞	10	∞
101	∞	∞	15
111	∞	35	30

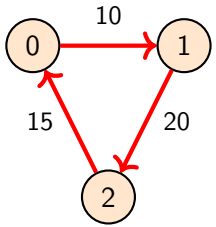


21. Exercício 2: Trace do TSP com Bitmask ($N = 3$)

Considere 3 cidades com distâncias: $d(0,1) = 10$, $d(0,2) = 15$, $d(1,2) = 20$. Encontre o ciclo ótimo.

Preencha a tabela DP (`dp[mask][u]`):

mask	u=0	u=1	u=2
001	0	∞	∞
011	∞	10	∞
101	∞	∞	15
111	∞	35	30



Custo = 45

Resposta

`ans = min(dp[111][1]+d(1,0), dp[111][2]+d(2,0))`

$= \min(35 + 10, 30 + 15) = \min(45, 45) =$ **45**

Ciclo: $0 \to 1 \to 2 \to 0$ ou

22. Exercício 3: Modelagem de Problema

Problema (estilo OBI)

Um robô deve inspecionar todos os **dutos** de uma refinaria. Os dutos formam um grafo não-direcionado. O robô pode começar em qualquer ponto de inspeção. Cada duto deve ser percorrido **exatamente uma vez**.

Dados: $N \leq 10^5$ pontos, $M \leq 2 \times 10^5$ dutos.

22. Exercício 3: Modelagem de Problema

Problema (estilo OBI)

Um robô deve inspecionar todos os **dutos** de uma refinaria. Os dutos formam um grafo não-direcionado. O robô pode começar em qualquer ponto de inspeção. Cada duto deve ser percorrido **exatamente uma vez**.

Dados: $N \leq 10^5$ pontos, $M \leq 2 \times 10^5$ dutos.

1. Como verificar se existe uma rota válida? *(Dica: Teorema de Euler)*

22. Exercício 3: Modelagem de Problema

Problema (estilo OBI)

Um robô deve inspecionar todos os **dutos** de uma refinaria. Os dutos formam um grafo não-direcionado. O robô pode começar em qualquer ponto de inspeção. Cada duto deve ser percorrido **exatamente uma vez**.

Dados: $N \leq 10^5$ pontos, $M \leq 2 \times 10^5$ dutos.

1. Como verificar se existe uma rota válida? *(Dica: Teorema de Euler)*
2. Se existir, como **encontrar** a rota em tempo eficiente?

22. Exercício 3: Modelagem de Problema

Problema (estilo OBI)

Um robô deve inspecionar todos os **dutos** de uma refinaria. Os dutos formam um grafo não-direcionado. O robô pode começar em qualquer ponto de inspeção. Cada duto deve ser percorrido **exatamente uma vez**.

Dados: $N \leq 10^5$ pontos, $M \leq 2 \times 10^5$ dutos.

1. Como verificar se existe uma rota válida? *(Dica: Teorema de Euler)*
2. Se existir, como **encontrar** a rota em tempo eficiente?
3. Se não existir (graus ímpares), qual a estratégia?

22. Exercício 3: Modelagem de Problema

Problema (estilo OBI)

Um robô deve inspecionar todos os **dutos** de uma refinaria. Os dutos formam um grafo não-direcionado. O robô pode começar em qualquer ponto de inspeção. Cada duto deve ser percorrido **exatamente uma vez**.

Dados: $N \leq 10^5$ pontos, $M \leq 2 \times 10^5$ dutos.

1. Como verificar se existe uma rota válida? *(Dica: Teorema de Euler)*
2. Se existir, como **encontrar** a rota em tempo eficiente?
3. Se não existir (graus ímpares), qual a estratégia?
4. Qual seria a complexidade total da sua solução?

22. Exercício 3: Modelagem de Problema

Problema (estilo OBI)

Um robô deve inspecionar todos os **dutos** de uma refinaria. Os dutos formam um grafo não-direcionado. O robô pode começar em qualquer ponto de inspeção. Cada duto deve ser percorrido **exatamente uma vez**.

Dados: $N \leq 10^5$ pontos, $M \leq 2 \times 10^5$ dutos.

1. Como verificar se existe uma rota válida? *(Dica: Teorema de Euler)*
2. Se existir, como **encontrar** a rota em tempo eficiente?
3. Se não existir (graus ímpares), qual a estratégia?
4. Qual seria a complexidade total da sua solução?

Gabarito

1. Verificar se todos os vértices têm grau par (Ciclo Euleriano) ou exatamente 2 têm grau ímpar (Caminho Euleriano).

2. Algoritmo de Hierholzer em $O(E)$.

3. Carteiro Chinês: emparelhamento mínimo nos vértices ímpares $O(k^3)$ onde k é o número

23. Exercício 4: Modelagem Hamiltoniana

Problema

Uma empresa de logística tem $N \leq 15$ depósitos e precisa fazer um circuito de inspeção visitando **cada depósito exatamente uma vez**. As distâncias entre depósitos são dadas numa matriz $N \times N$. Qual o menor percurso total?

23. Exercício 4: Modelagem Hamiltoniana

Problema

Uma empresa de logística tem $N \leq 15$ depósitos e precisa fazer um circuito de inspeção visitando **cada depósito exatamente uma vez**. As distâncias entre depósitos são dadas numa matriz $N \times N$. Qual o menor percurso total?

1. Qual abordagem algorítmica usar? Por quê?

23. Exercício 4: Modelagem Hamiltoniana

Problema

Uma empresa de logística tem $N \leq 15$ depósitos e precisa fazer um circuito de inspeção visitando **cada depósito exatamente uma vez**. As distâncias entre depósitos são dadas numa matriz $N \times N$. Qual o menor percurso total?

1. Qual abordagem algorítmica usar? Por quê?
2. Qual é a complexidade de tempo e espaço?

23. Exercício 4: Modelagem Hamiltoniana

Problema

Uma empresa de logística tem $N \leq 15$ depósitos e precisa fazer um circuito de inspeção visitando **cada depósito exatamente uma vez**. As distâncias entre depósitos são dadas numa matriz $N \times N$. Qual o menor percurso total?

1. Qual abordagem algorítmica usar? Por quê?
2. Qual é a complexidade de tempo e espaço?
3. Se $N = 15$, quantas operações estimamos?

23. Exercício 4: Modelagem Hamiltoniana

Problema

Uma empresa de logística tem $N \leq 15$ depósitos e precisa fazer um circuito de inspeção visitando **cada depósito exatamente uma vez**. As distâncias entre depósitos são dadas numa matriz $N \times N$. Qual o menor percurso total?

1. Qual abordagem algorítmica usar? Por quê?
2. Qual é a complexidade de tempo e espaço?
3. Se $N = 15$, quantas operações estimamos?

Gabarito

1. TSP com DP Bitmask: $dp[\texttt{mask}][u]$ = menor custo visitando `mask` e terminando em u.
2. Tempo: $O(2^N \cdot N^2)$. Espaço: $O(2^N \cdot N)$.
3. $N = 15$: $2^{15} \times 15^2 \approx 7.4 \times 10^6$ operações. Confortável!
Espaço: $2^{15} \times 15 \times 4$ bytes ≈ 2 MB.

24. Resumo: O Grande Quadro Comparativo

Problema	Condição	Algoritmo	Complexidade

24. Resumo: O Grande Quadro Comparativo

Problema	**Condição**	**Algoritmo**	**Complexidade**
Ciclo Euleriano	Todos graus pares	Hierholzer	$O(E)$

24. Resumo: O Grande Quadro Comparativo

Problema	**Condição**	**Algoritmo**	**Complexidade**
Ciclo Euleriano	Todos graus pares	Hierholzer	$O(E)$
Caminho Euleriano	0 ou 2 graus ímpares	Hierholzer mod.	$O(E)$

24. Resumo: O Grande Quadro Comparativo

Problema	**Condição**	**Algoritmo**	**Complexidade**
Ciclo Euleriano	Todos graus pares	Hierholzer	$O(E)$
Caminho Euleriano	0 ou 2 graus ímpares	Hierholzer mod.	$O(E)$
Carteiro Chinês	Qualquer	MST + Matching	$O(V^3)$

24. Resumo: O Grande Quadro Comparativo

Problema	**Condição**	**Algoritmo**	**Complexidade**
Ciclo Euleriano	Todos graus pares	Hierholzer	$O(E)$
Caminho Euleriano	0 ou 2 graus ímpares	Hierholzer mod.	$O(E)$
Carteiro Chinês	Qualquer	MST + Matching	$O(V^3)$
Ciclo Hamiltoniano	NP-Completo	–	Exponencial

24. Resumo: O Grande Quadro Comparativo

Problema	**Condição**	**Algoritmo**	**Complexidade**
Ciclo Euleriano	Todos graus pares	Hierholzer	$O(E)$
Caminho Euleriano	0 ou 2 graus ímpares	Hierholzer mod.	$O(E)$
Carteiro Chinês	Qualquer	MST + Matching	$O(V^3)$
Ciclo Hamiltoniano	NP-Completo	–	Exponencial
TSP Exato	$N \leq 20$	DP Bitmask	$O(2^N \cdot N^2)$

24. Resumo: O Grande Quadro Comparativo

Problema	**Condição**	**Algoritmo**	**Complexidade**
Ciclo Euleriano	Todos graus pares	Hierholzer	$O(E)$
Caminho Euleriano	0 ou 2 graus ímpares	Hierholzer mod.	$O(E)$
Carteiro Chinês	Qualquer	MST + Matching	$O(V^3)$
Ciclo Hamiltoniano	NP-Completo	–	Exponencial
TSP Exato	$N \leq 20$	DP Bitmask	$O(2^N \cdot N^2)$
TSP Aproximado	N grande	2-OPT, Christofides	$O(N^2)/O(N^3)$

24. Resumo: O Grande Quadro Comparativo

Problema	**Condição**	**Algoritmo**	**Complexidade**
Ciclo Euleriano	Todos graus pares	Hierholzer	$O(E)$
Caminho Euleriano	0 ou 2 graus ímpares	Hierholzer mod.	$O(E)$
Carteiro Chinês	Qualquer	MST + Matching	$O(V^3)$
Ciclo Hamiltoniano	NP-Completo	–	Exponencial
TSP Exato	$N \leq 20$	DP Bitmask	$O(2^N \cdot N^2)$
TSP Aproximado	N grande	2-OPT, Christofides	$O(N^2)/O(N^3)$

Mensagem Final

Euleriano é sobre **arestas** $\Rightarrow$ critério simples, algoritmo eficiente (P).
Hamiltoniano é sobre **vértices** $\Rightarrow$ sem critério geral, NP-Completo.
Esta distinção é um dos exemplos mais elegantes da Teoria da Complexidade!

25. Checklist de Maratona

Quando usar o quê?

25. Checklist de Maratona

Quando usar o quê?

1. O problema pede percorrer **cada aresta** uma vez? $\Rightarrow$ **Euleriano**.
   - Verifica graus ($O(E)$) e aplica Hierholzer.

25. Checklist de Maratona

Quando usar o quê?

1. O problema pede percorrer **cada aresta** uma vez? $\Rightarrow$ **Euleriano**.
   - Verifica graus ($O(E)$) e aplica Hierholzer.
2. Tem vértices de grau ímpar? $\Rightarrow$ **Carteiro Chinês**.
   - Emparelhamento mínimo nos vértices ímpares.

25. Checklist de Maratona

Quando usar o quê?

1. O problema pede percorrer **cada aresta** uma vez? $\Rightarrow$ **Euleriano**.
   - Verifica graus ($O(E)$) e aplica Hierholzer.
2. Tem vértices de grau ímpar? $\Rightarrow$ **Carteiro Chinês**.
   - Emparelhamento mínimo nos vértices ímpares.
3. O problema pede percorrer **cada vértice** uma vez? $\Rightarrow$ **Hamiltoniano/TSP**.
   - $N \leq 20$: DP Bitmask $O(2^N N^2)$.
   - N grande: heurísticas.

25. Checklist de Maratona

Quando usar o quê?

1. O problema pede percorrer **cada aresta** uma vez? $\Rightarrow$ **Euleriano**.
   - Verifica graus ($O(E)$) e aplica Hierholzer.
2. Tem vértices de grau ímpar? $\Rightarrow$ **Carteiro Chinês**.
   - Emparelhamento mínimo nos vértices ímpares.
3. O problema pede percorrer **cada vértice** uma vez? $\Rightarrow$ **Hamiltoniano/TSP**.
   - $N \leq 20$: DP Bitmask $O(2^N N^2)$.
   - N grande: heurísticas.
4. Precisa apenas saber **se existe** ciclo Hamiltoniano? $\Rightarrow$ NP-Completo, use heurísticas ou instâncias pequenas.

25. Checklist de Maratona

Quando usar o quê?

1. O problema pede percorrer **cada aresta** uma vez? $\Rightarrow$ **Euleriano**.
   - Verifica graus ($O(E)$) e aplica Hierholzer.
2. Tem vértices de grau ímpar? $\Rightarrow$ **Carteiro Chinês**.
   - Emparelhamento mínimo nos vértices ímpares.
3. O problema pede percorrer **cada vértice** uma vez? $\Rightarrow$ **Hamiltoniano/TSP**.
   - $N \leq 20$: DP Bitmask $O(2^N N^2)$.
   - N grande: heurísticas.
4. Precisa apenas saber **se existe** ciclo Hamiltoniano? $\Rightarrow$ NP-Completo, use heurísticas ou instâncias pequenas.

Dica Extra

Grafos de grau alto (Dirac: grau $\geq N/2$) tendem a ter Ciclo Hamiltoniano. Para competições, este é um sinal de que a instância foi construída para ter solução.