

ESTRUTURAS DE DADOS APLICADAS A SISTEMAS REAIS

POR QUE ESTUDAR ESTRUTURAS DE DADOS?

As estruturas de dados são fundamentais para resolver problemas comuns em sistemas:

- Armazenamento eficiente
- Organização lógica
- Performance nas buscas e operações



ONDE ELAS APARECEM NO MUNDO REAL?



Redes sociais
Apps de banco
Serviços de entrega e GPS
Jogos e sistemas operacionais

OBJETIVO DA APRESENTAÇÃO

Mostrar como as estruturas de dados são aplicadas na prática para melhorar a eficiência, organização e desempenho de sistemas reais e vamos ver as principais estruturas de dados e onde elas aparecem nos sistemas reais que usamos todos os dias.

VETORES / LISTAS

📌 O que são?

Estruturas lineares onde os elementos são armazenados em sequência e acessados por índice.



Usos reais:

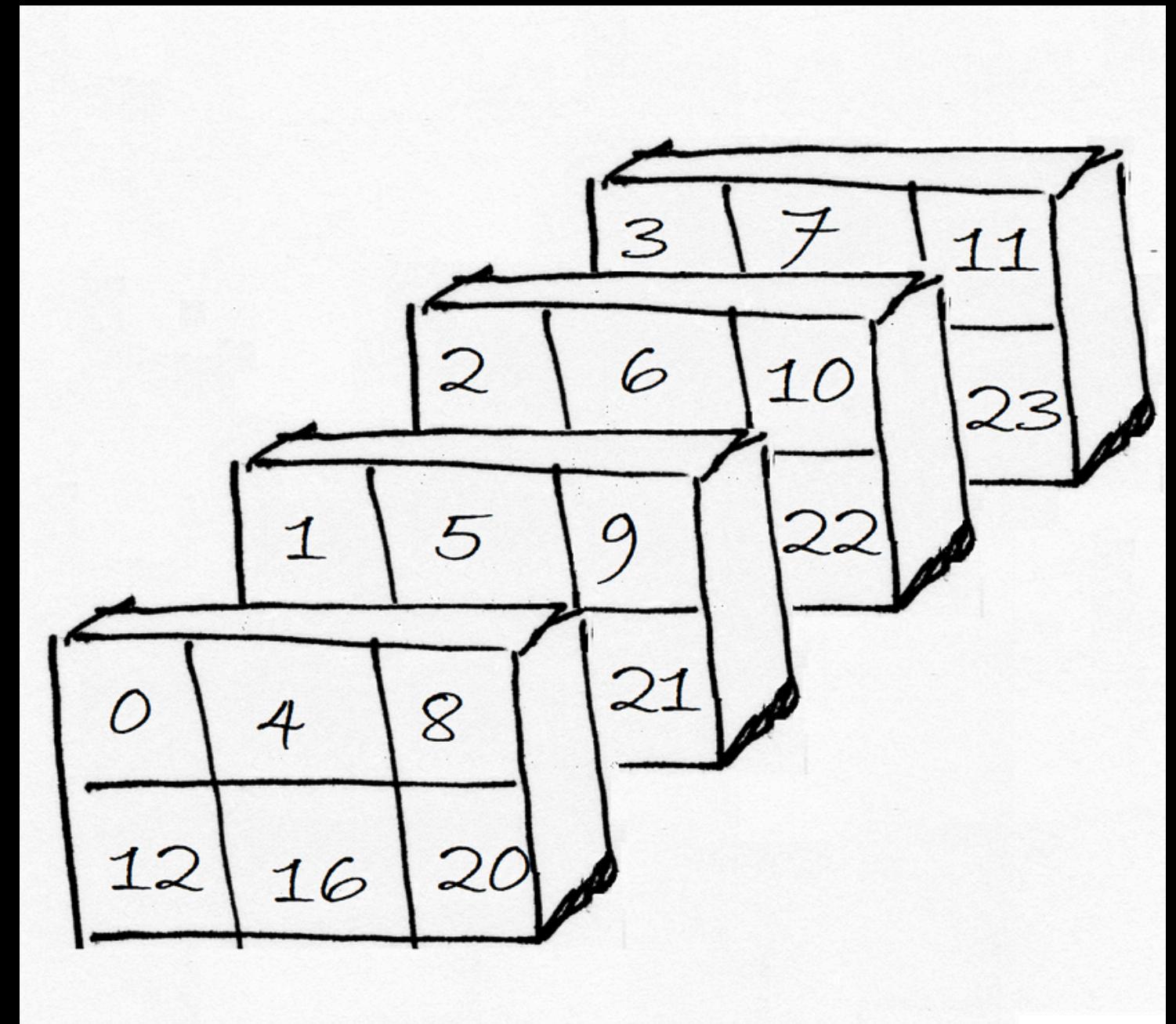
- Lista de produtos em e-commerce
- Lista de contatos em um app de mensagens
- Carrinho de compras

Vetores

Vetor é uma estrutura linear com elementos do mesmo tipo armazenados em posições contíguas de memória.

Permite acesso direto a qualquer elemento pelo índice ($O(1)$).

Também conhecido como array.



Características principais dos Vetores

- Tamanho fixo definido na criação.
- Acesso rápido ($O(1)$) pelo índice.
- Inserção/removal no meio é custosa ($O(n)$).
- Armazenamento contíguo na memória.

Operações básicas em Vetores

- Acesso direto: leitura e escrita em $O(1)$.
- Inserção/remoção no final em $O(1)$ se houver espaço.
- Inserção/remoção no meio requer deslocamento ($O(n)$).
- Percorrer elementos: $O(n)$.

Vantagens e Desvantagens dos Vetores

Vantagens:

- acesso rápido
- fácil implementação.

Desvantagens:

- tamanho fixo
- + custo em inserções e remoções.

Exemplo prático e aplicação de Vetores

- Usado para armazenar dados estáticos, buffers, tabelas.

Exemplo simples em Python:

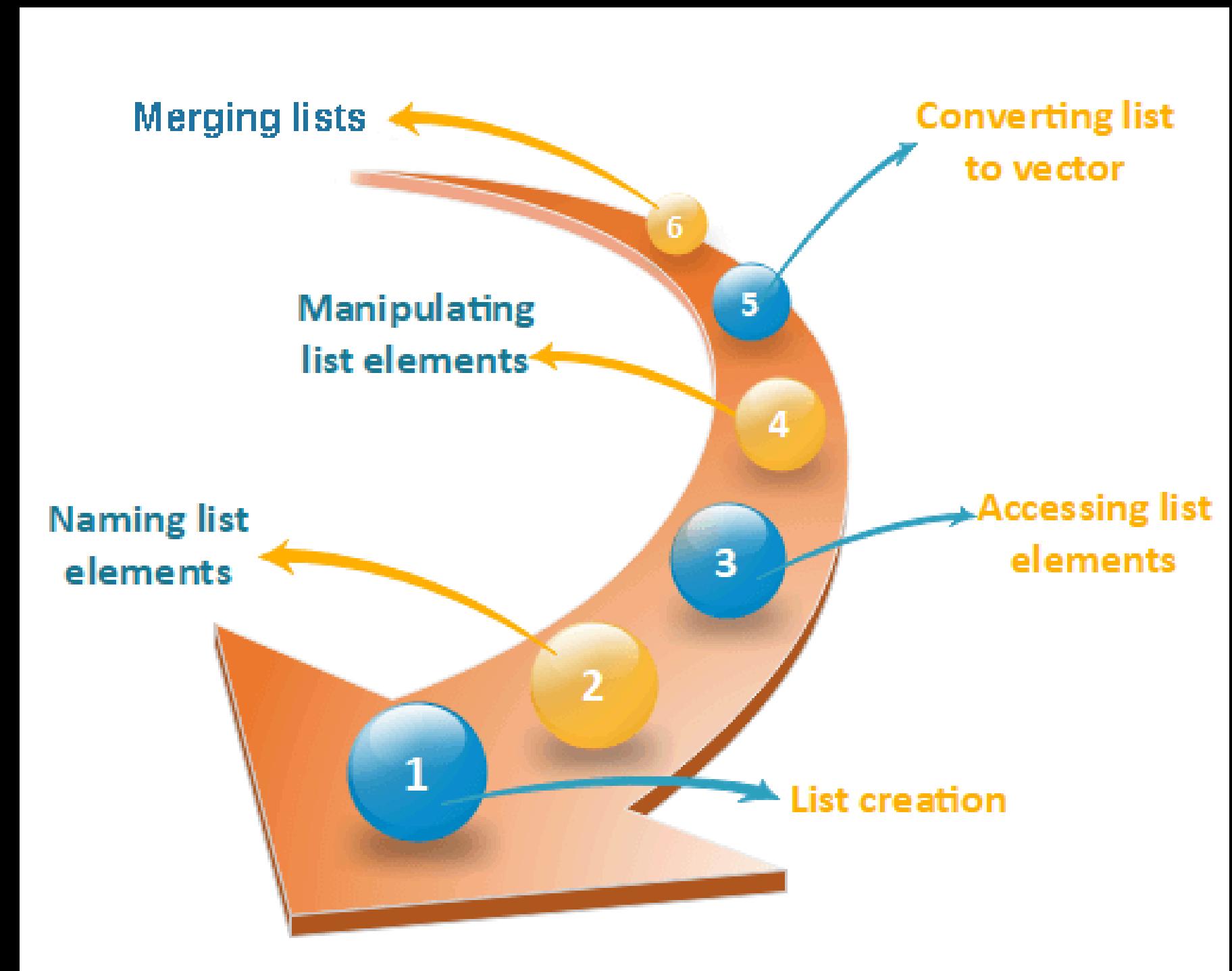
```
array = [10, 20, 30]  
print(array[1]) # 20
```

Listas

Lista é uma estrutura linear de nós, cada nó aponta para o próximo.

Pode ser simples ou duplamente encadeada.

Tamanho dinâmico, pode crescer e diminuir.



Características principais das Listas

- Elementos não contíguos em memória.
- Inserção/removal eficientes ($O(1)$)
com referência ao nó.
- Acesso sequencial lento ($O(n)$).

Operações básicas em Listas

- Inserção/removal ajustando ponteiros.
- Percorrer para busca/extracão.
- Listas duplas permitem navegação para frente e para trás.

Vantagens e Desvantagens das Listas:

Vantagens:

- tamanho dinâmico
- inserção/removal rápidas.

Desvantagens:

- acesso lento
- memória usada por ponteiros.

Exemplo prático e aplicações de Listas:

Uso em sistemas operacionais, pilhas, filas.

Exemplo em Python:

```
class Node:  
    def __init__(self, val):  
        self.val = val  
        self.next = None
```

PILHAS (STACKS)

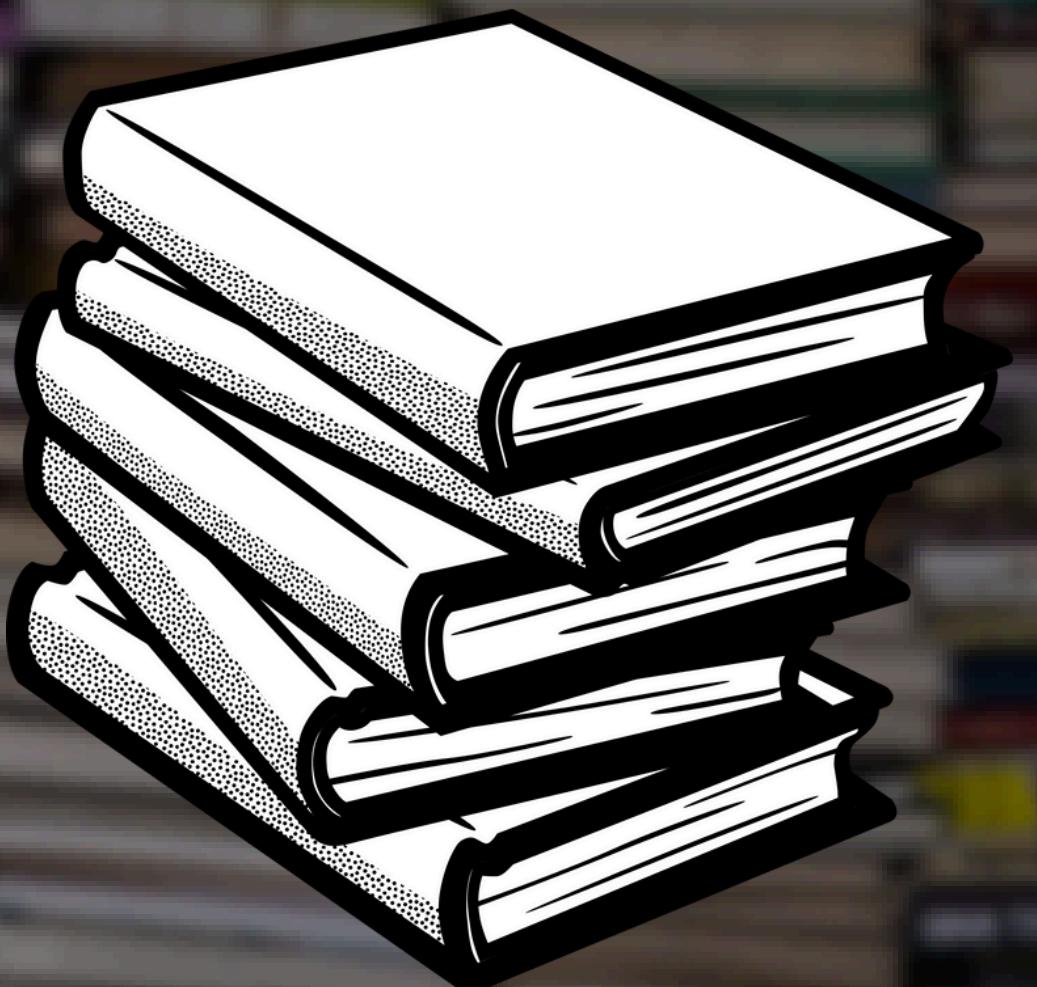
📌 O que são?

Estruturas LIFO (Último a Entrar, Primeiro a Sair)



Usos reais:

- Desfazer ações (Ctrl+Z)
- Navegação de páginas (histórico do navegador)
- Chamadas de função (recursão)



NOME

FILAS (QUEUES)

📌 O que são?

Estrutura de dados do tipo FIFO (First In, First Out), onde o primeiro elemento inserido é o primeiro a ser removido.

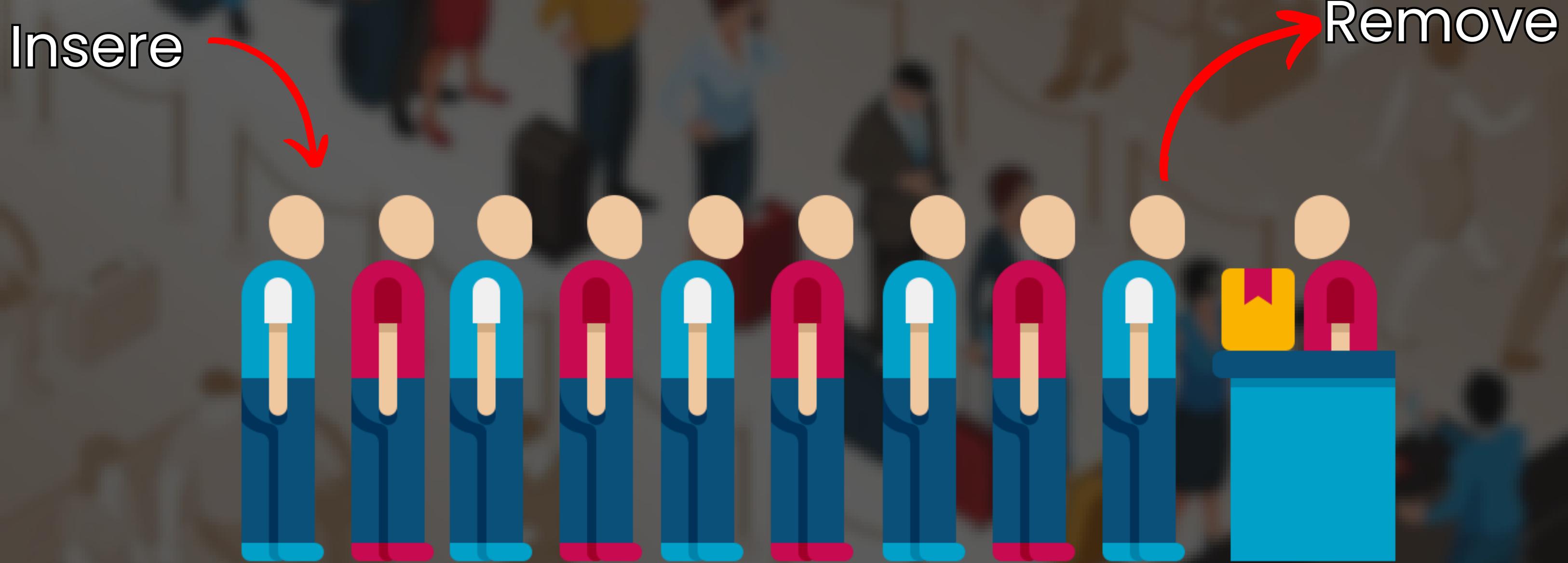
Muito usada em sistemas de atendimento, filas de impressão e processamento de tarefas.

Principais operações:

- enqueue (inserir no final)
- dequeue (remover do início)



 **FIFO (First In, First Out): o primeiro elemento a entrar é o primeiro a sair.**





EXEMPLOS PRÁTICOS DE USO:

🖨️ Impressoras

As tarefas de impressão são colocadas em uma fila: o primeiro documento enviado é o primeiro a ser impresso. Isso evita conflitos e garante a ordem correta de execução.

⚙️ Processamento de tarefas em background

Sistemas operacionais e servidores usam filas para organizar tarefas em segundo plano, como downloads ou requisições. Isso permite processar uma de cada vez, na ordem em que chegaram.

FILAS DE PRIORIDADE / HEAPS



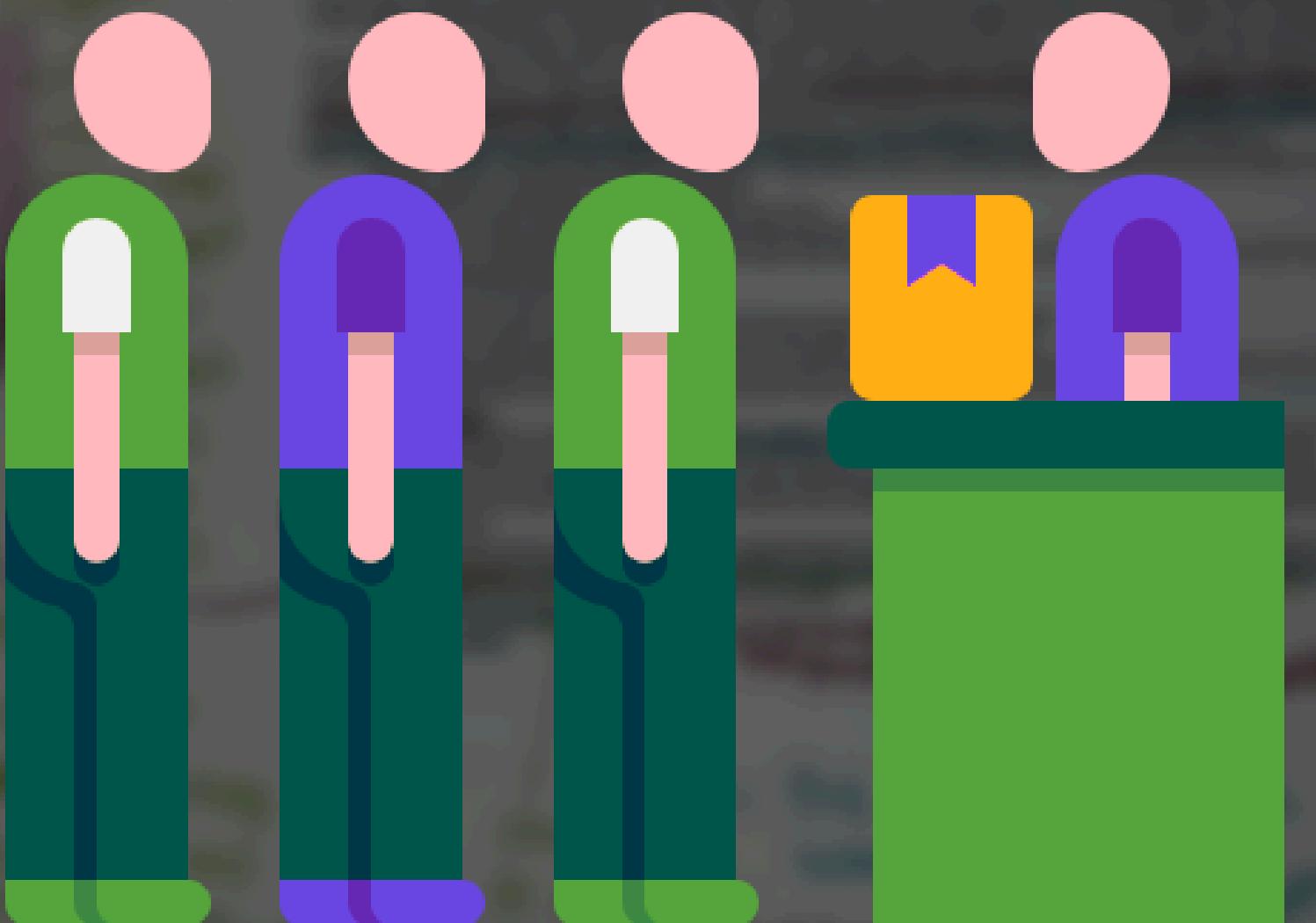
O que são?

Filas em que cada item tem uma prioridade, e o de maior prioridade sai primeiro.

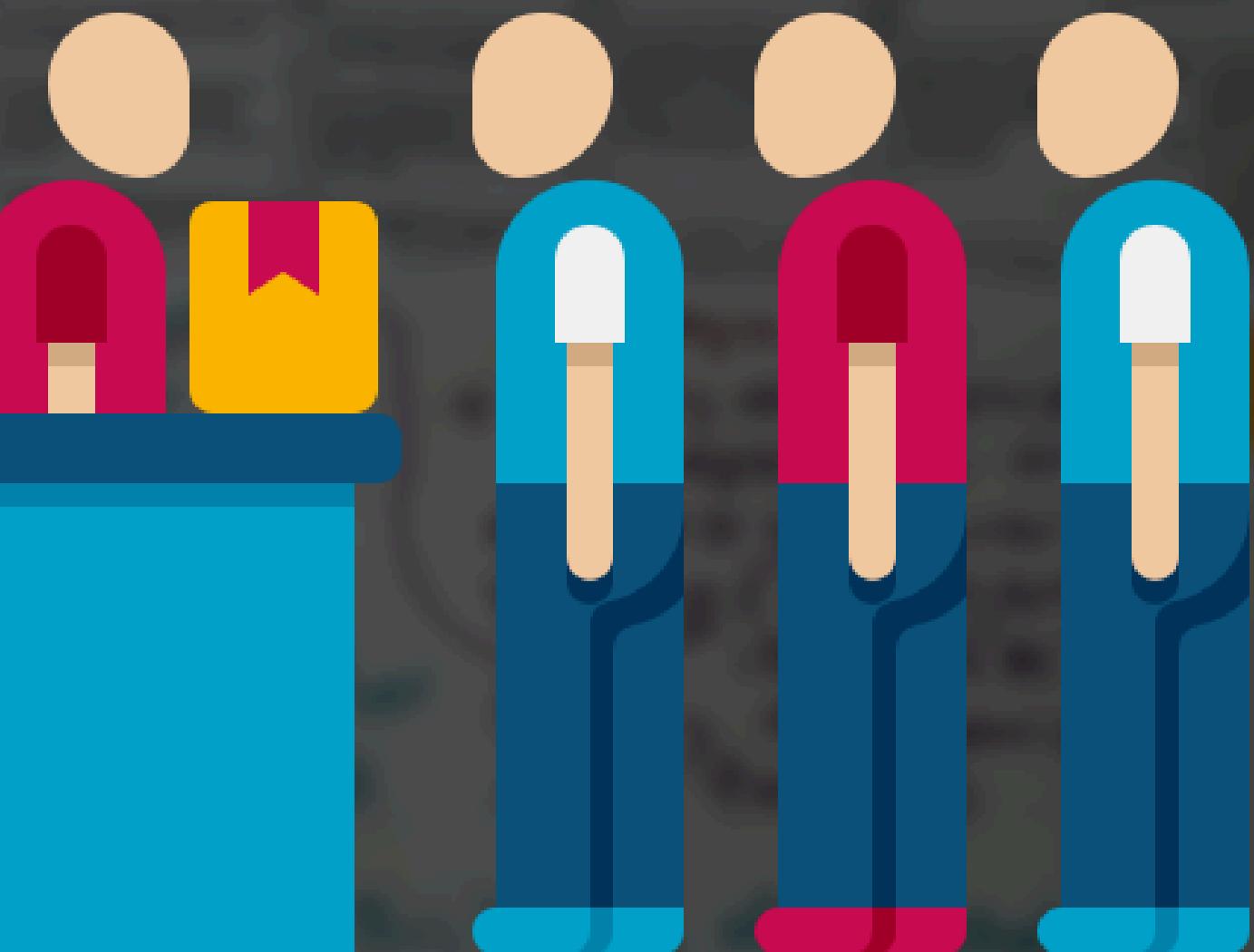
Ao contrário das filas comuns (FIFO), nas filas de prioridade os elementos são removidos com base em sua prioridade – não na ordem de entrada.

Ao contrário das filas comuns (FIFO), nas filas de prioridade os elementos são removidos com base em sua prioridade – não na ordem de entrada.

[Maria] → [Ana] → [Lucas]



[Maria (1)] → [Ana (3)] → [Lucas (2)]





EXEMPLOS PRÁTICOS DE USO:

Emergências médicas

Em um hospital, pacientes em estado grave são atendidos antes, mesmo que tenham chegado depois de outros pacientes com casos leves.

Controle de tráfego aéreo

Aeronaves com baixo combustível ou emergências têm prioridade para pouso, mesmo que outras estejam aguardando há mais tempo.

LISTAS LIGADAS



O que são?

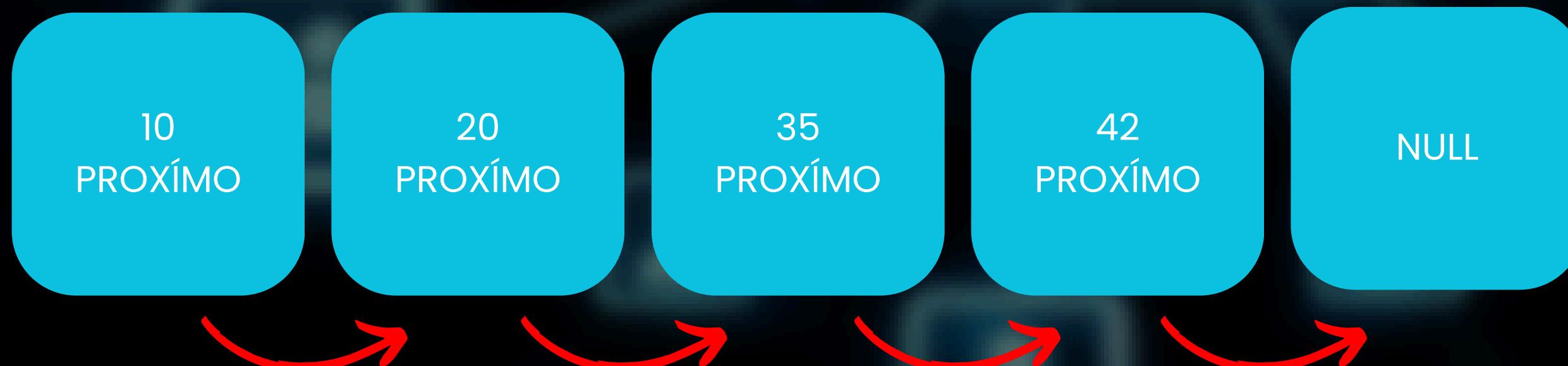
Estruturas compostas por nós conectados entre si, com ponteiros.



CADA BLOCO É UM NÓ, CONTENDO UM VALOR E UM PONTEIRO PARA O PRÓXIMO



EXEMPLO COM DADOS REAIS:



- ● "BLOCO = NÓ"
- ↗ "SETA VERMELHA = LIGAÇÃO COM O PRÓXIMO"
- ✘ "NULL = FIM DA LISTA"



EXEMPLOS PRÁTICOS DE USO:

HISTÓRICO DE NAVEGAÇÃO (BROWSER)

Cada página visitada é um nó que aponta para a próxima (ou anterior), formando uma lista de páginas visitadas.

EDITOR DE TEXTO

Cada linha ou parágrafo pode ser um nó, permitindo que o editor adicione ou remova trechos dinamicamente, sem reorganizar tudo.

ÁRVORES (TREES)



O que são?

Estruturas hierárquicas com um nó raiz e filhos.



Usos reais:

- Organização de arquivos em sistemas operacionais
- Índices em bancos de dados (como B-Trees)
- Árvores de decisão em IA



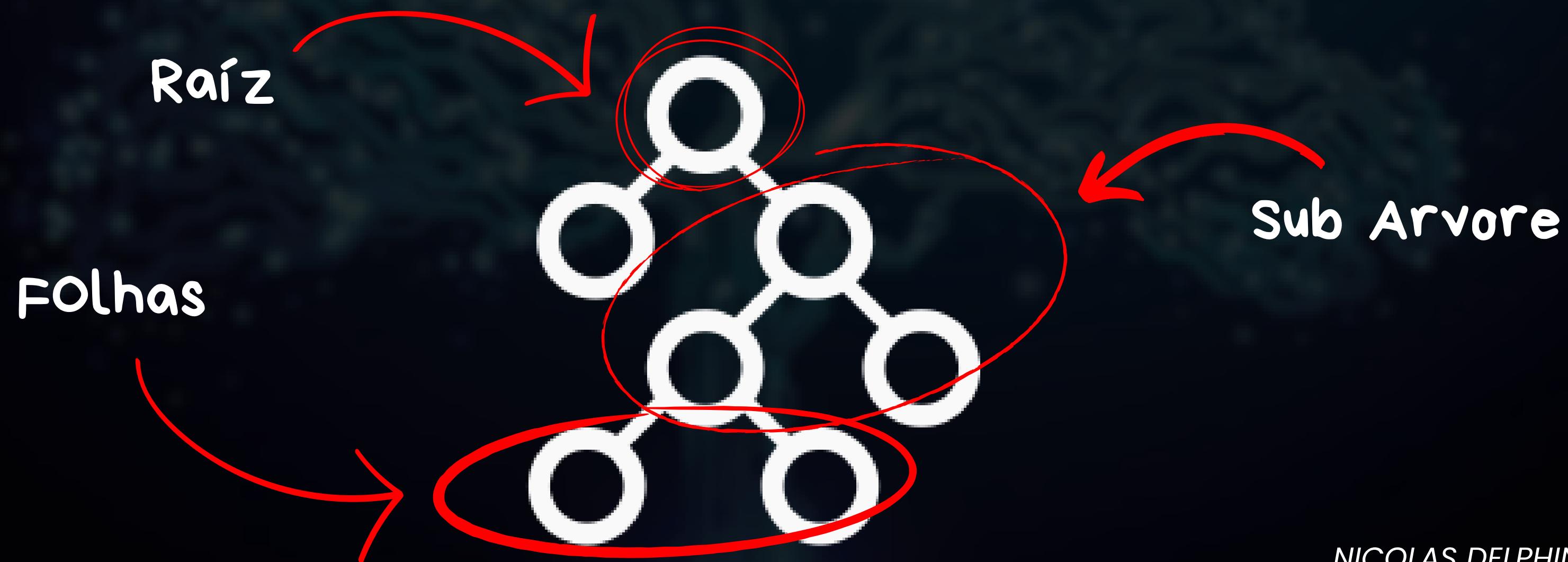
ÁRVORES (TREES)

- Composta de: Raiz, nós e folhas
- Cada nó pode ter no máximo dois filhos (esquerda e direita)
- Valores Menores que a raiz inseridos à esquerda, valores maiores que a raiz inseridos à direita



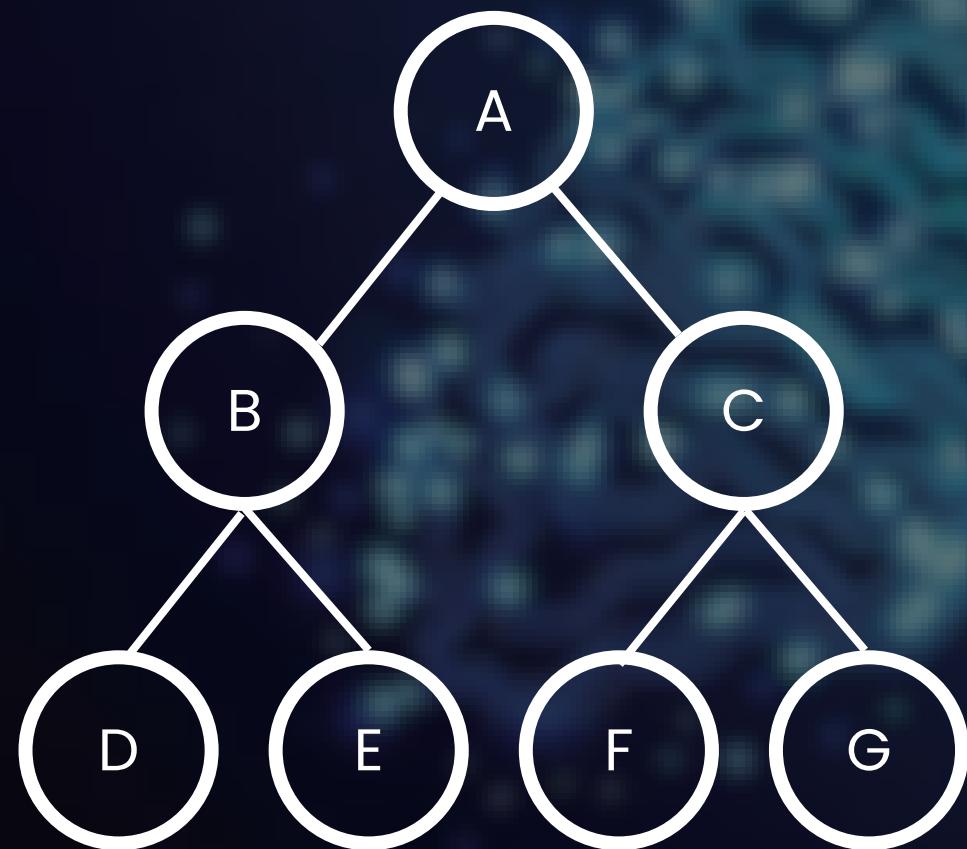
ÁRVORES (TREES)

Cada nó pode ser o pai de outros dois, chamados de filho esquerdo e direito. Nós que não têm filhos são folhas. Um nó com filhos forma uma subárvore.

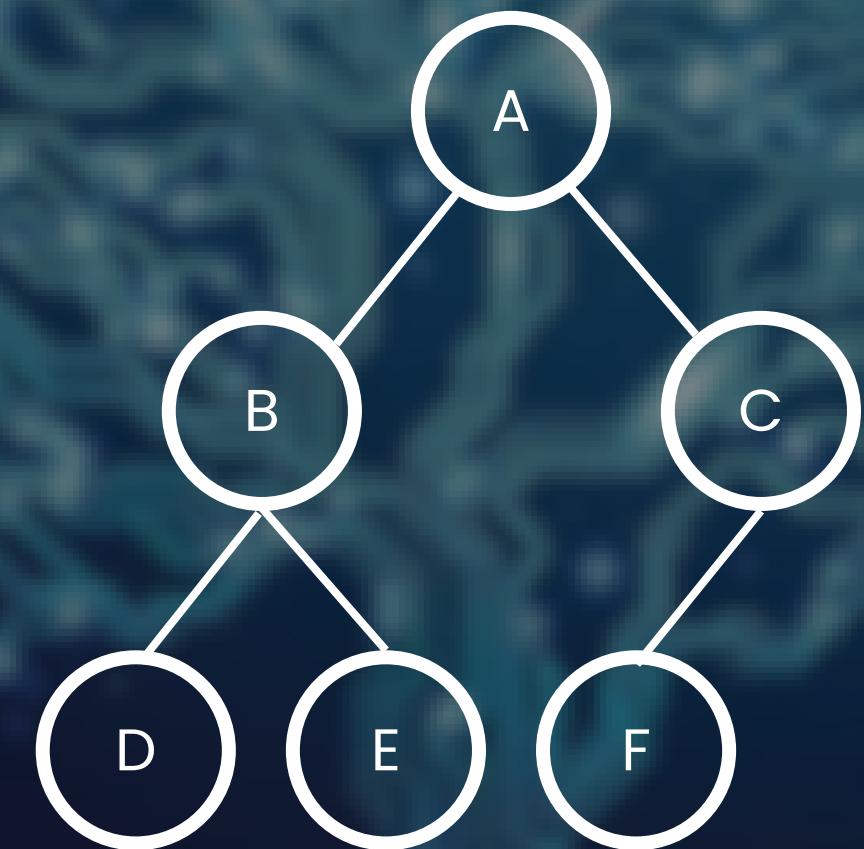


ÁRVORES (TREES)

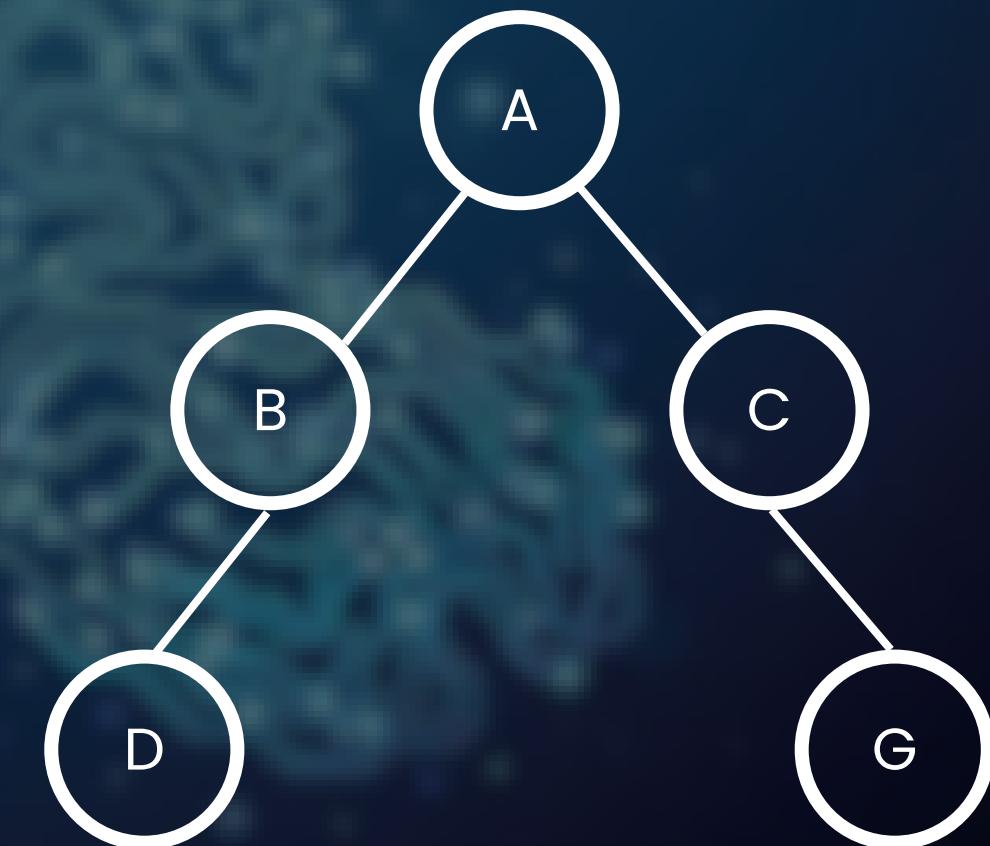
ÁRVORE CHEIA



ÁRVORE COMPLETA



ÁRVORE BALANCEADA



ÁRVORES (TREES)

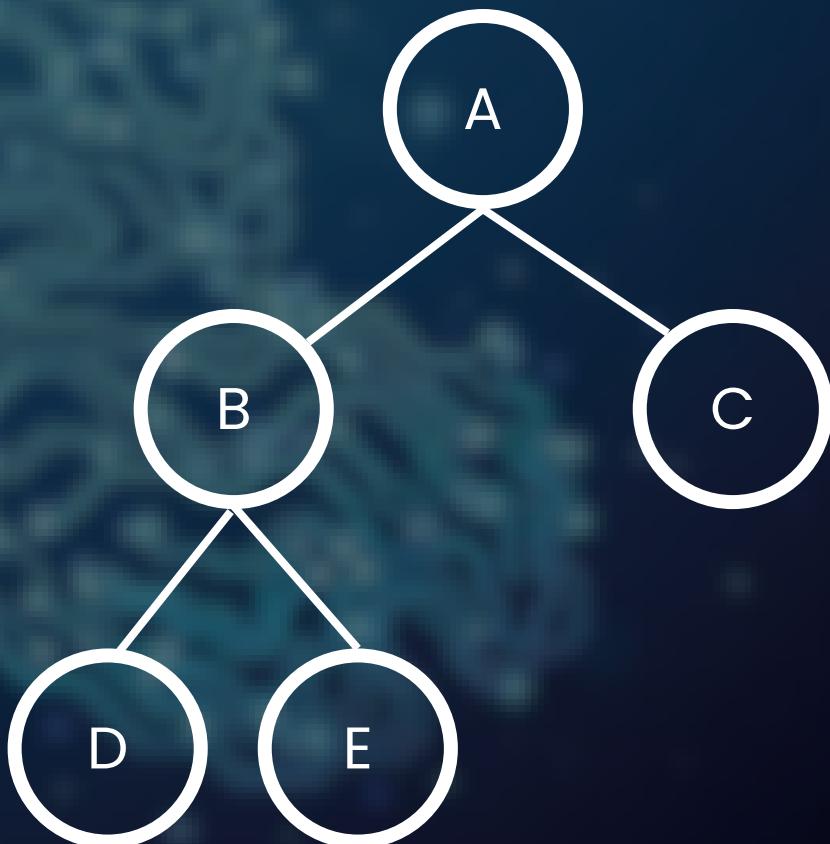


ÁRVORES (TREES)



Banco de Dados – Organização e Busca

Usamos árvores como *BST*, *AVL* ou árvores *B* para manter os dados ordenados e permitir busca rápida.

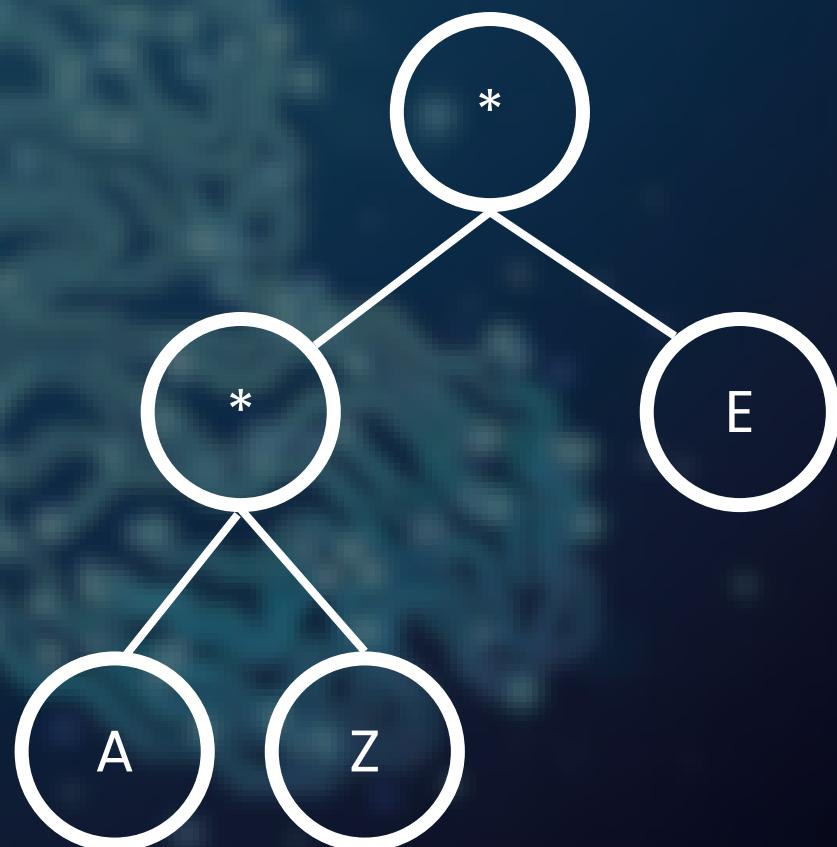


ÁRVORES (TREES)



Compressão – Árvore de Huffman

Em compressão de arquivos, a árvore de Huffman codifica caracteres com menos bits para os mais comuns.



ÁRVORES (TREES)



IA e Jogos – Árvores de Decisão

Cada nó representa uma situação do jogo, e os filhos representam ações possíveis. A IA explora a árvore para simular decisões.



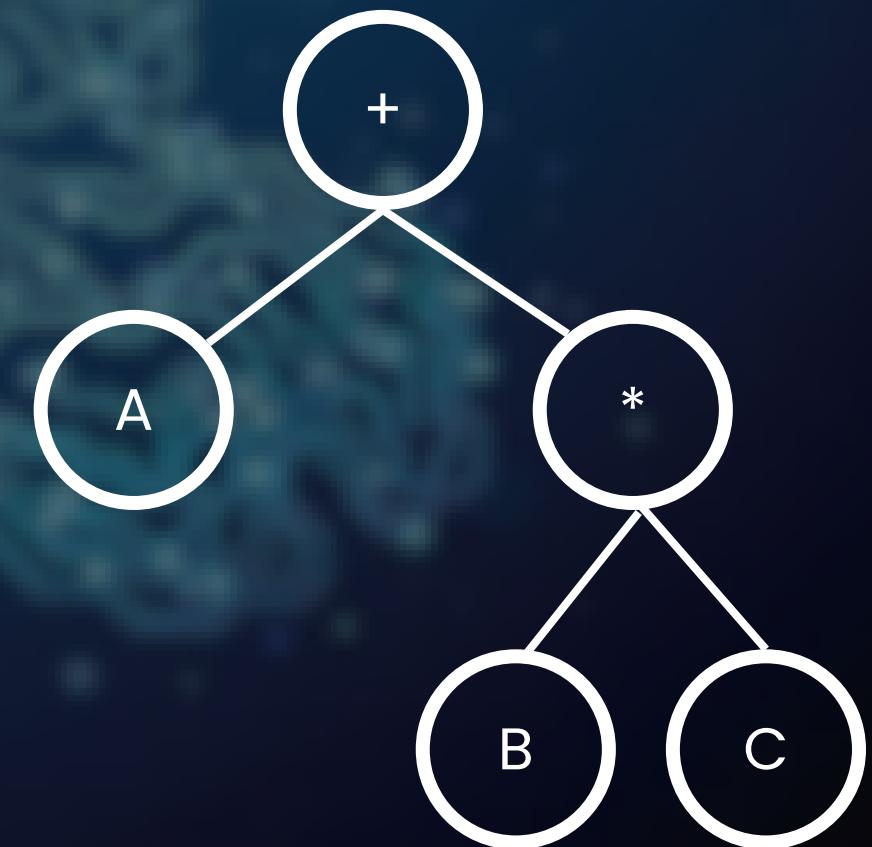
ÁRVORES (TREES)



Compiladores – Árvore de Sintaxe (AST)

Representa a estrutura do código-fonte, facilitando a análise e a tradução para linguagem de máquina.

A+B*C



GRAFOS (GRAPHS)



O que são?

Conjunto de nós (vértices) conectados por arestas (ligações) (relações)

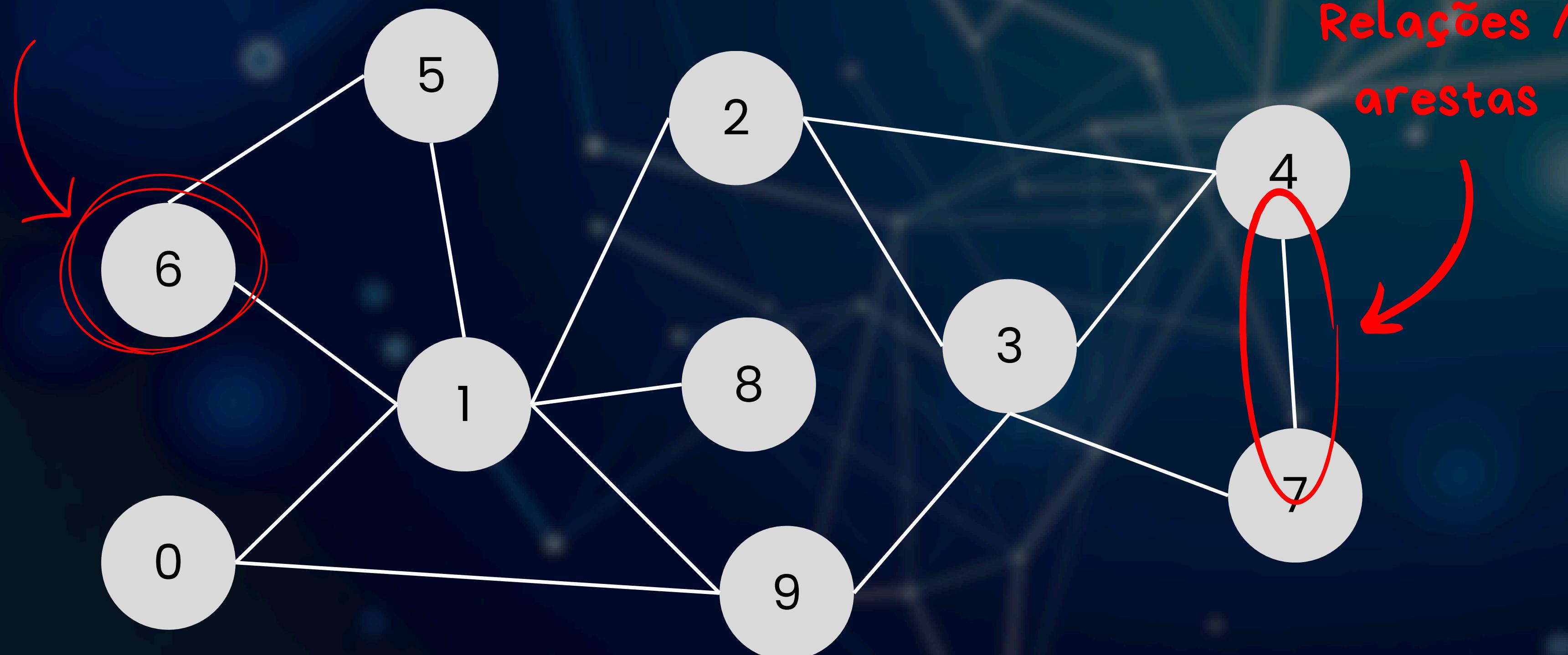


Usos reais:

- Redes sociais (conexões de amigos)
- Sistemas de navegação (GPS, trânsito)
- Recomendação de conteúdo (YouTube, Netflix)
- Sistema de ranqueamento de páginas (Google)

GRAFOS (GRAPHS)

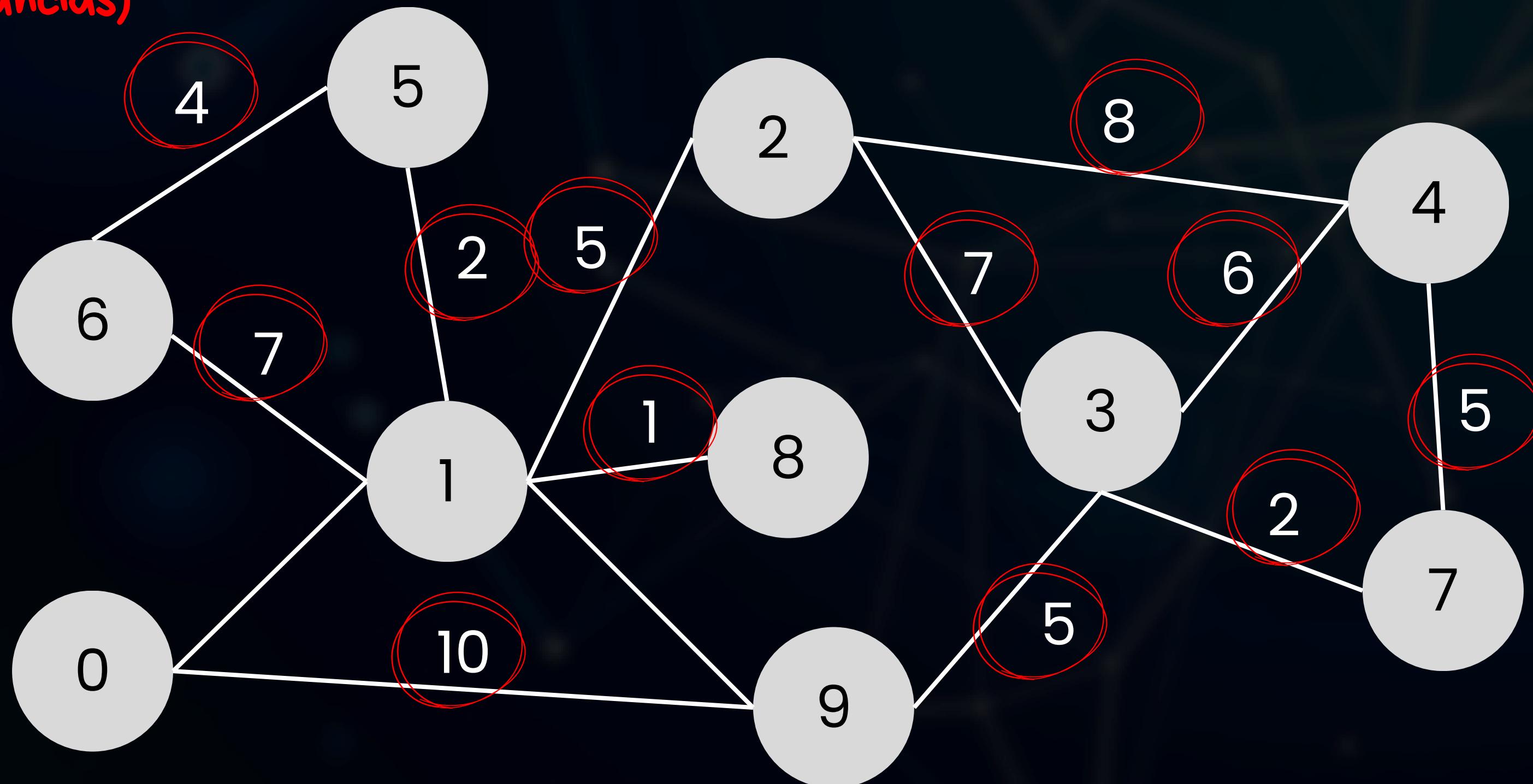
Nó / objeto/
vertice



NOME

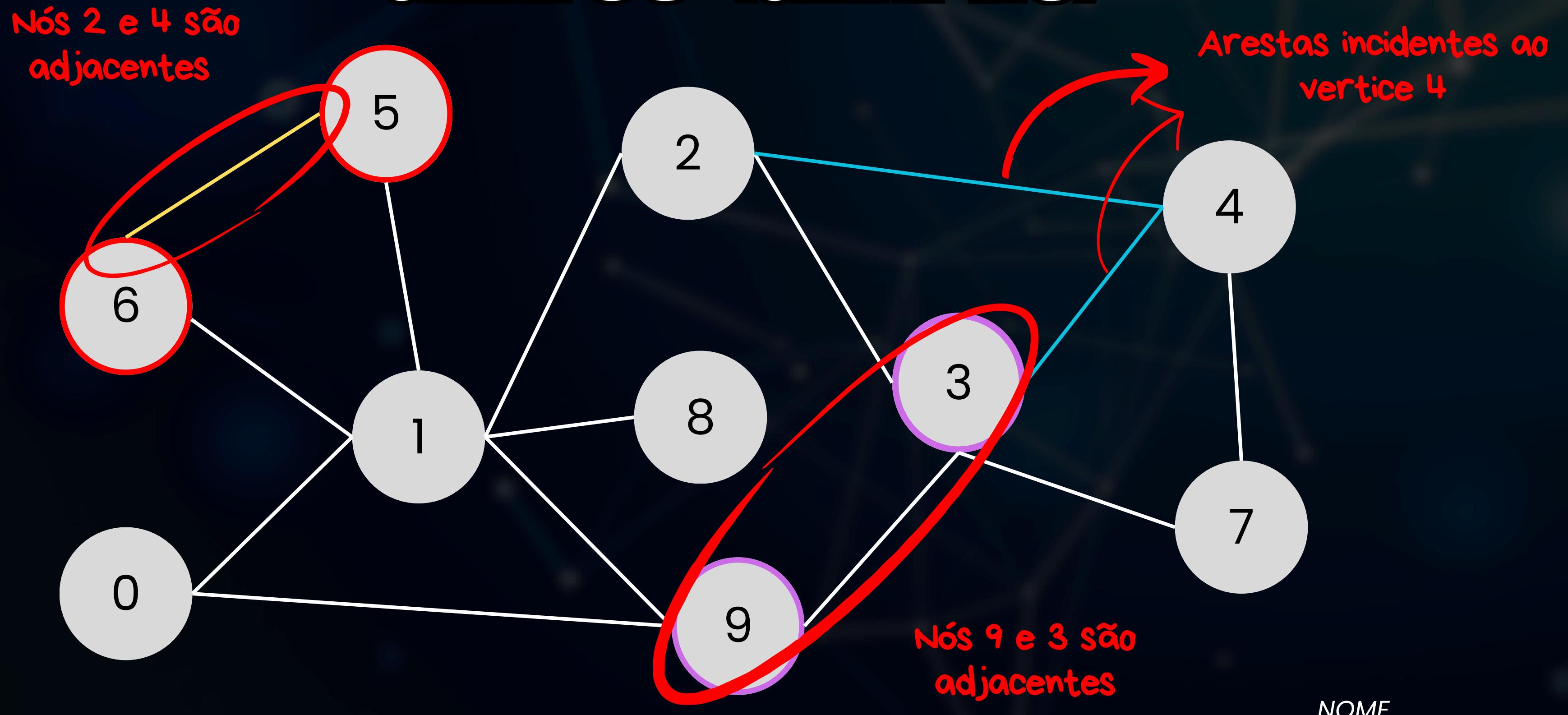
Arestas podem ser
ponderadas
(distâncias)

GRAFOS (GRAPHS)



NOME

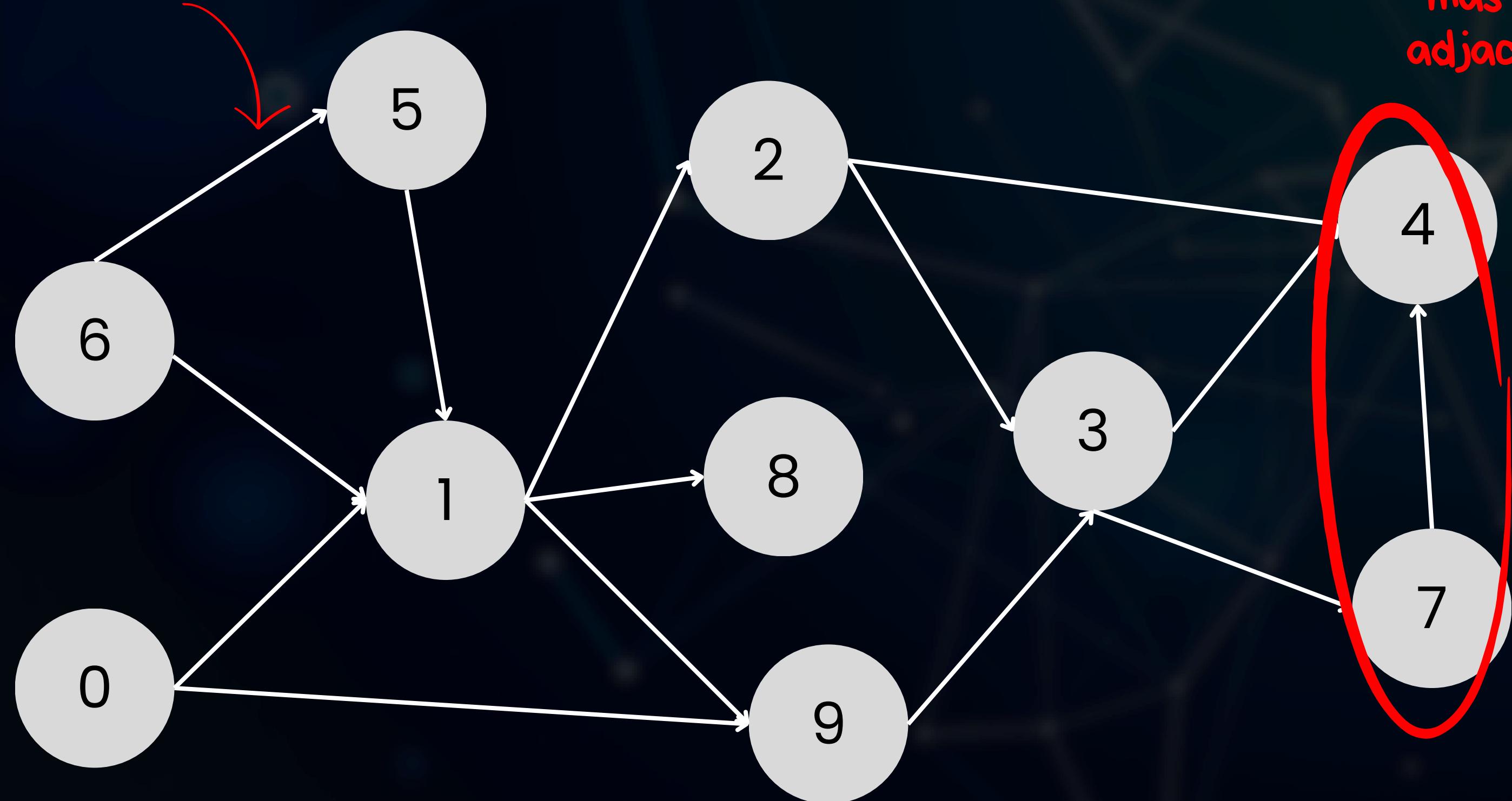
GRAFOS (GRAPHS)



DGRAFOS

de 6 para 5 e não de
5 para 6

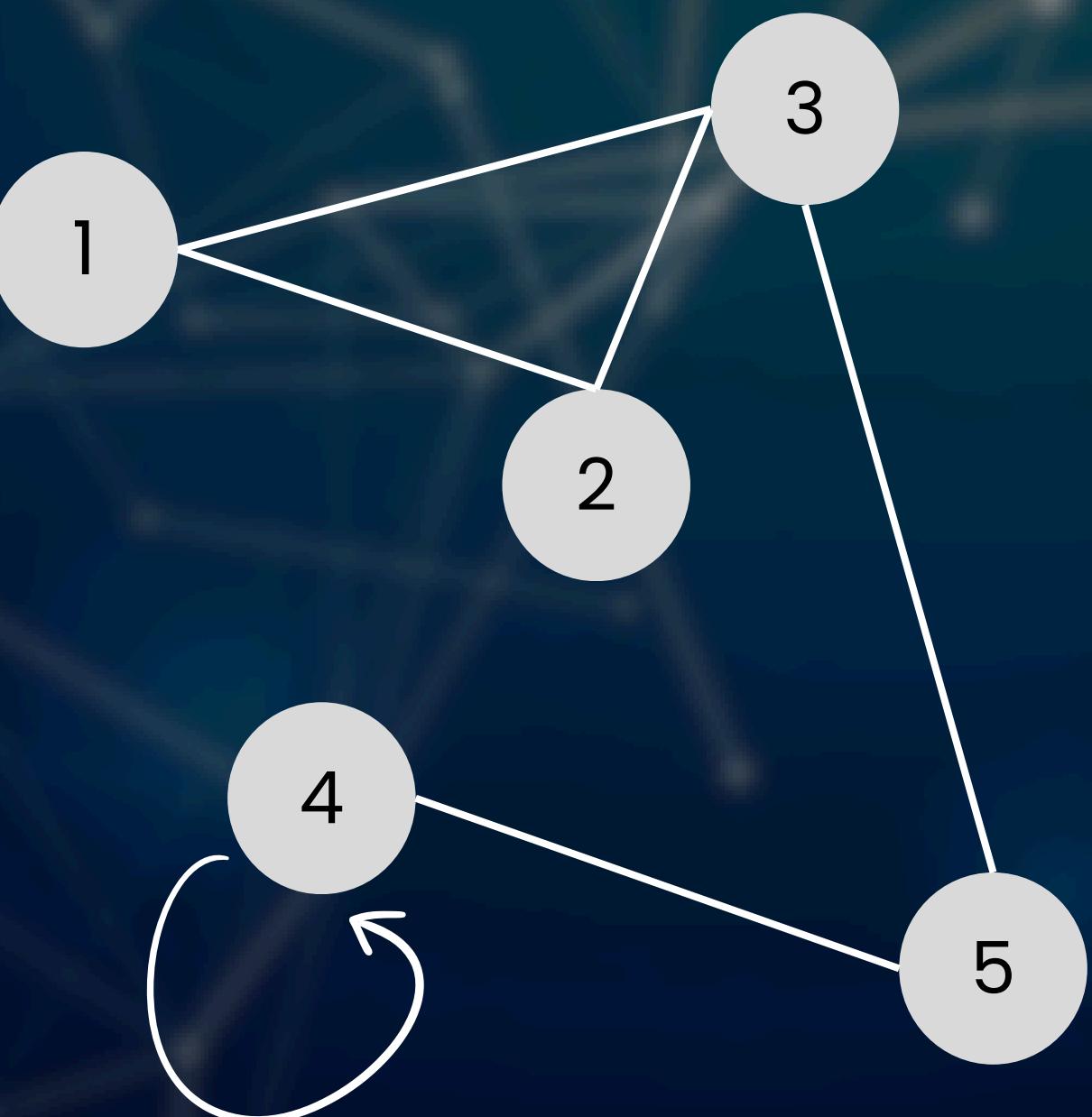
7 é adjacente a 4
mas 4 não é
adjacente a 7



NOME

GRAFOS (GRAPHS)

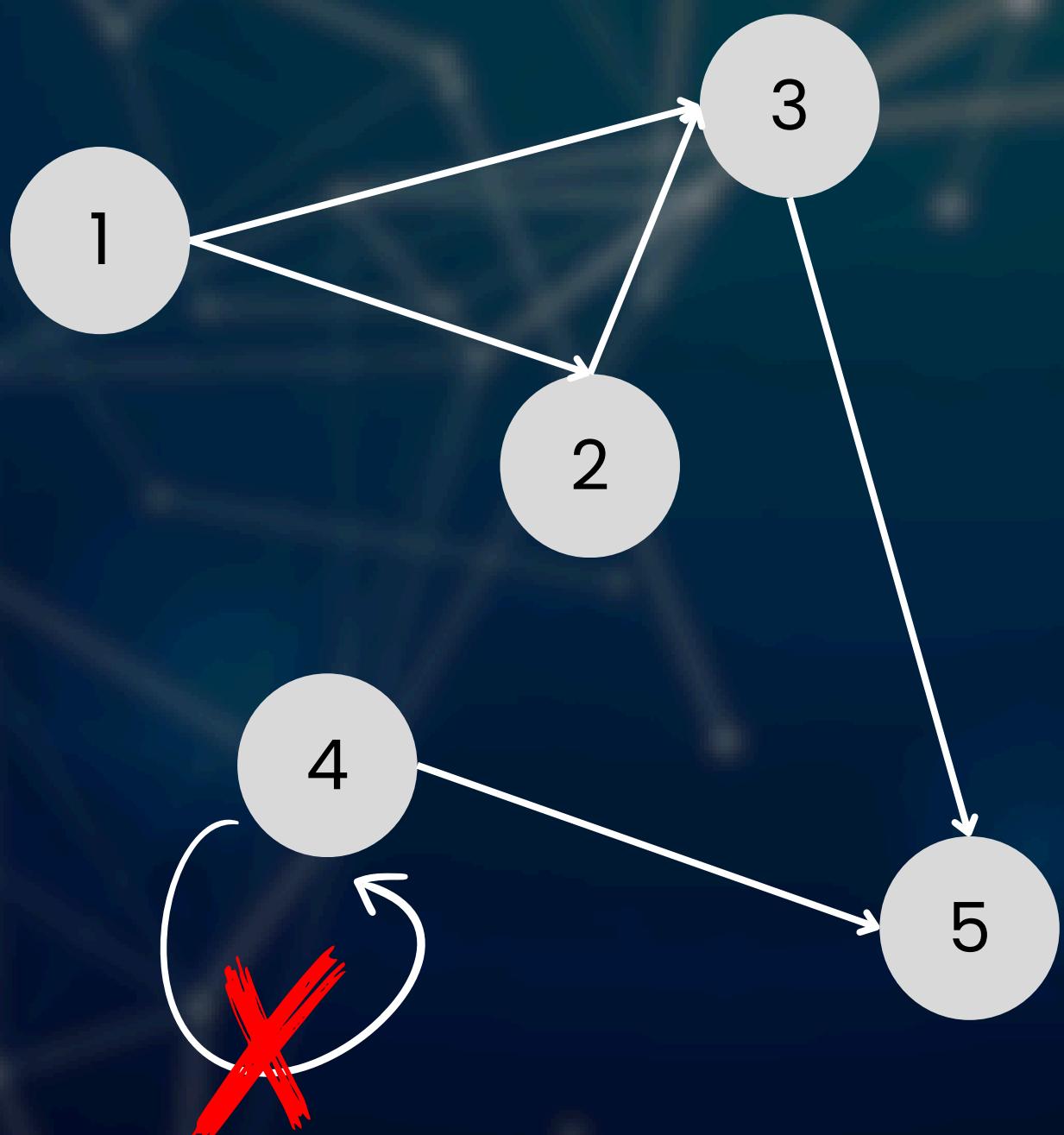
- Tem graus definidos para quantidade de vizinhos dos nó
- $\text{gr}(1) = \text{gr}(2) = \text{gr}(5) = 2$
- $\text{gr}(3) = \text{gr}(4) = 3$



NOME

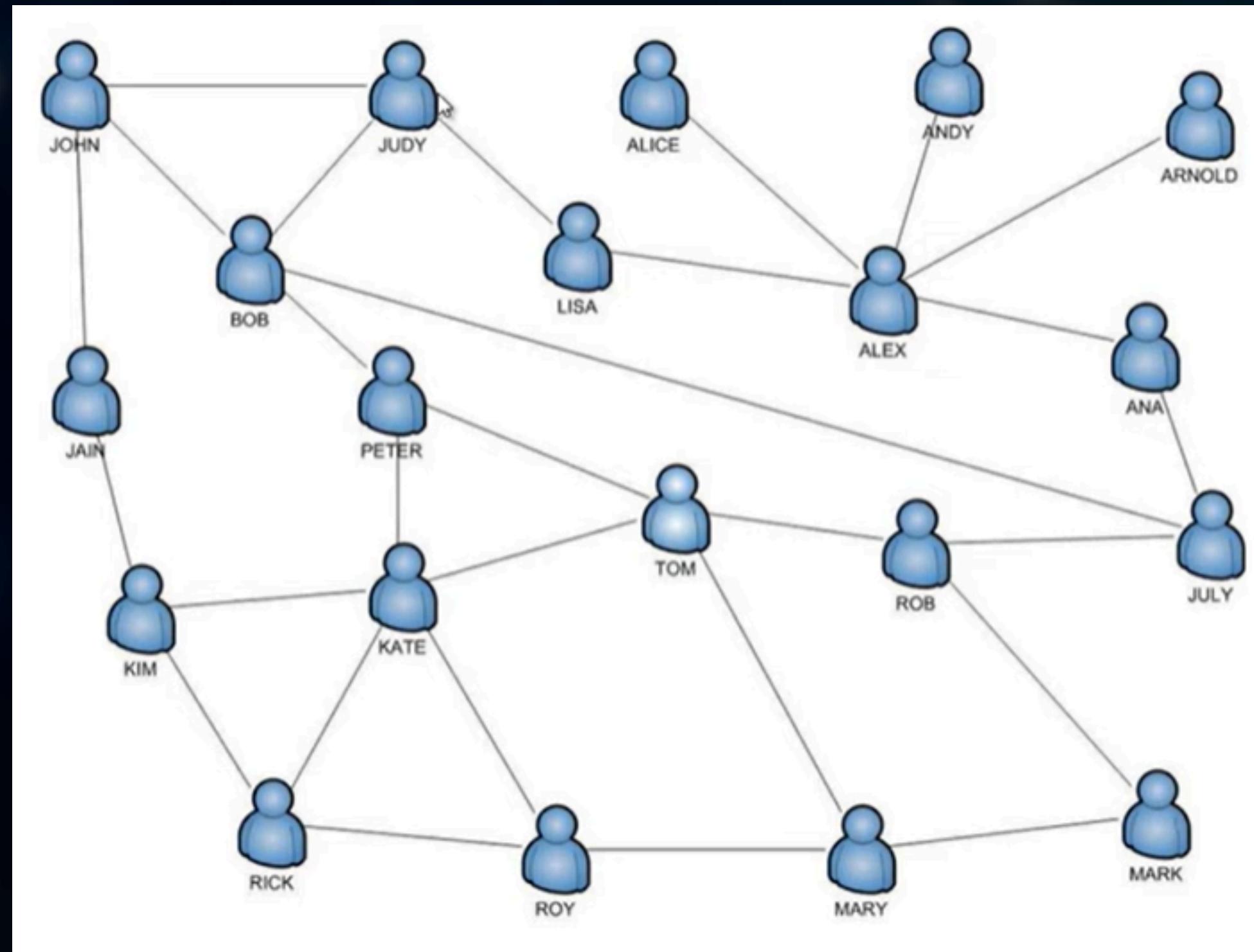
DGRAFOS

- Tem graus definidos pela quantidade de arestas que chegam no nó
- $\text{gr}(1) = \text{gr}(4) = 0$
- $\text{gr}(3) = \text{gr}(5) = 2$
- $\text{gr}(2) = 1$



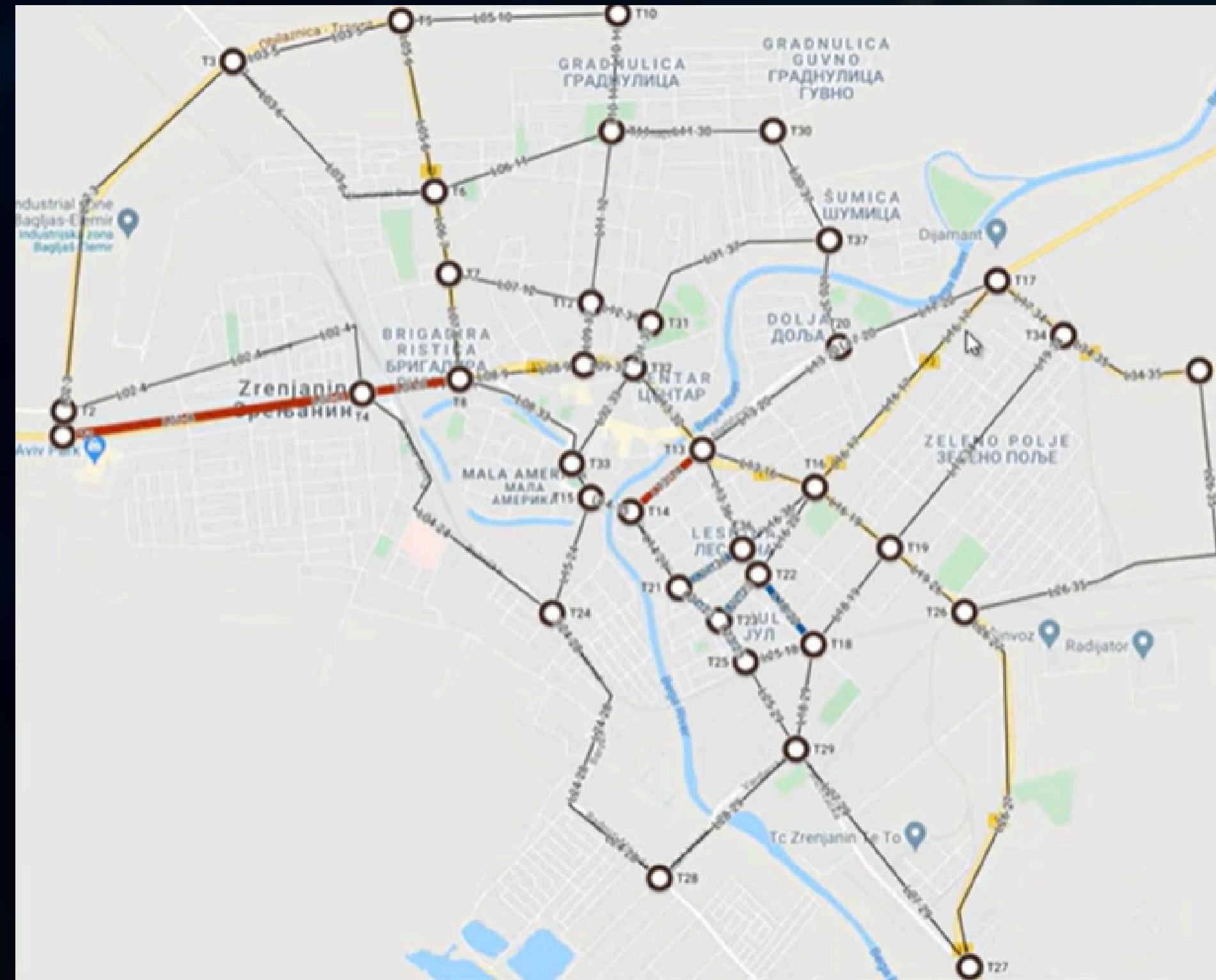
NOME

GRAFOS (GRAPHS)

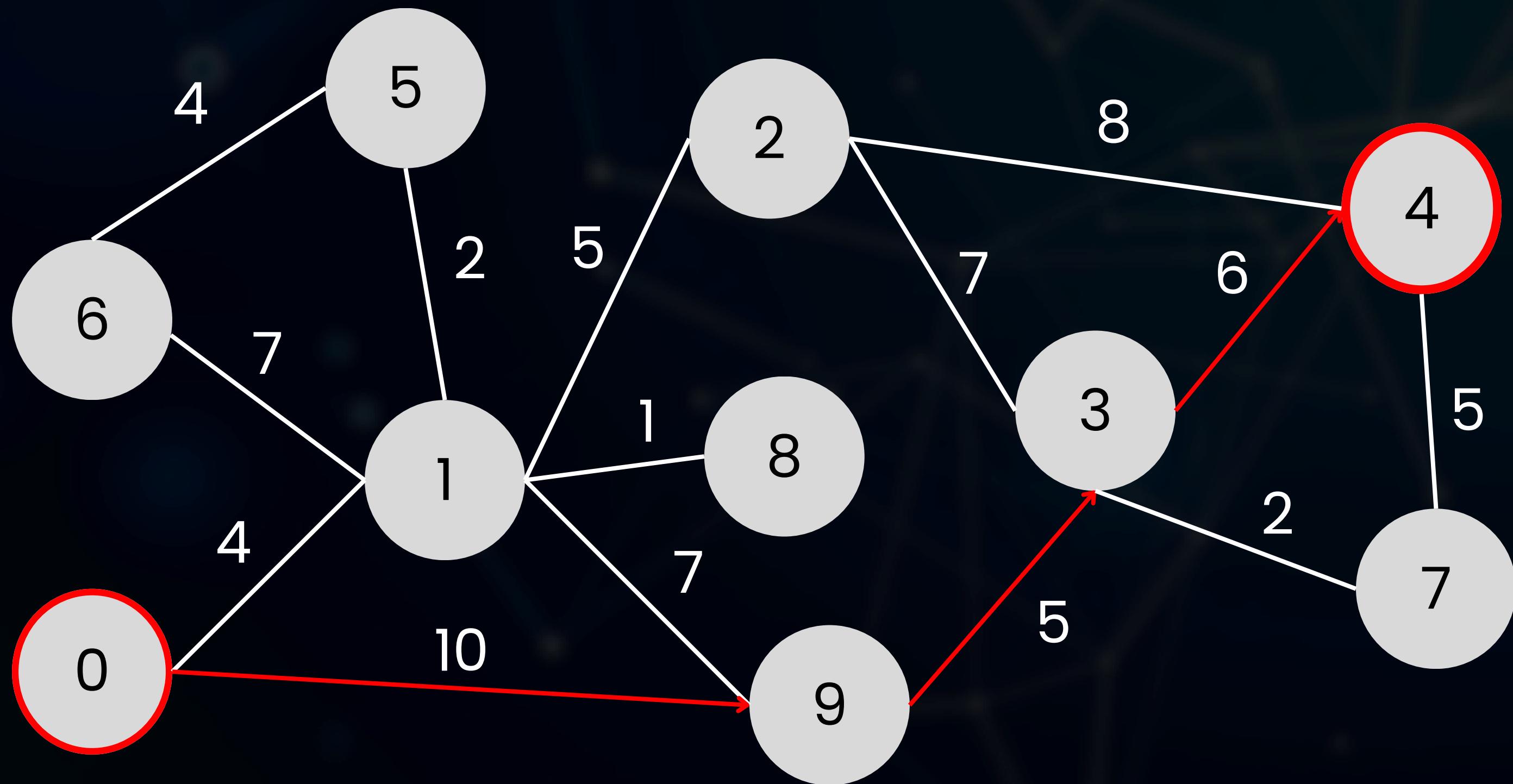


NOME

GRAFOS (GRAPHS)

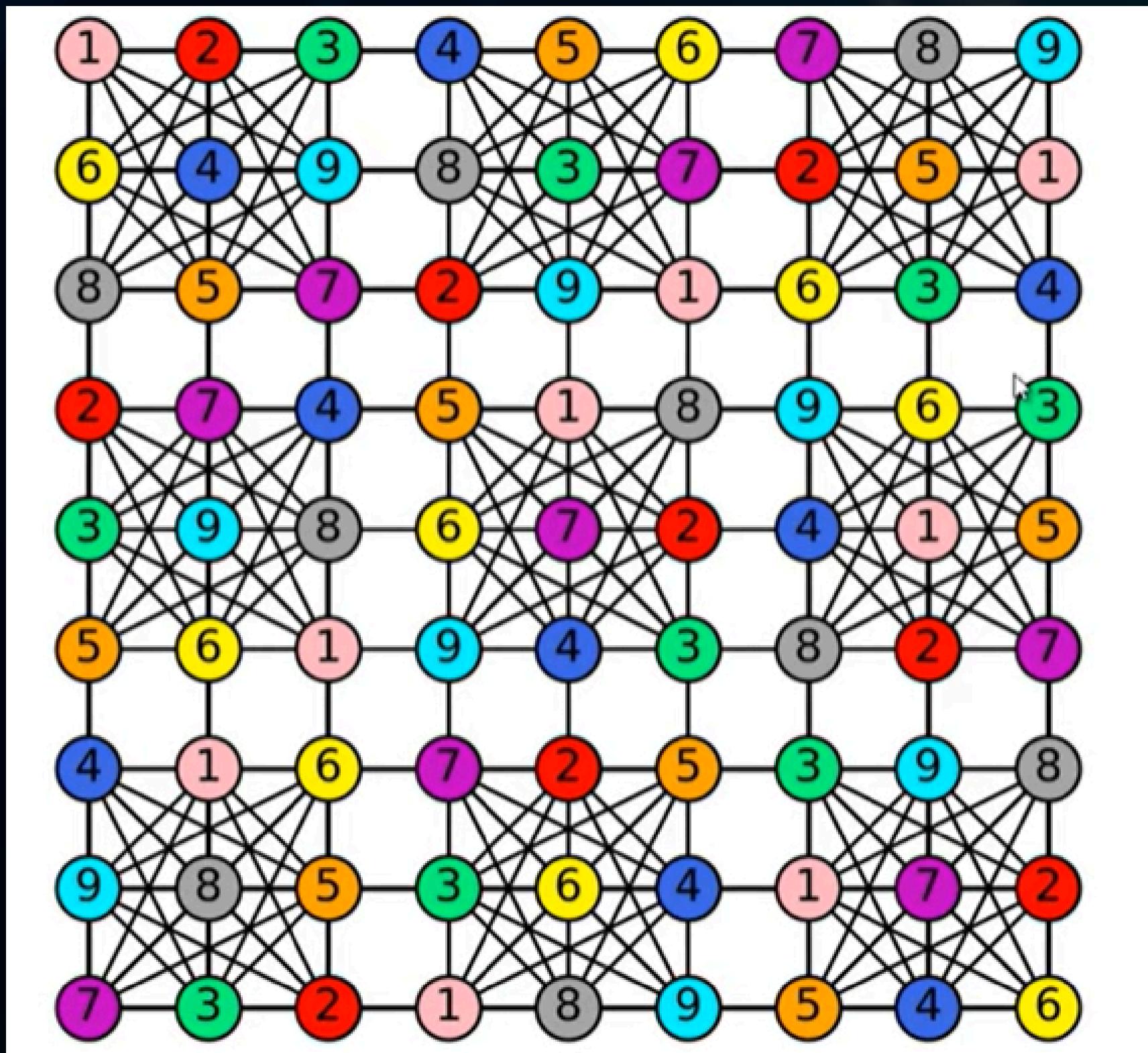


GRAFOS (GRAPHS)



NOME

GRAFOS (GRAPHS)



NOME

TABELAS HASH (HASH TABLES)



O que são?

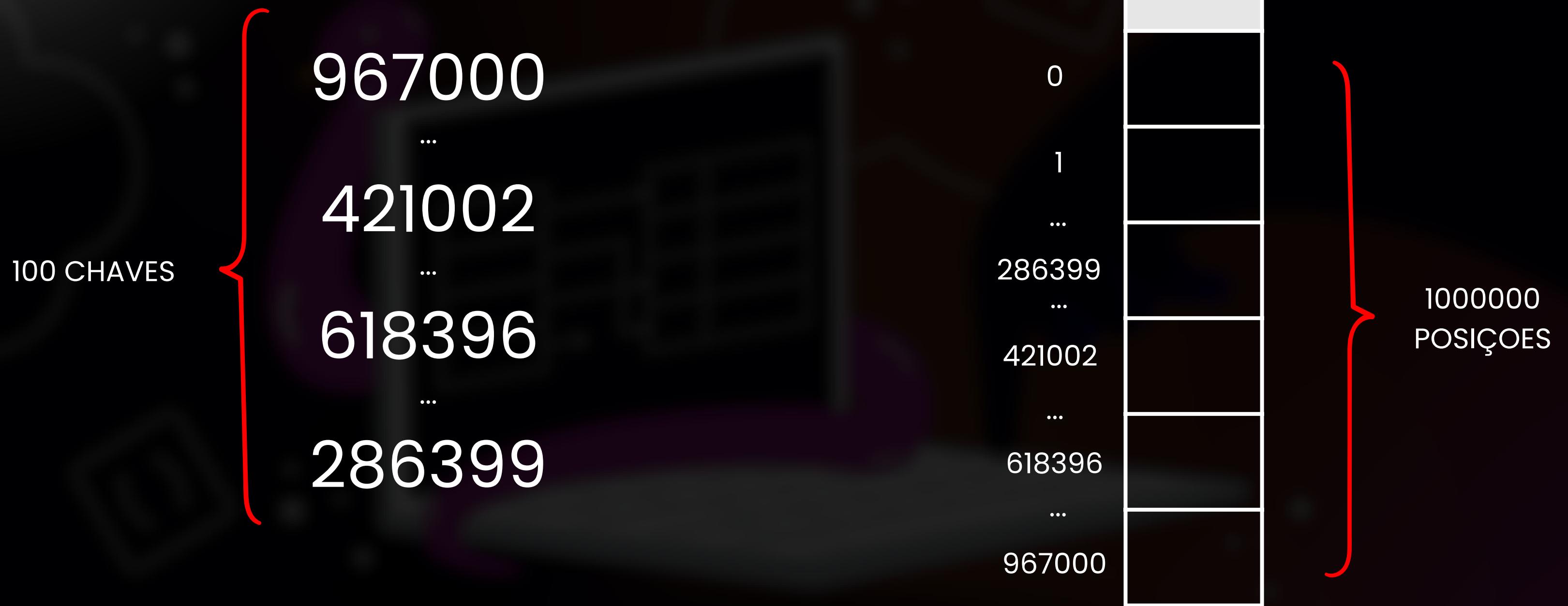
Estruturas que armazenam pares chave-valor com acesso rápido via função de hash (gaveteiro e chave)



Usos reais:

- Login de usuários (validação de senhas)
- Cache de dados
- Dicionários e mapas em linguagens de programação

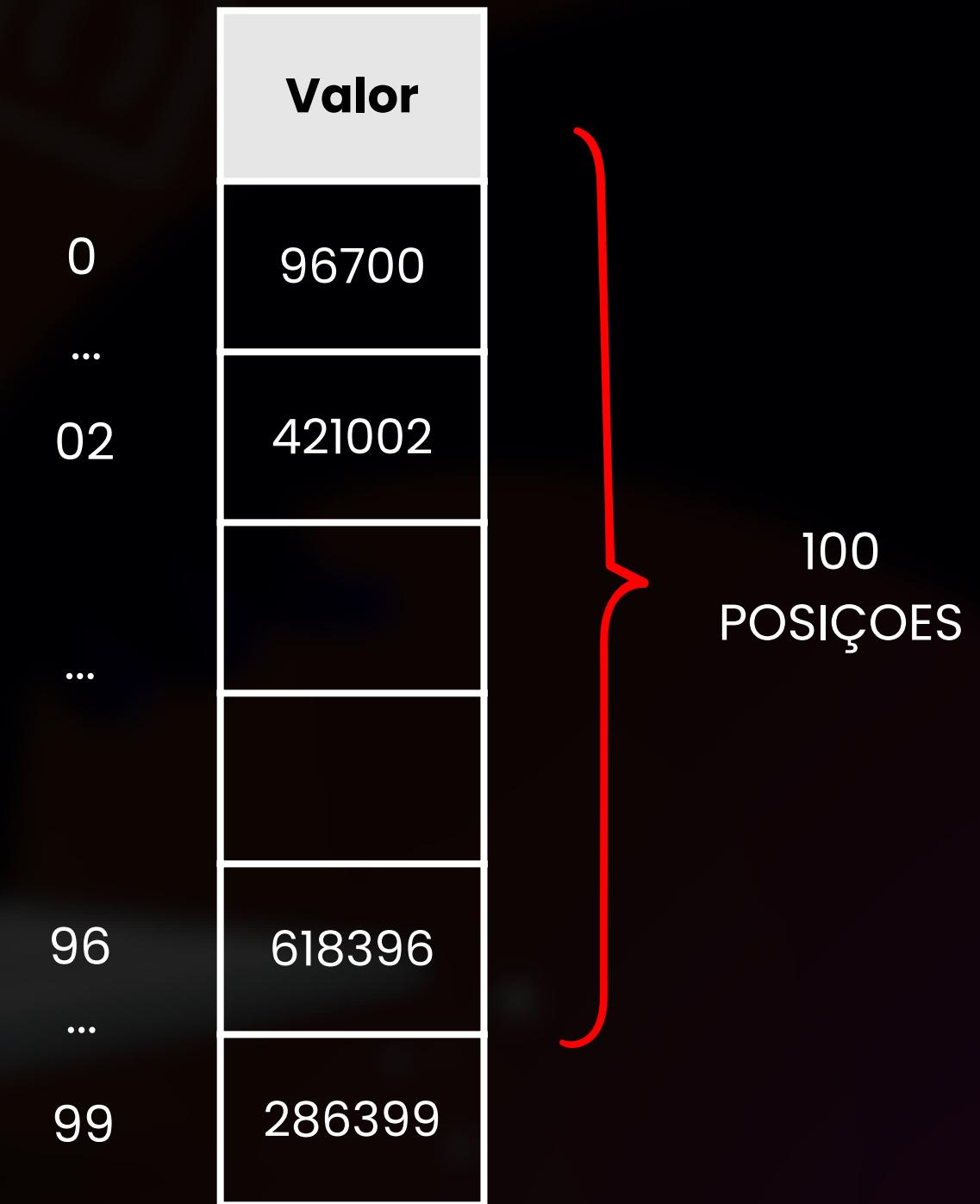
TABLAS HASH (HASH TABLES)



TABELAS HASH (HASH TABLES)



TABELAS HASH (HASH TABLES)



TABELAS HASH (HASH TABLES)

TAMANHO

$$M = 2 * 6 = 11 \text{ (primo mais proximo do resultado)}$$

FATOR DE CARGA

$$fc = 6/11 = 0,54$$

FUNÇÃO DE ESPALHAMENTO (HASH)

$$38 \% 11 = 5$$

$$8 \% 11 = 8$$

$$171 \% 11 = 6$$

$$24 \% 11 = 2$$

$$73 \% 11 = 7$$

$$40 \% 11 = 7$$

	0
	1
24	2
	3
	4
38	5
171	6
33	7
88	8
40	9
	10

TABELAS HASH (HASH TABLES)

TAMANHO

$$M = 2 * 6 = 11 \text{ (primo mais proximo do resultado)}$$

FATOR DE CARGA

$$fc = 6/11 = 0,54$$

FUNÇÃO DE ESPALHAMENTO (HASH)

BUSCAR O 40:

$$40 \% 11 = 7$$

	0
	1
24	2
	3
	4
38	5
171	6
33	7
88	8
40	9
	10

TABELAS HASH (HASH TABLES)

TAMANHO

$$M = 2 * 6 = 11 \text{ (primo mais proximo do resultado)}$$

FATOR DE CARGA

$$fc = 6/11 = 0,54$$

FUNÇÃO DE ESPALHAMENTO (HASH)

BUSCAR O 40:

$$29 \% 11 = 7$$

	0
	1
24	2
	3
	4
38	5
171	6
33	7
88	8
40	9
	10

TABELAS HASH (HASH TABLES)

TAMANHO

$$M = 2 * 6 = 11 \text{ (primo mais proximo do resultado)}$$

FATOR DE CARGA

$$fc = 6/11 = 0,54$$

FUNÇÃO DE ESPALHAMENTO (HASH)

$$38 \% 11 = 5$$

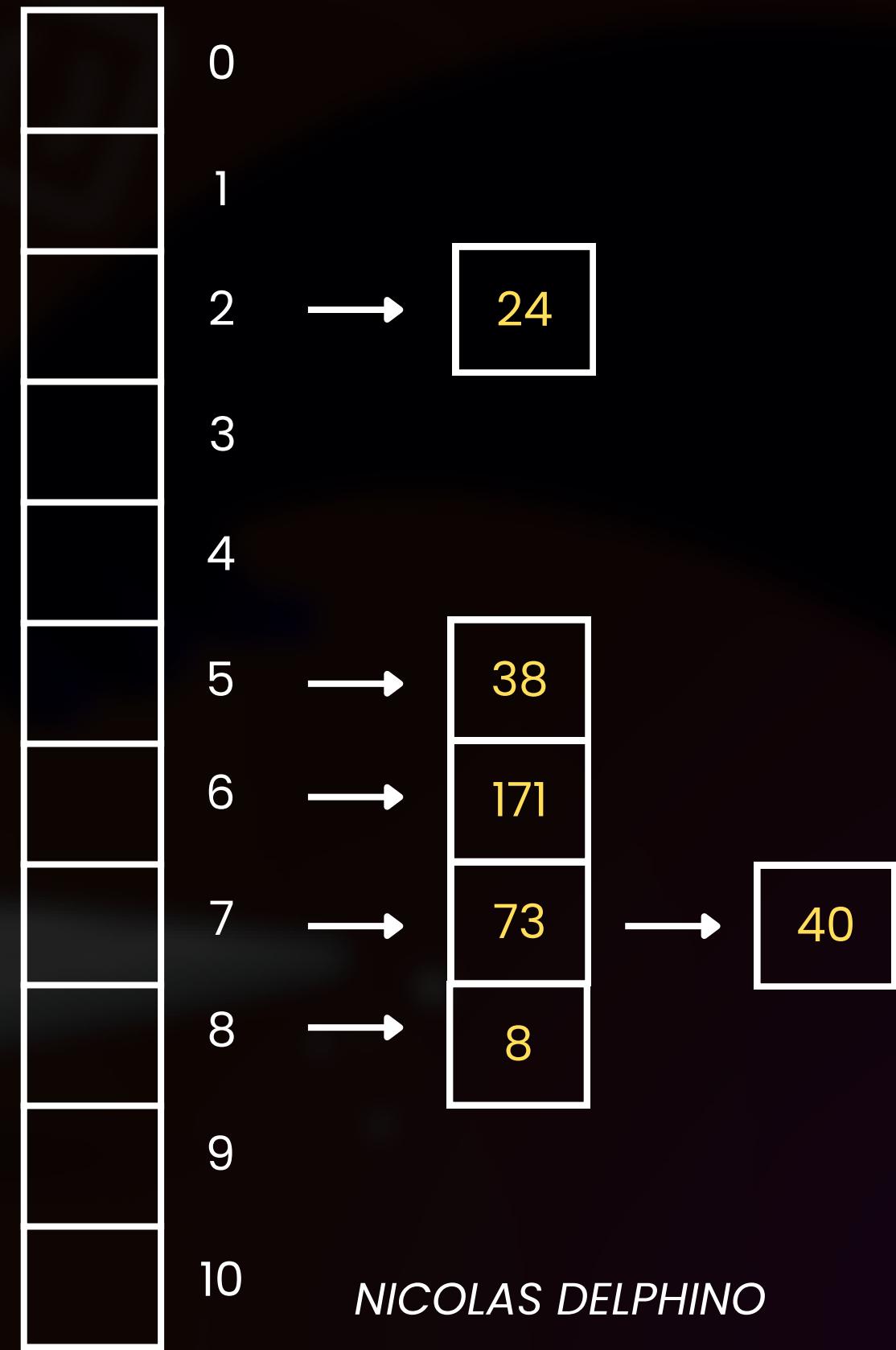
$$8 \% 11 = 8$$

$$171 \% 11 = 6$$

$$24 \% 11 = 2$$

$$73 \% 11 = 7$$

$$40 \% 11 = 7$$



TABELAS HASH (HASH TABLES)

diccionários

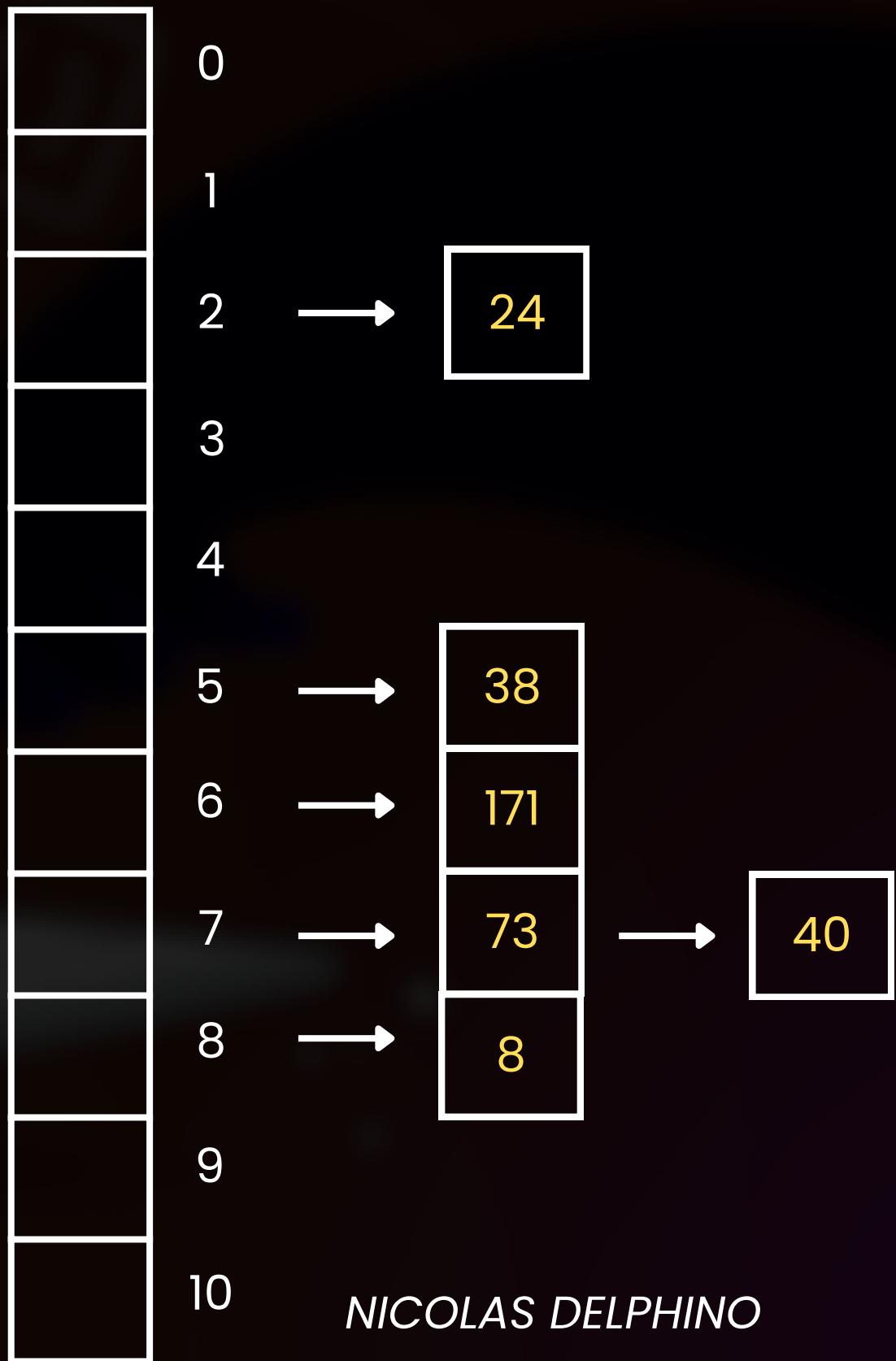
{“Nome”: “João”, “Idade” : 20}

BANCO DE DADOS (ÍNDICES DE ACESSO)

Permitem encontrar registros rapidamente

CRIPTOGRAFIA E SEGURANÇA

Funções hash como SHA-256 transformam senhas em códigos únicos.



SISTEMA DE GESTÃO DE FUNCIONÁRIOS

Estruturas de Dados Aplicadas



Lista (Array):

Armazena todos os funcionários cadastrados no sistema.



Tabela Hash (Map):

Relaciona rapidamente cada funcionário ao seu departamento através de um departamentold.



Árvore (opcional para escalar):

Para buscas hierárquicas por cargos, departamentos ou localização geográfica.



Pilha:

Armazena histórico de ações para permitir desfazer alterações (ex: remoção acidental de funcionário).

NOME

Benefícios no Sistema Real

- ⌚ Performance eficiente nas buscas e filtragens de funcionários.
- * Vínculo direto entre funcionário e departamento.
- 🧠 Facilidade de manutenção com estrutura clara.
- 💾 Uso de memória otimizado, com listas e mapas bem definidos.

IMPACTO NO DESEMPENHO

Por que a escolha da estrutura importa?

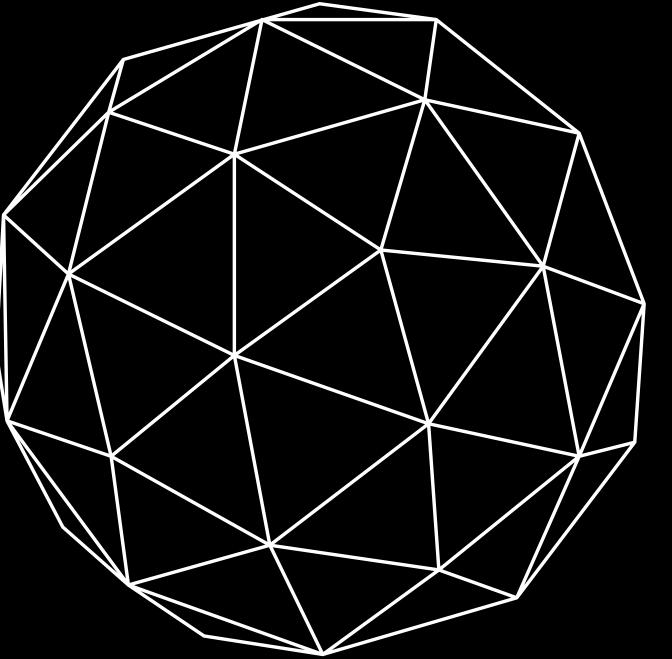
A estrutura de dados certa reduz o tempo de execução, melhora o uso da memória e deixa o sistema mais escalável.

⌚ Exemplo prático: busca de funcionário

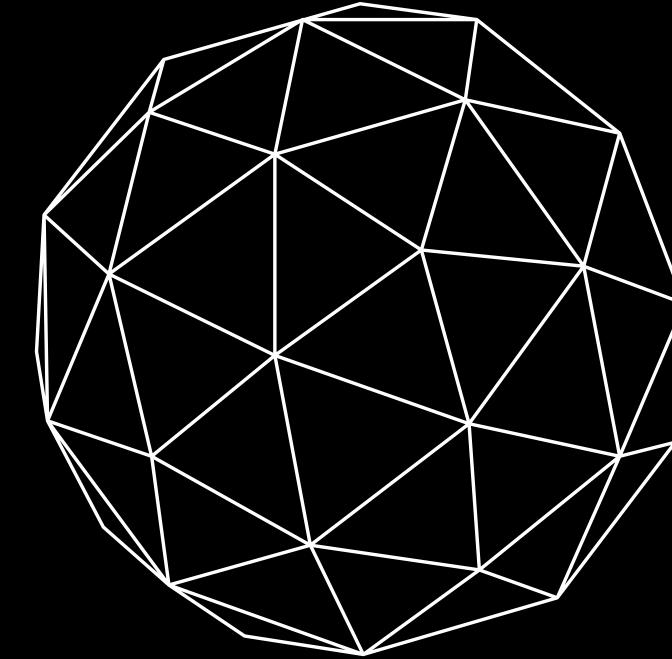
- Usando uma lista comum (Array):
 - Precisa percorrer todos os itens → $O(n)$
- Usando uma tabela hash:
 - Acesso direto pela chave (CPF, ID) → $O(1)$

⚠ Escolhas erradas custam caro!

- Mais lentidão no sistema
- Gastos maiores com recursos de hardware
- Usuário frustrado com a experiência



CONCLUSÃO

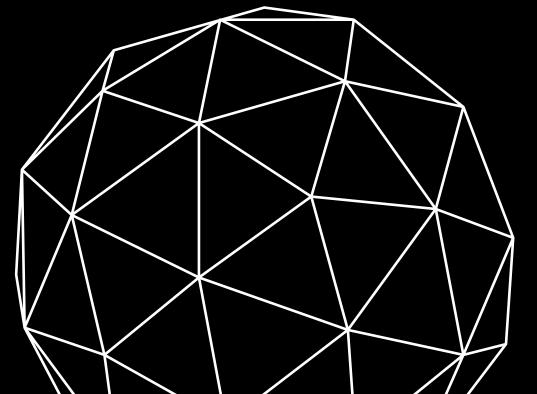


O que aprendemos?

Estruturas de dados estão em todos os sistemas reais.

Elas ajudam a organizar, buscar e manipular informações com eficiência.

Cada estrutura tem seu uso ideal dependendo do problema.



NOME

OBRIGADO