

Estruturas de Dados Lineares

As estruturas de dados lineares são o fundamento da ciência da computação. Elas servem como base para algoritmos eficientes e são amplamente utilizadas em sistemas reais.

 **por Mayana Duarte**

O que são Estruturas de Dados?



Organização

Organizam e armazenam dados de forma estruturada e lógica.



Acesso Eficiente

Permitem que os dados sejam acessados e manipulados rapidamente.



Desempenho

Influenciam diretamente o desempenho e a velocidade dos programas.



Data

Classificação das Estruturas de Dados

Estruturas Lineares

Elementos organizados em sequência, um após o outro.

- Listas
- Filas
- Pilhas

Estruturas Não-Lineares

Elementos sem ordenação sequencial clara.

- Árvores
- Grafos
- Tabelas Hash

Importância das Estruturas Lineares



Aprendizado Fundamental

Base para entender conceitos mais complexos.



Operações Previsíveis

Inserção, remoção e busca seguem padrões claros.



Aplicações Amplas

Usadas em algoritmos básicos e intermediários.



Visão Geral das Estruturas Lineares

Listas

Coleção ordenada de elementos acessíveis por índice.

Pilhas

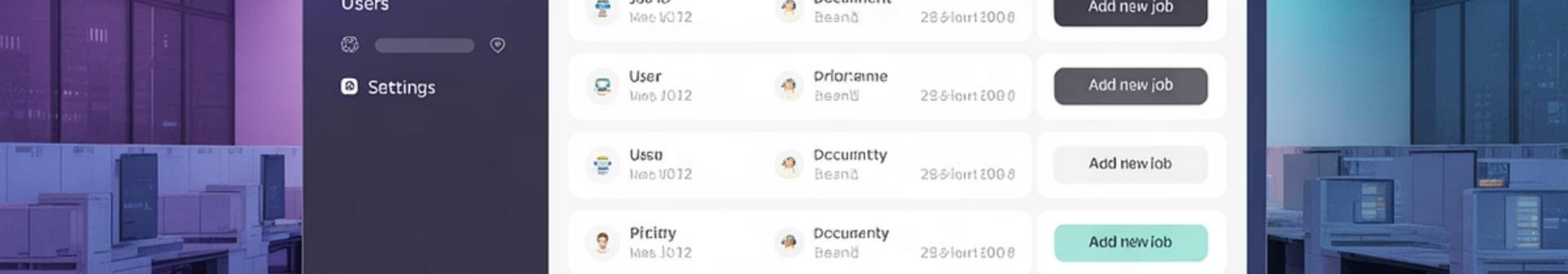
Estrutura LIFO (Last In, First Out) com acesso restrito ao topo.

Filas

Estrutura FIFO (First In, First Out) com inserção no final e remoção no início.

Deque

Filas duplas com inserção e remoção em ambas as extremidades.



Aplicações Reais



Pilha de Execução

Gerencia chamadas de funções em linguagens de programação.



Sistemas de Impressão

Filas organizam documentos para impressão ordenada.



Gerenciamento de Tarefas

Listas para controle de pendências e atividades.



Vantagens e Desvantagens

Vantagens

Simplicidade de implementação

Baixo consumo de memória

Eficiência em casos específicos

Desvantagens

Limitações de flexibilidade

Restrições de operações

Desempenho variável com tamanho

Estruturas Estáticas vs. Dinâmicas



Arrays

Tamanho fixo, alocação contínua, acesso rápido.



Listas Ligadas

Tamanho dinâmico, alocação dispersa, flexível.



Compromisso

Escolha baseada em requisitos específicos.



Introdução às Listas Lineares

Definição



Coleção ordenada de elementos do mesmo tipo em sequência.

Operações Básicas



- Inserir elementos
- Remover elementos
- Buscar elementos

Indexação



Acesso direto aos elementos por posição numérica.

Arrays (Vetores)

Alocação Contínua

Elementos armazenados em blocos adjacentes de memória.



Acesso Rápido

Tempo constante $O(1)$ para acessar qualquer elemento.



Operações Custosas

Inserção e remoção podem exigir realocação de elementos.



Aplicação

Ideal para listas com tamanho conhecido, como registros de alunos.



Operações em Arrays



Busca

- Sequencial: $O(n)$
- Binária (ordenada): $O(\log n)$



Inserção

- Final: $O(1)$
- Início/meio: $O(n)$



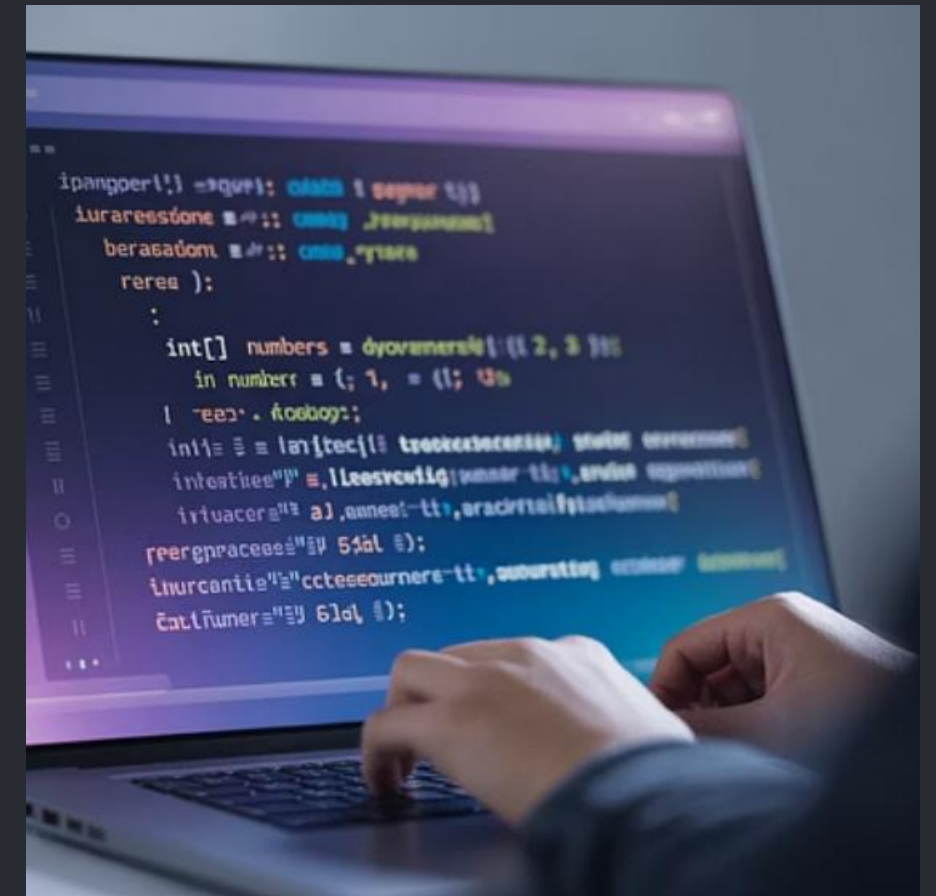
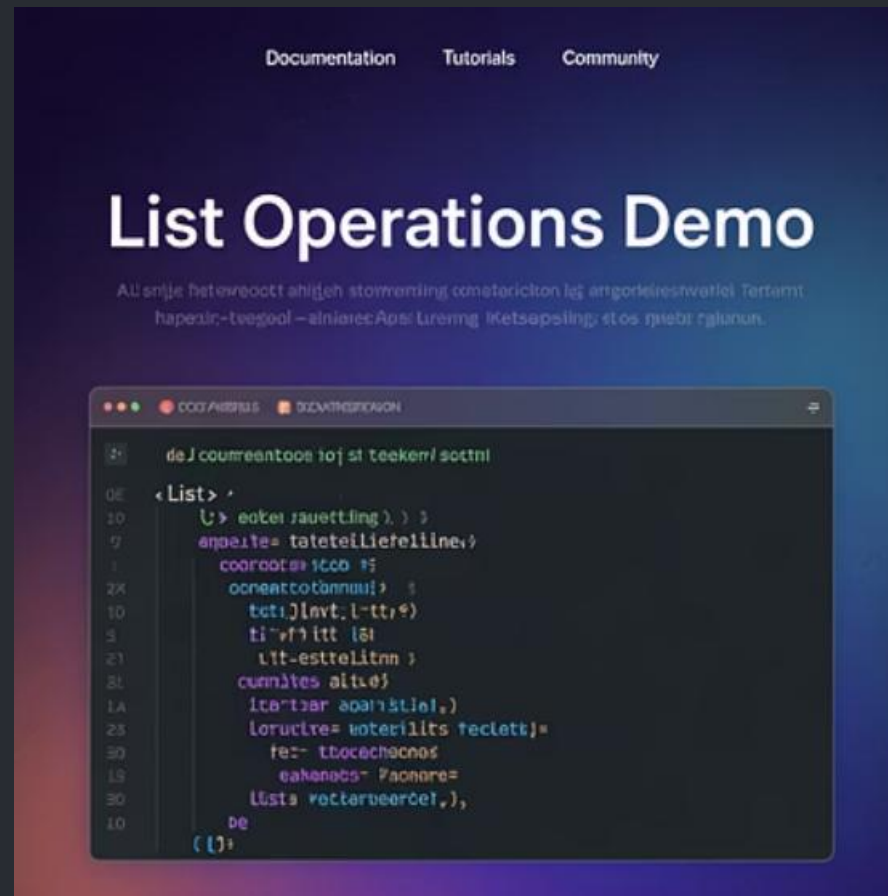
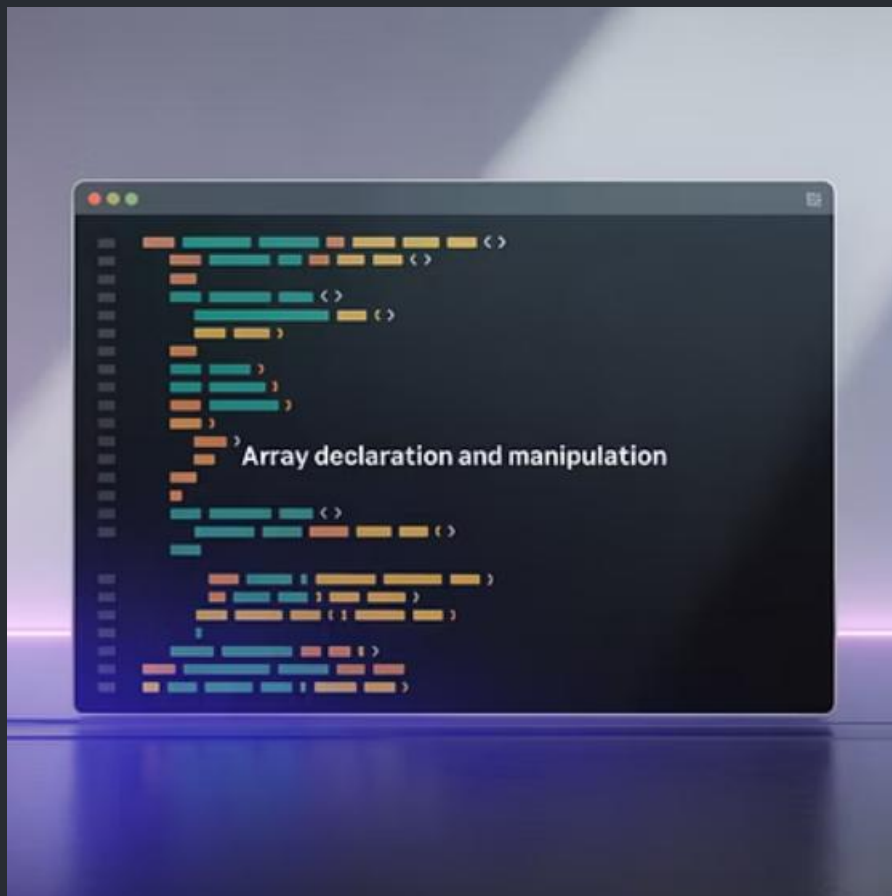
Remoção

- Final: $O(1)$
- Início/meio: $O(n)$

Array Operations



Uso de Arrays em Linguagens Populares



As linguagens modernas oferecem implementações diversas de arrays. C usa tamanho fixo. Python tem listas dinâmicas. Java combina ambas abordagens com vários tipos de coleções.

Listas Dinâmicas

0

Tamanho Inicial

Começa vazia ou com capacidade mínima.

2x

Fator de Crescimento

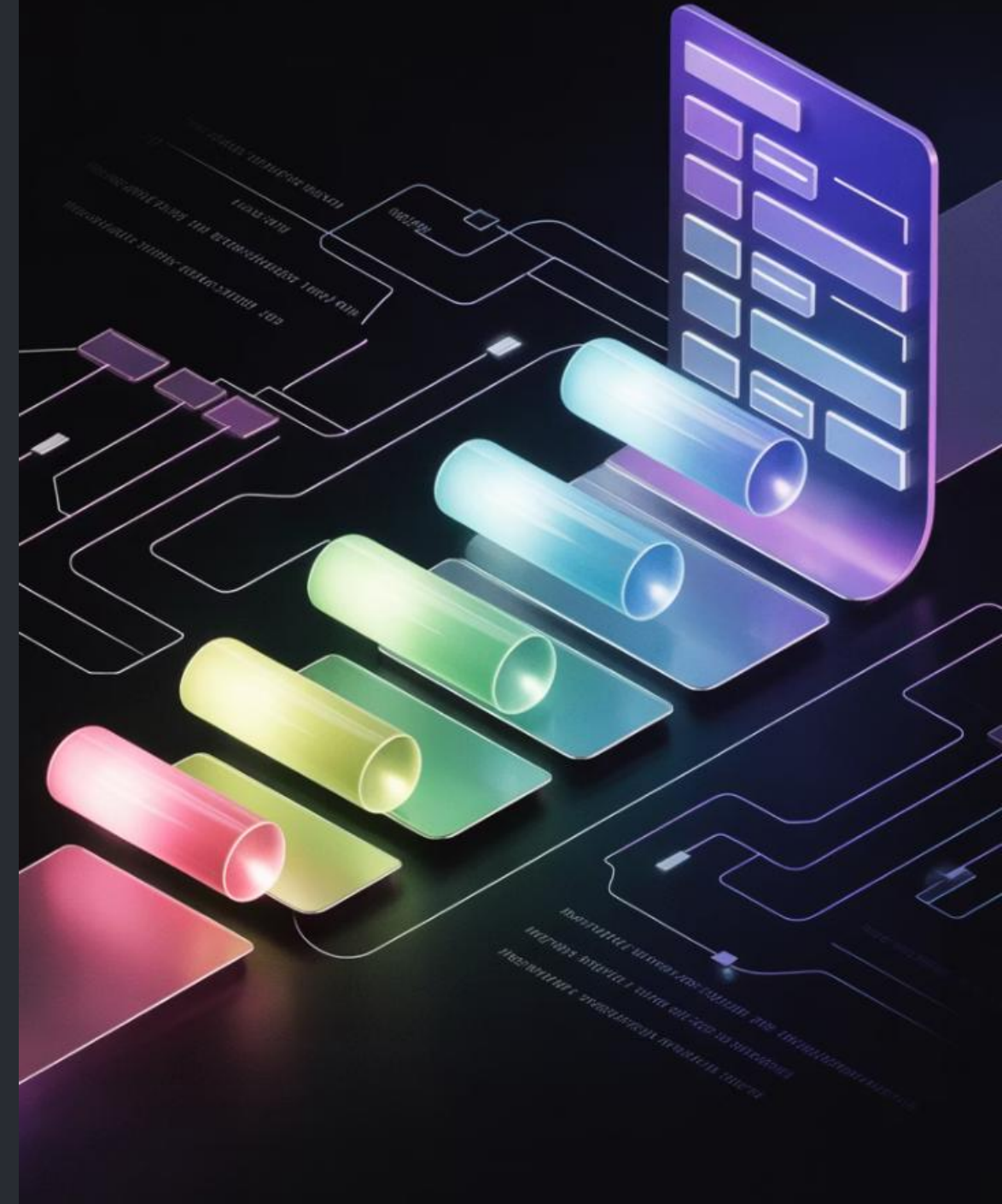
Aumento de capacidade quando necessário.

∞

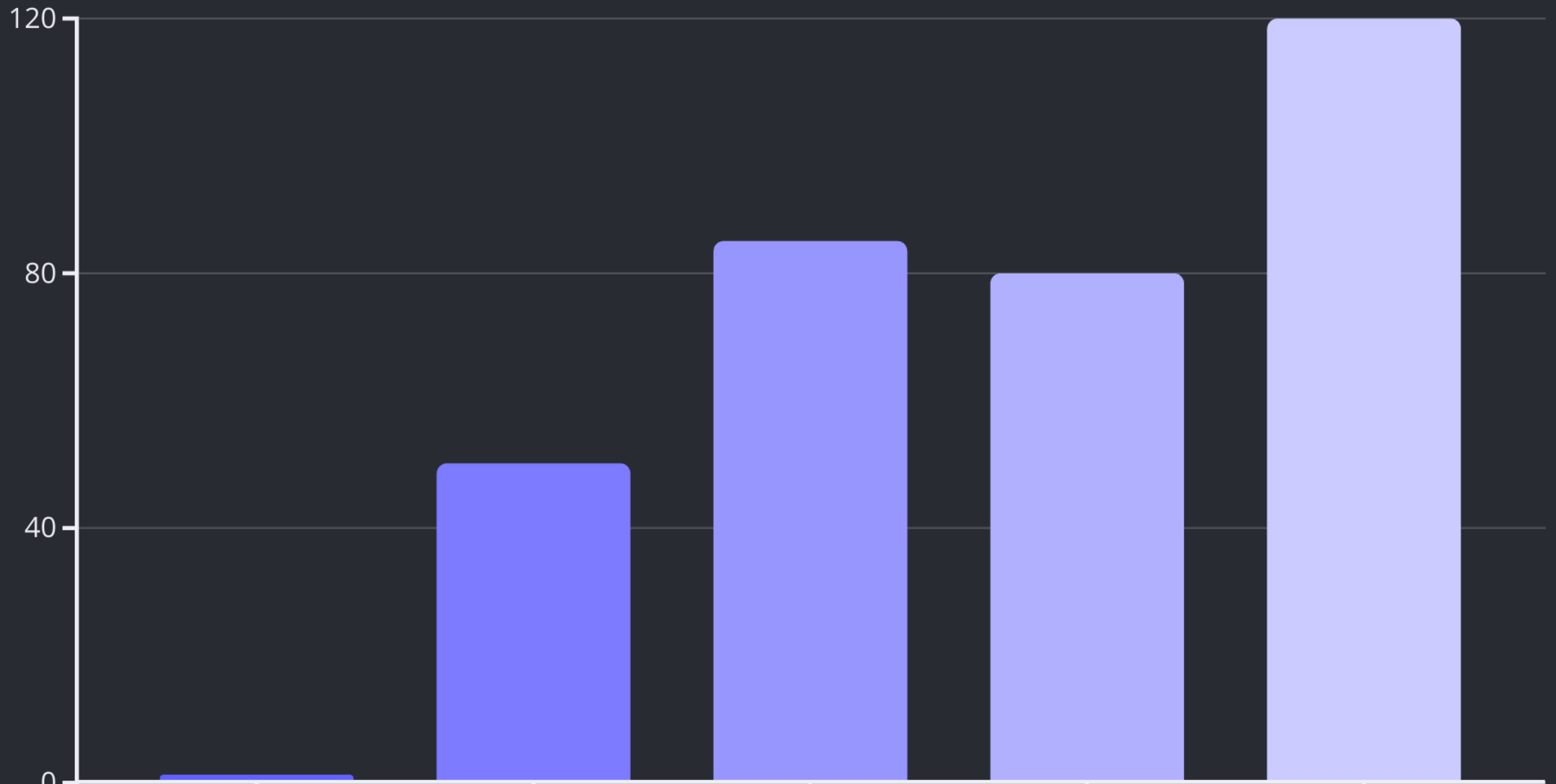
Limite Teórico

Limitado apenas pela memória disponível.

Memory Reallocation



Eficiência e Análise de Arrays





Pilhas: Stack

Princípio LIFO

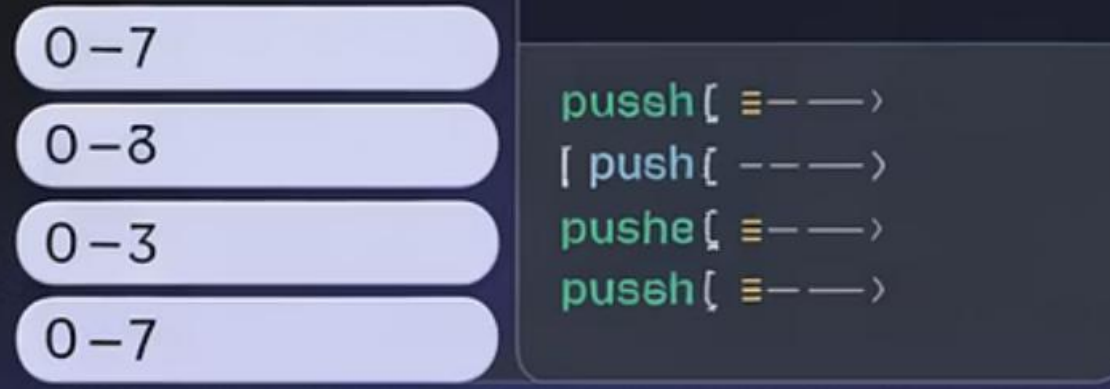
Last In, First Out - O último elemento inserido é o primeiro a sair.

Restrição de Acesso

Operações ocorrem apenas no topo da pilha.

Operações Básicas

- Push: Inserir no topo
- Pop: Remover do topo
- Peek/Top: Consultar o topo



Implementação de Pilhas com Arrays



Array Base

Estrutura subjacente para armazenamento.



Índice de Topo

Variável que controla a posição do último elemento.



Limitação

Tamanho máximo definido na criação.

Implementação de Pilhas com Listas Ligadas



Estrutura Básica

Cada nó contém dado e referência para o próximo elemento.



Crescimento Dinâmico

Novas posições são alocadas conforme necessário.



Sem Limite Fixo

Crescimento limitado apenas pela memória disponível no sistema.

Aplicações Práticas de Pilhas

Controle de Funções

Gerenciamento de chamadas e retornos de funções.



Navegação

Histórico de navegação em sites e aplicativos.



Undo/Redo

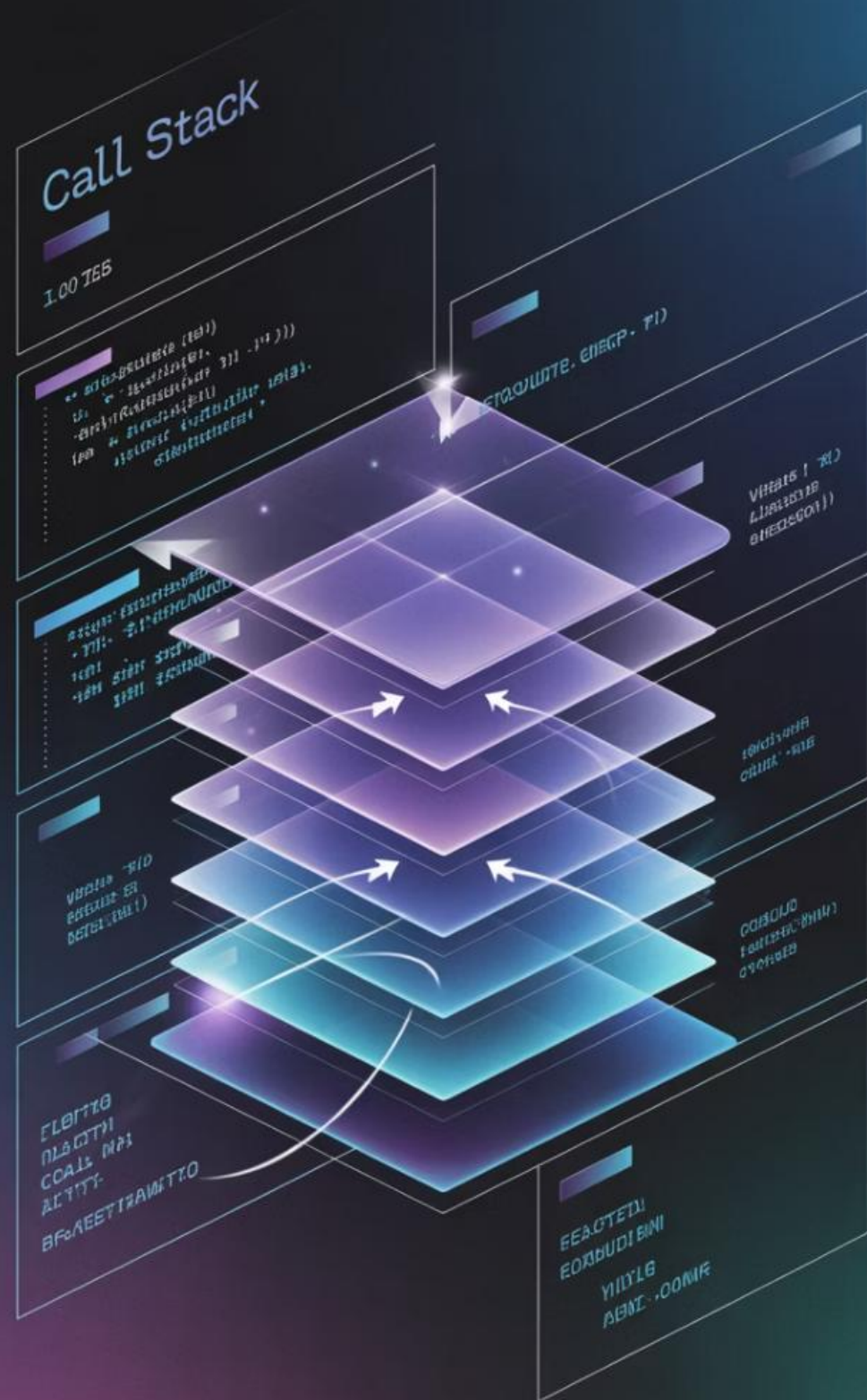
Histórico de alterações em editores de texto e imagem.



Expressões Matemáticas

Avaliação e conversão de expressões infixa para posfixa.





Exemplo: Pilha de Execução



Chamada Principal

Função main() é colocada na base da pilha.



Chamadas Aninhadas

Cada nova função é empilhada acima da anterior.

3

Retornos

Funções são desempilhadas ao completar execução.



Stack Overflow

Limite atingido em recursão profunda ou infinita.

Algoritmo: Conversão de Expressão Infixa para Posfixa

Expressão Infixa

$A + B * C \rightarrow$ Notação matemática comum com operadores entre operandos.

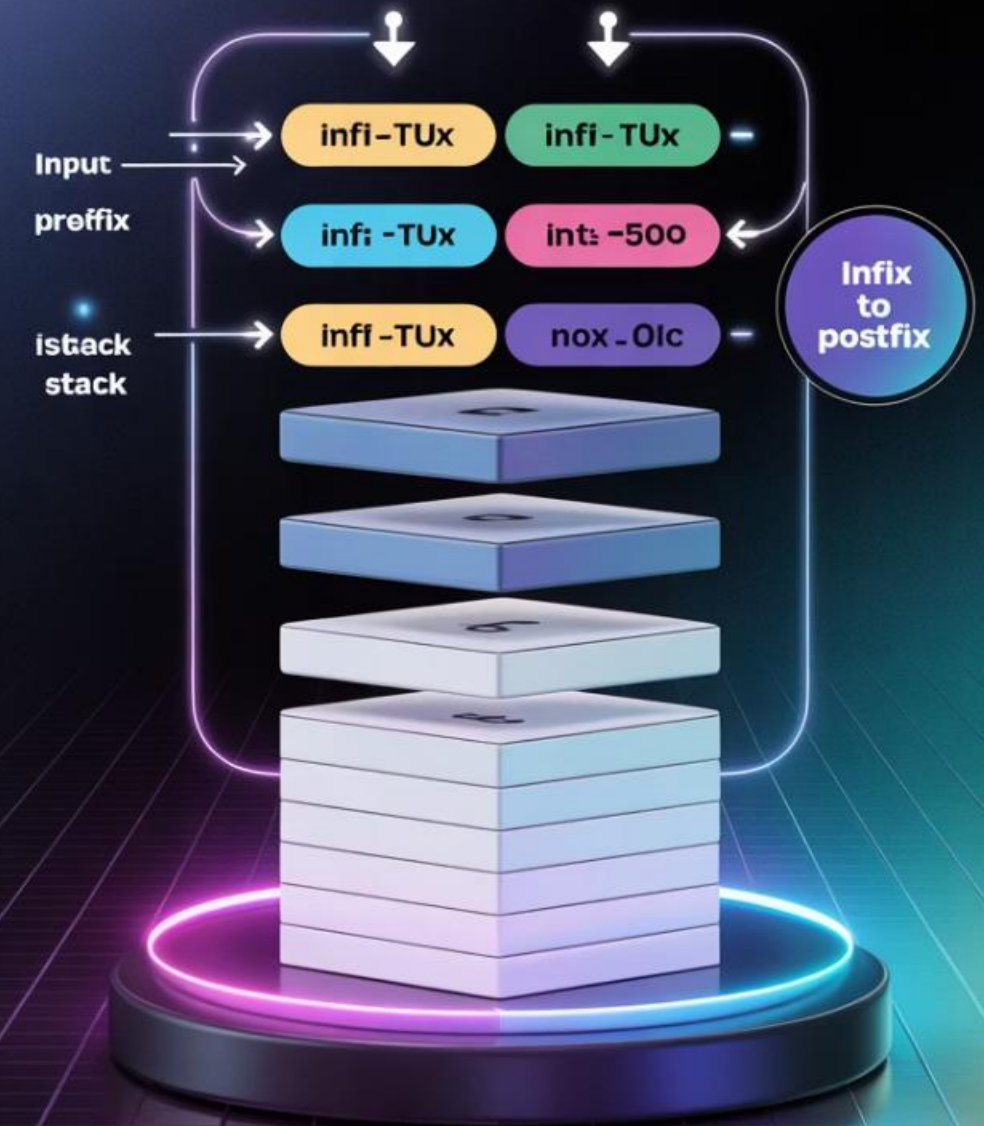
Expressão Posfixa

$A B C * + \rightarrow$ Notação polonesa reversa com operadores após operandos.

Processo de Conversão

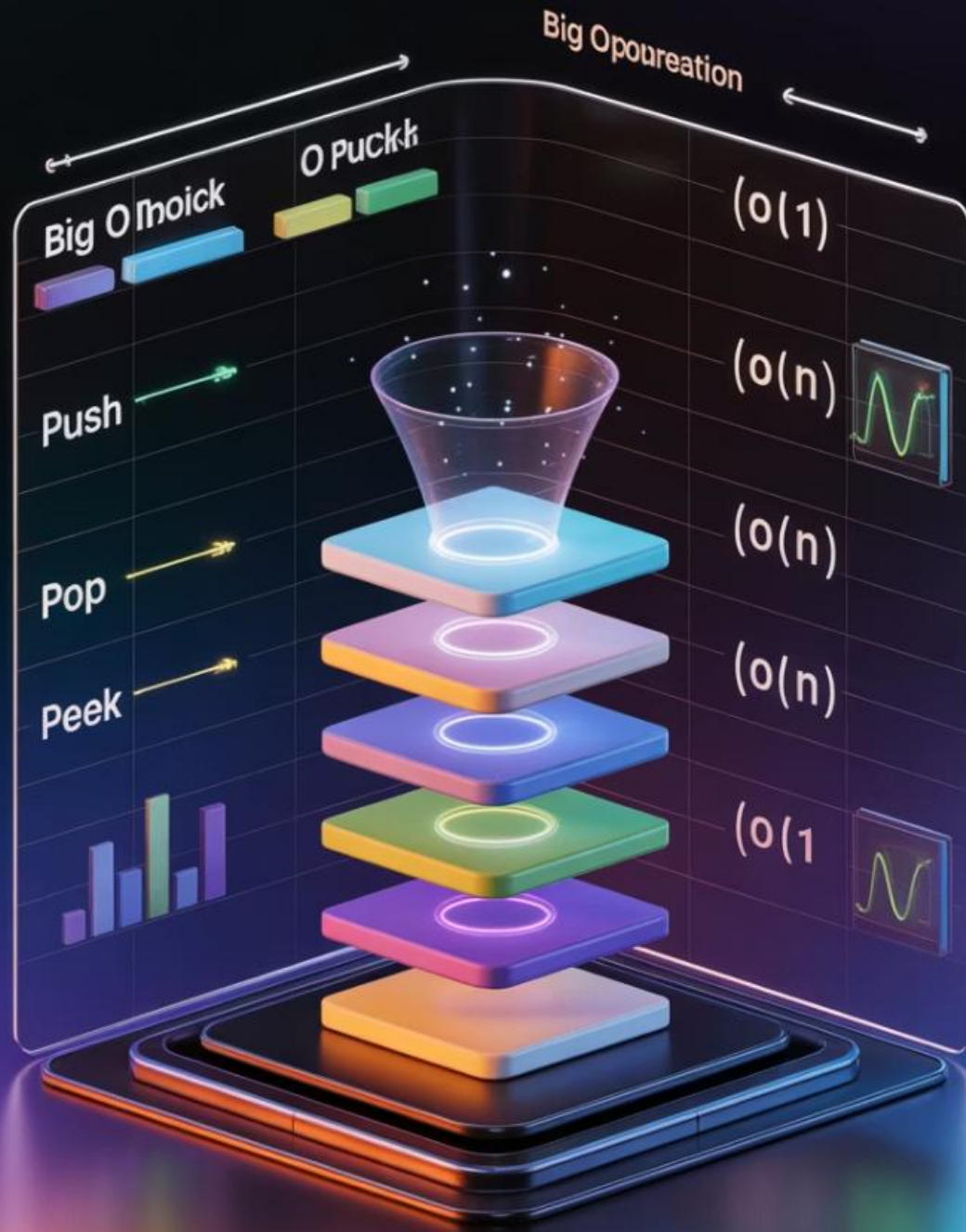
1. Leia cada símbolo da expressão infix
2. Use pilha para operadores
3. Aplique regras de precedência

Infix to postfix



Stack Operations

Time Complexity



Complexidade das Pilhas

$O(1)$

Push

Inserção no topo em tempo constante.

$O(1)$

Pop

Remoção do topo em tempo constante.

$O(1)$

Peek

Consulta do topo em tempo constante.

$O(n)$

Busca

Sem acesso direto a elementos que não estão no topo.

Filas: Queue



Princípio FIFO

First In, First Out - O primeiro a entrar é o primeiro a sair.



Pontos de Acesso

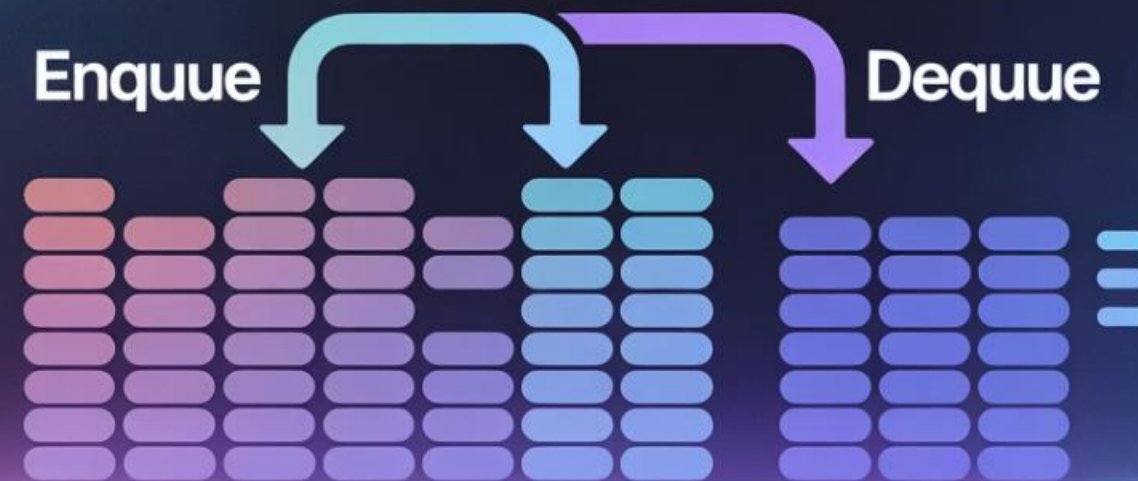
Inserção no final (rear) e remoção no início (front).



Operações

Enqueue (inserir), Dequeue (remover), Front (consultar início).

Queue



First-in, First-out (FIFO)

[Learn More](#)

Implementação de Filas com Arrays



Problema de Desalinhamento

Remoções criam espaços vazios no início.



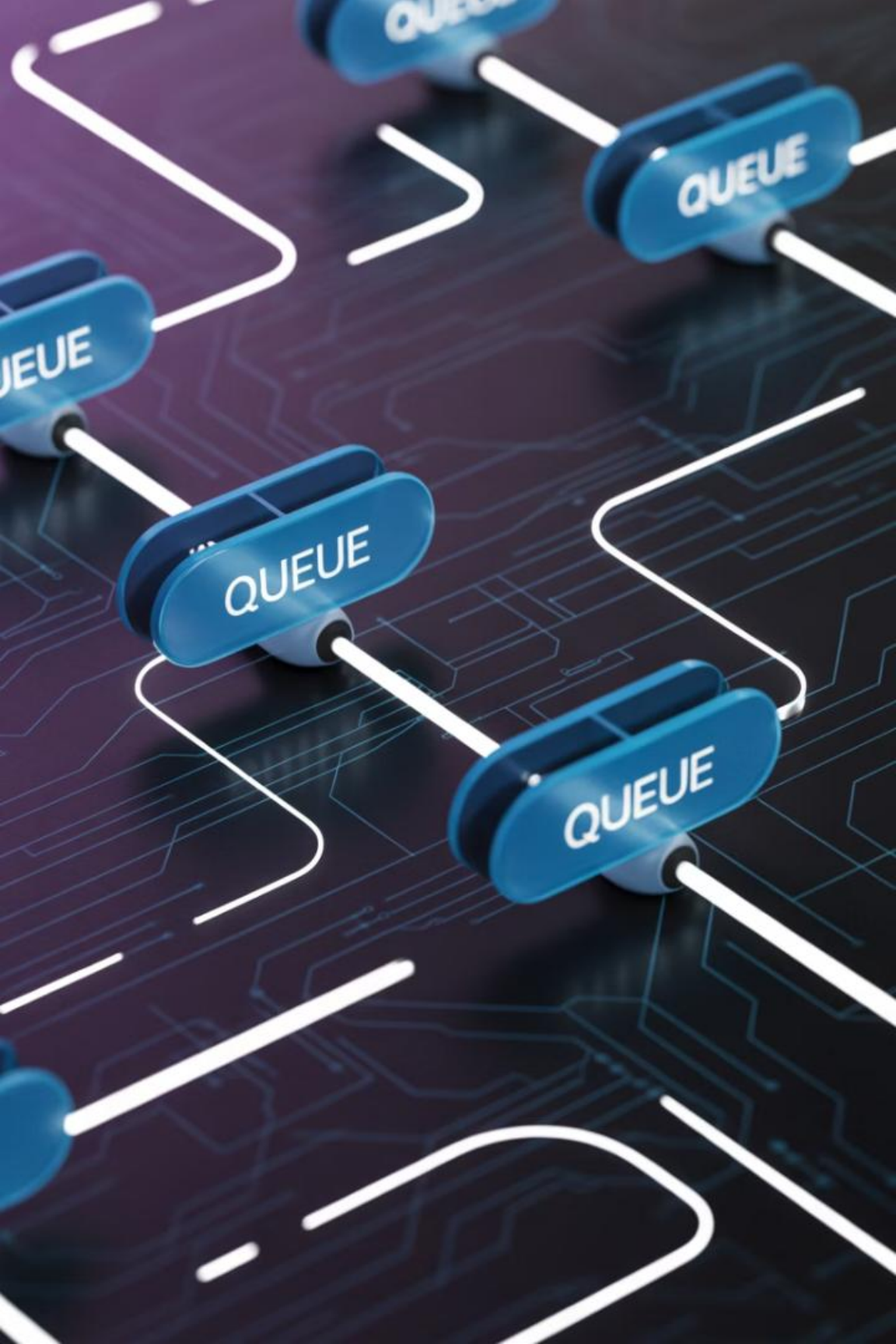
Solução: Fila Circular

Reaproveita espaços vazios com índices circulares.



Implementação

Usa aritmética modular para cálculo de índices.



Implementação de Filas com Listas Ligadas



Estrutura

Nós conectados com referências para o próximo elemento.



Ponteiros de Controle

Referências para início (front) e fim (rear) da fila.



Vantagem

Crescimento dinâmico sem necessidade de realocação.

Aplicações de Filas



Impressoras em Rede

Sistema de spooler processa documentos na ordem de chegada.



Call Centers

Atendimento sequencial de chamadas telefônicas.



Busca em Largura

Algoritmo BFS para exploração de grafos.

Fila Circular (Ring Buffer)

Conceito

Estrutura que reutiliza espaços vazios após remoções.

Aritmética Modular

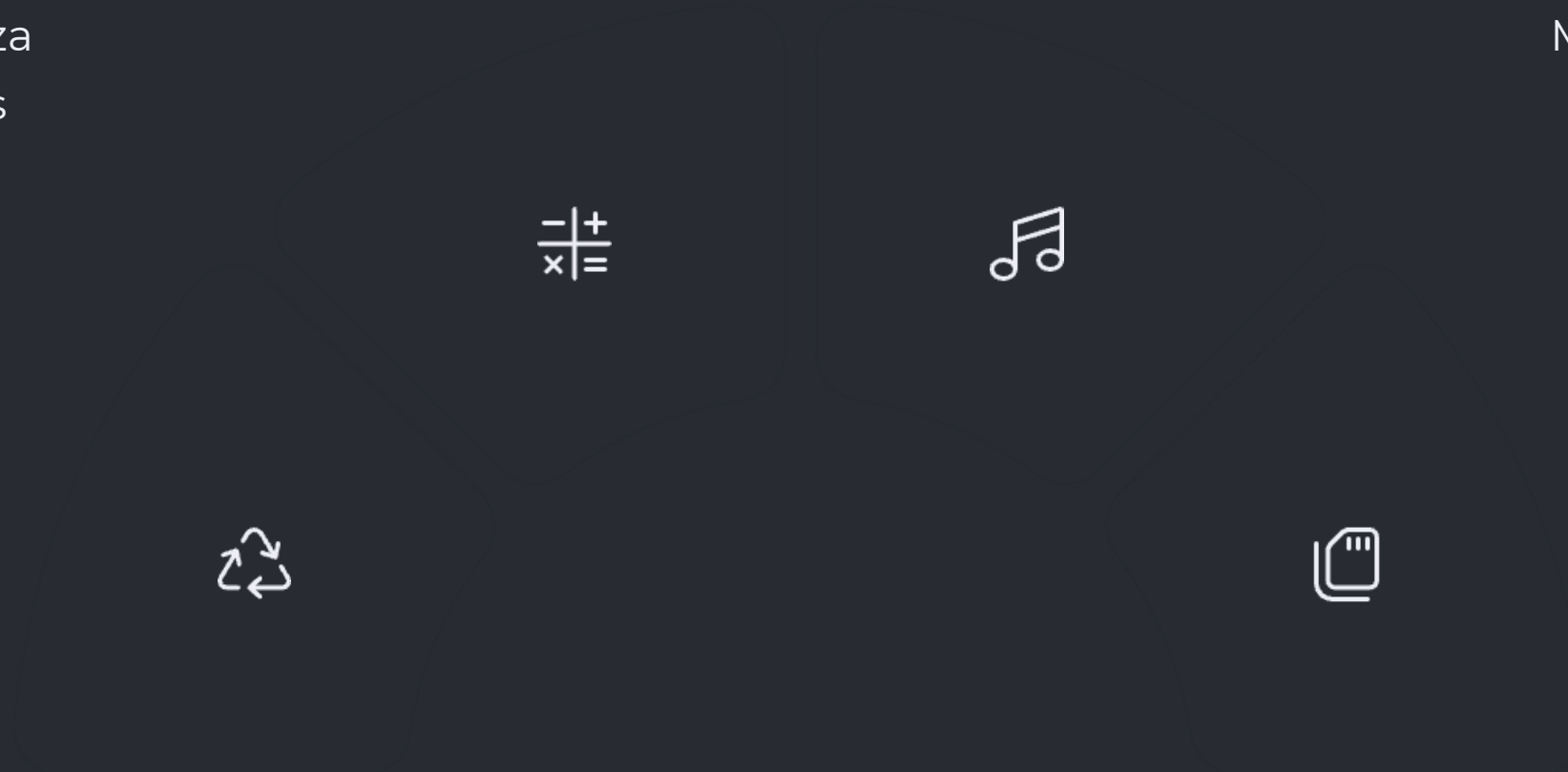
Índices calculados com resto da divisão pelo tamanho.

Aplicação

Muito usada em buffers de áudio e vídeo.

Eficiência

Melhor uso de memória sem realocações frequentes.



Algoritmo: Busca em Largura (BFS)



Início

Coloque o nó inicial na fila.



Exploração

Retire um nó da fila e visite todos seus vizinhos.



Expansão

Adicione os vizinhos não visitados à fila.

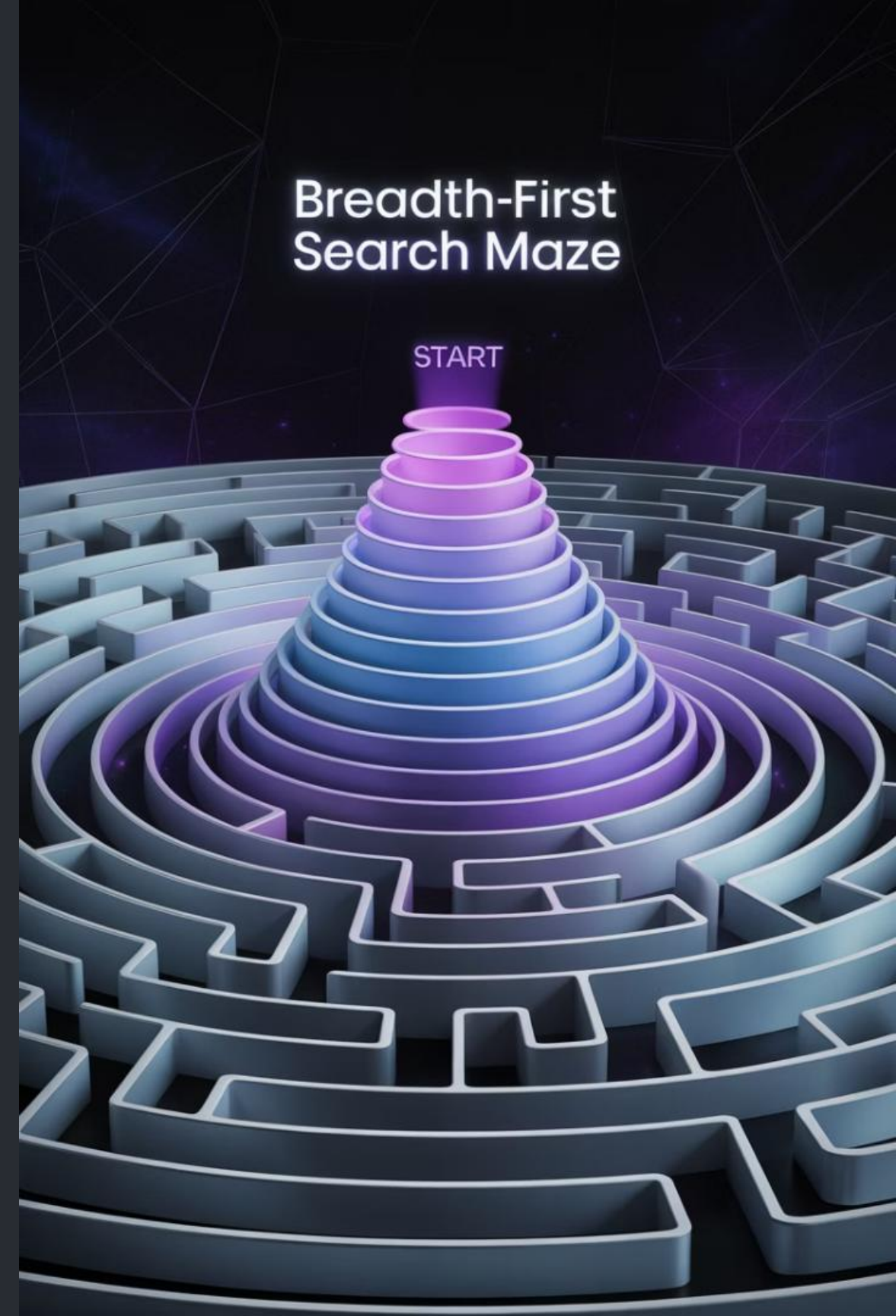


Repetição

Continue até que a fila fique vazia.

Breadth-First
Search Maze

START



Complexidade das Filas

Inserção (Enqueue)

$O(1)$ - Tempo constante quando implementada corretamente.

Remoção (Dequeue)

$O(1)$ - Tempo constante independente do tamanho da fila.

Acesso (Front)

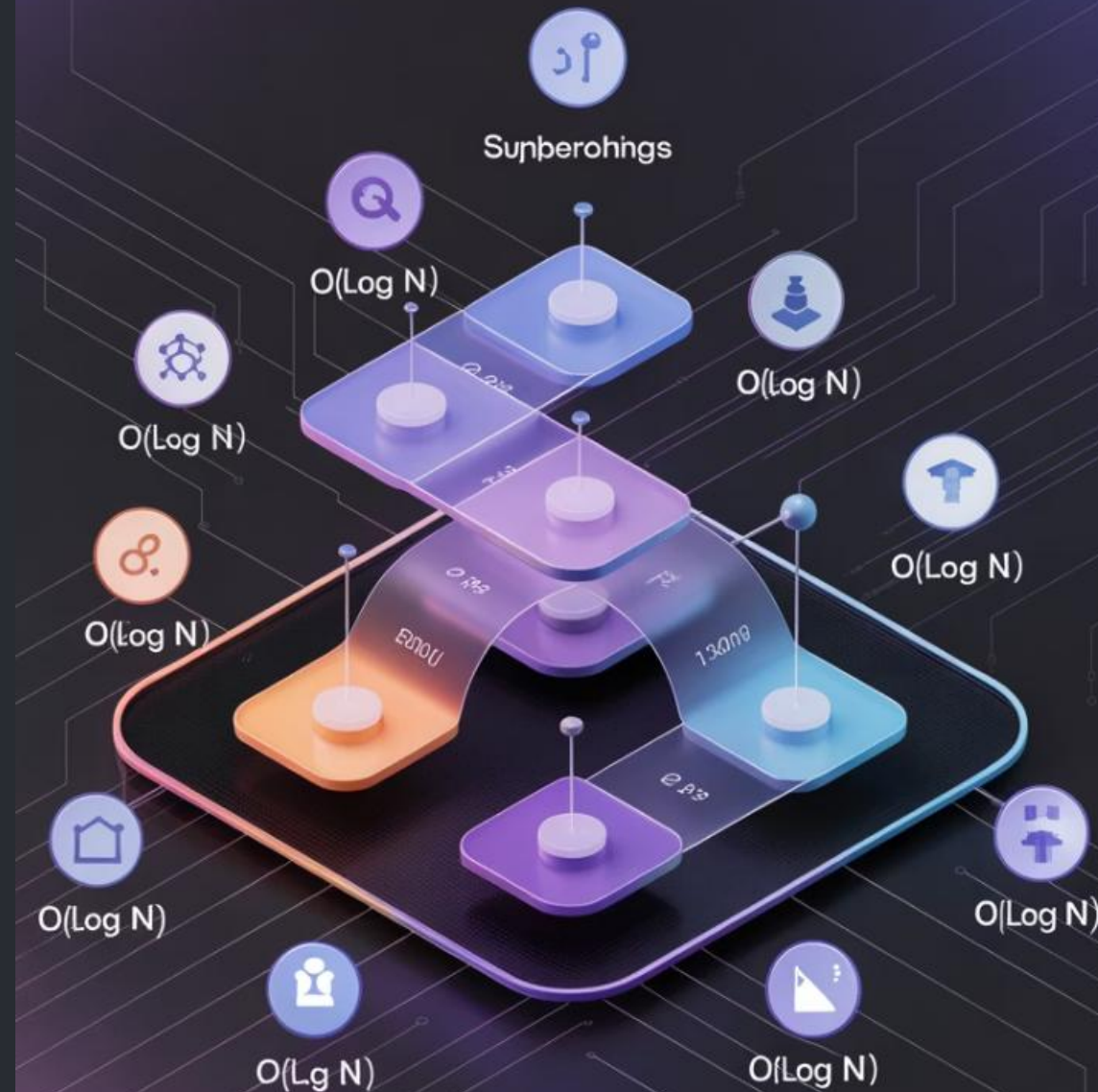
$O(1)$ - Acesso direto ao primeiro elemento.

Busca

$O(n)$ - Necessário percorrer todos os elementos no pior caso.

Queue Time Complexity

Big O Notation



Deques (Filas Duplas)

1

Flexibilidade Total

Combinação de pilha e fila.



Acesso Duplo

Operações em ambas as extremidades.



Operações Principais

push_front, push_back, pop_front, pop_back

Implementação de Deques

Arrays Circulares

Eficiente em memória e acesso, mas com tamanho limitado.

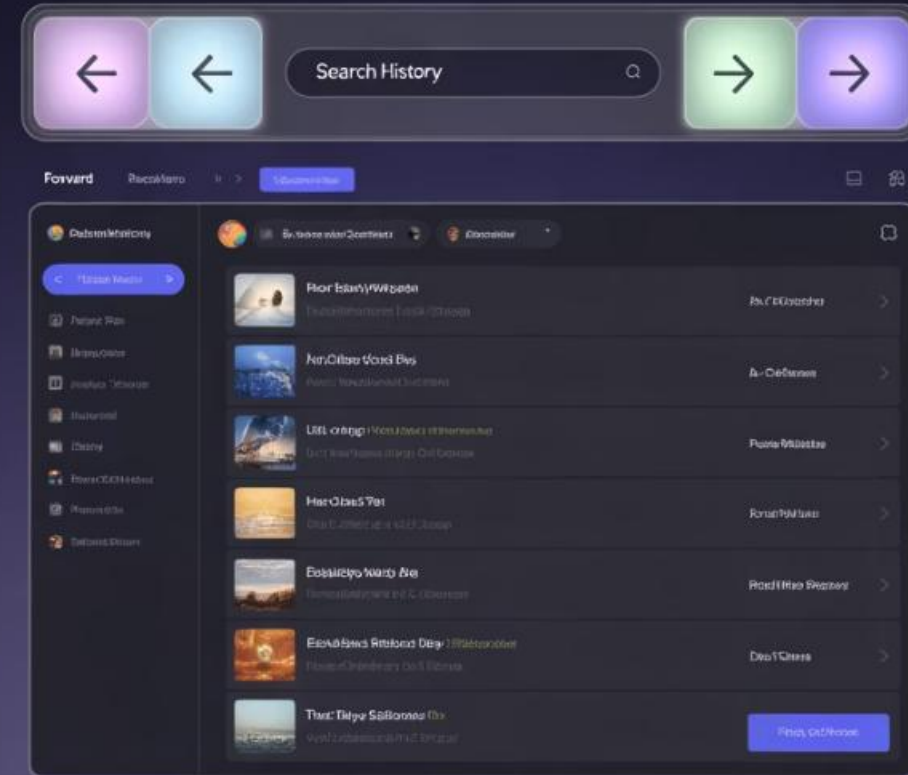
- Uso de aritmética modular
- Índices para frente e final
- Realocação quando cheio

Listas Duplamente Ligadas

Flexível em tamanho, mas com maior overhead de memória.

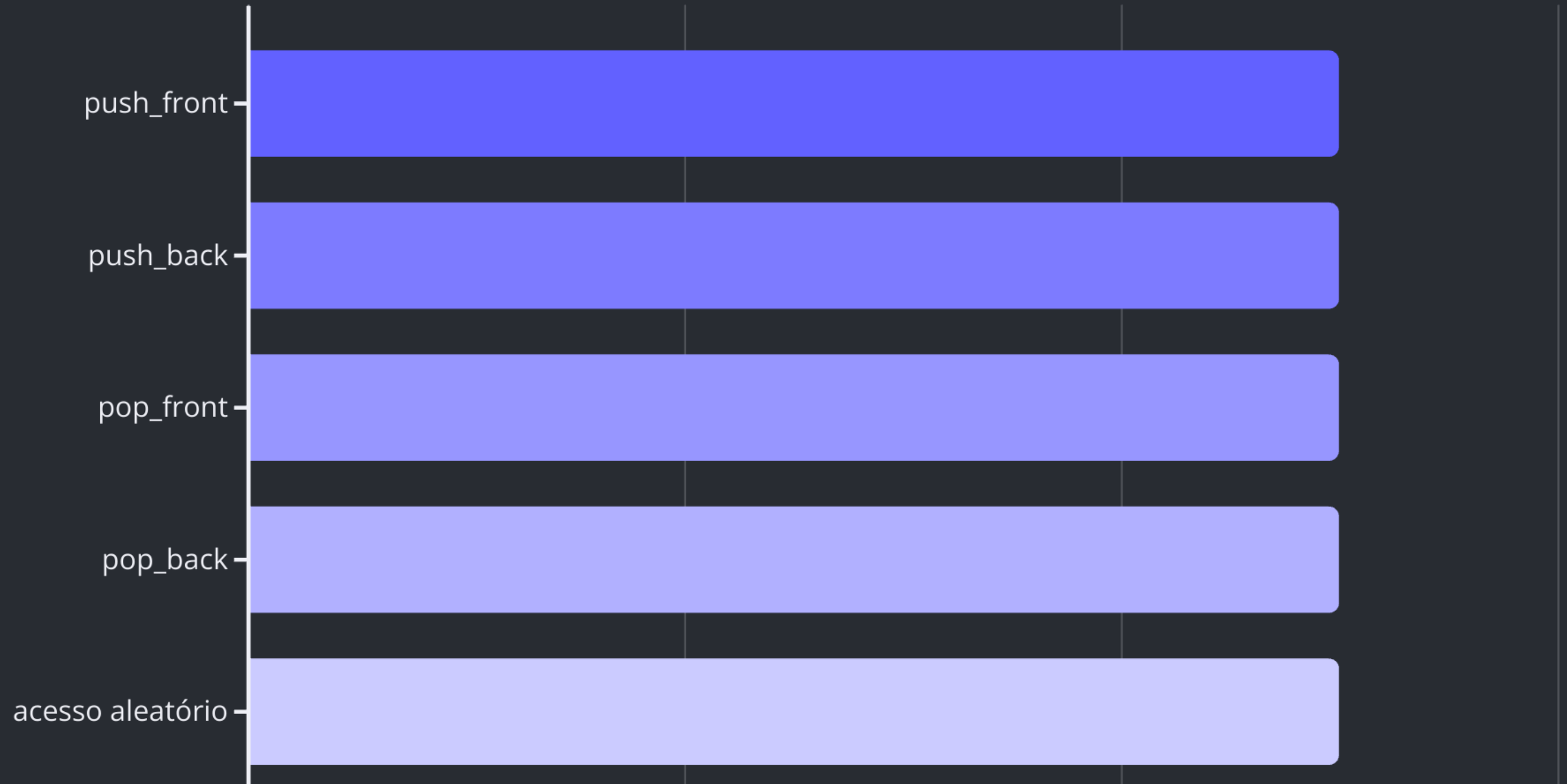
- Nós com ponteiros para anterior e próximo
- Crescimento dinâmico ilimitado
- Acesso $O(1)$ em ambas as extremidades

Aplicações de Deques



Palindrome Checking Algorithm

Complexidade dos Deques

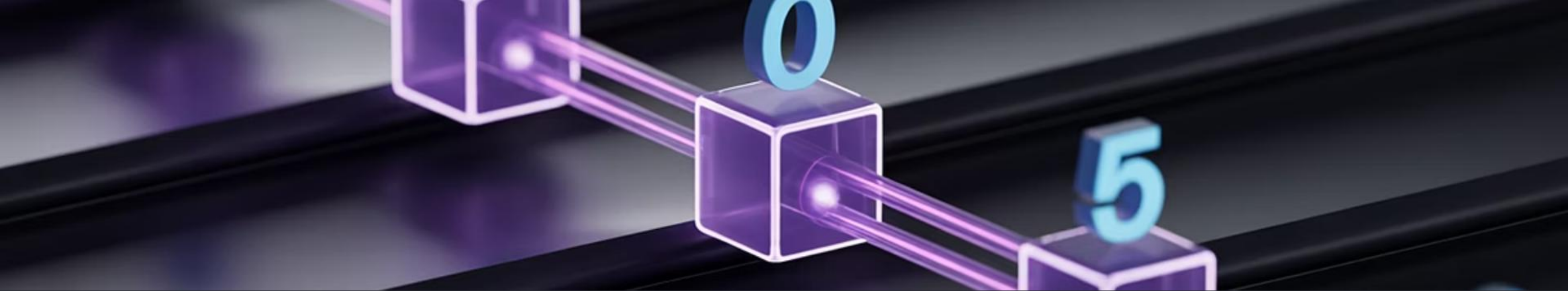




Stacks a Differences
Quaiques + Deques

Resumo Comparativo: Pilha vs. Fila vs. Deque

Estrutura	Princípio	Inserção	Remoção	Flexibilid ade
Pilha	LIFO	Topo	Topo	Baixa
Fila	FIFO	Final	Início	Média
Deque	Ambos	Ambas pontas	Ambas pontas	Alta



Listas Ligadas: Conceito

Estrutura Básica

Coleção de elementos chamados nós, cada um com dados e referência.

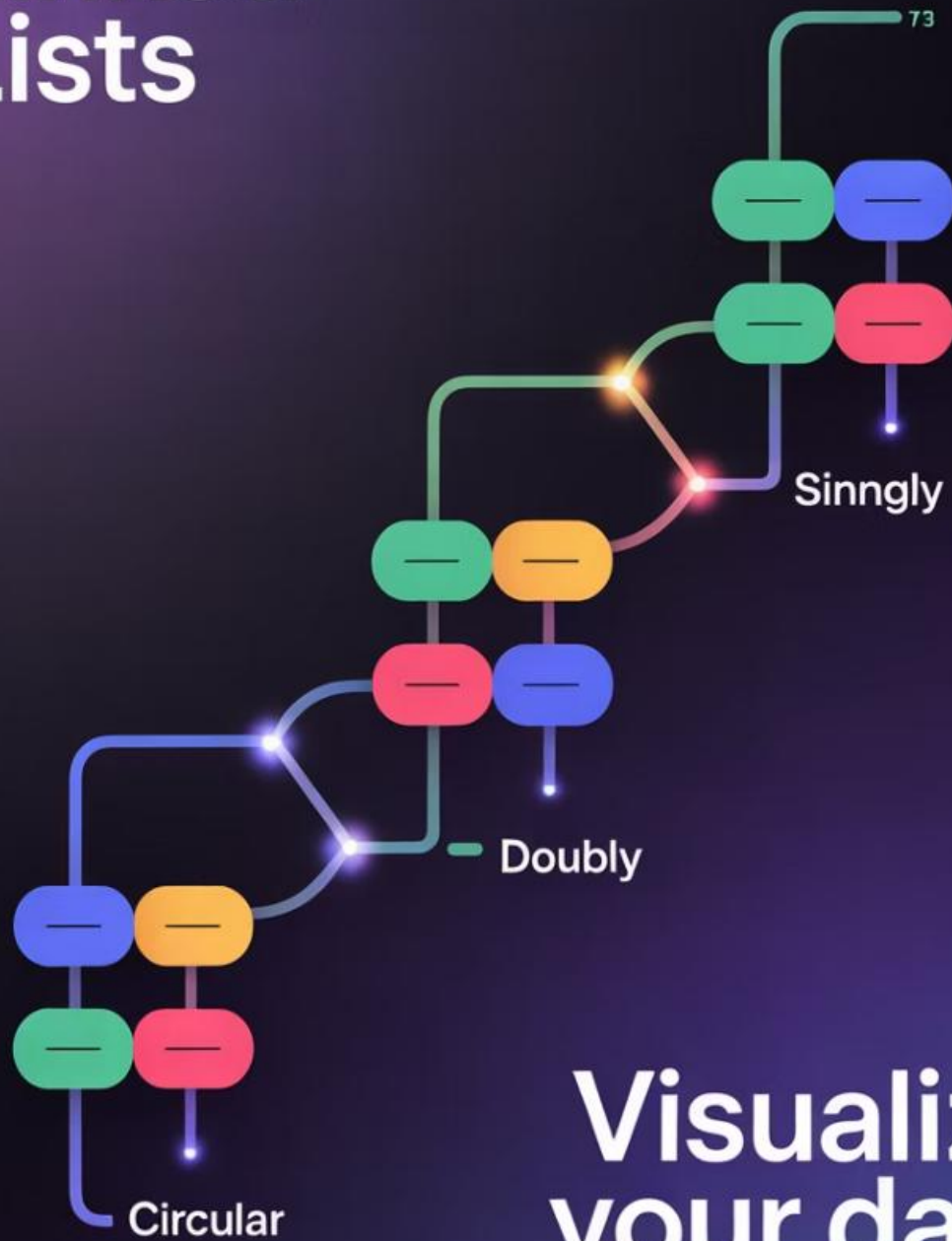
Conexão

Cada nó contém ponteiro ou referência para o próximo elemento.

Dinâmica

Estrutura flexível que cresce e diminui conforme necessário.

Linked Lists



Visualize
your data

Tipos de Listas Ligadas



Simplesmente Ligadas

Nós com referência apenas para o próximo elemento.



Duplamente Ligadas

Nós com referências para elementos anterior e próximo.

Circulares

Último elemento aponta para o primeiro, formando um círculo.

Lista Simplesmente Ligada

Estrutura do Nó

Cada nó contém dois componentes principais:

- Dado (valor armazenado)
- Ponteiro para o próximo nó

O último nó da lista aponta para NULL.

Vantagens

- Inserção eficiente no início: $O(1)$
- Remoção eficiente no início: $O(1)$
- Uso flexível de memória
- Sem realocação ao crescer

Lista Duplamente Ligada

Estrutura do Nó



Cada nó contém dado, ponteiro para o próximo e para o anterior.

Navegação Bidirecional

2

Possibilidade de percorrer a lista em ambas as direções.

Consumo de Memória

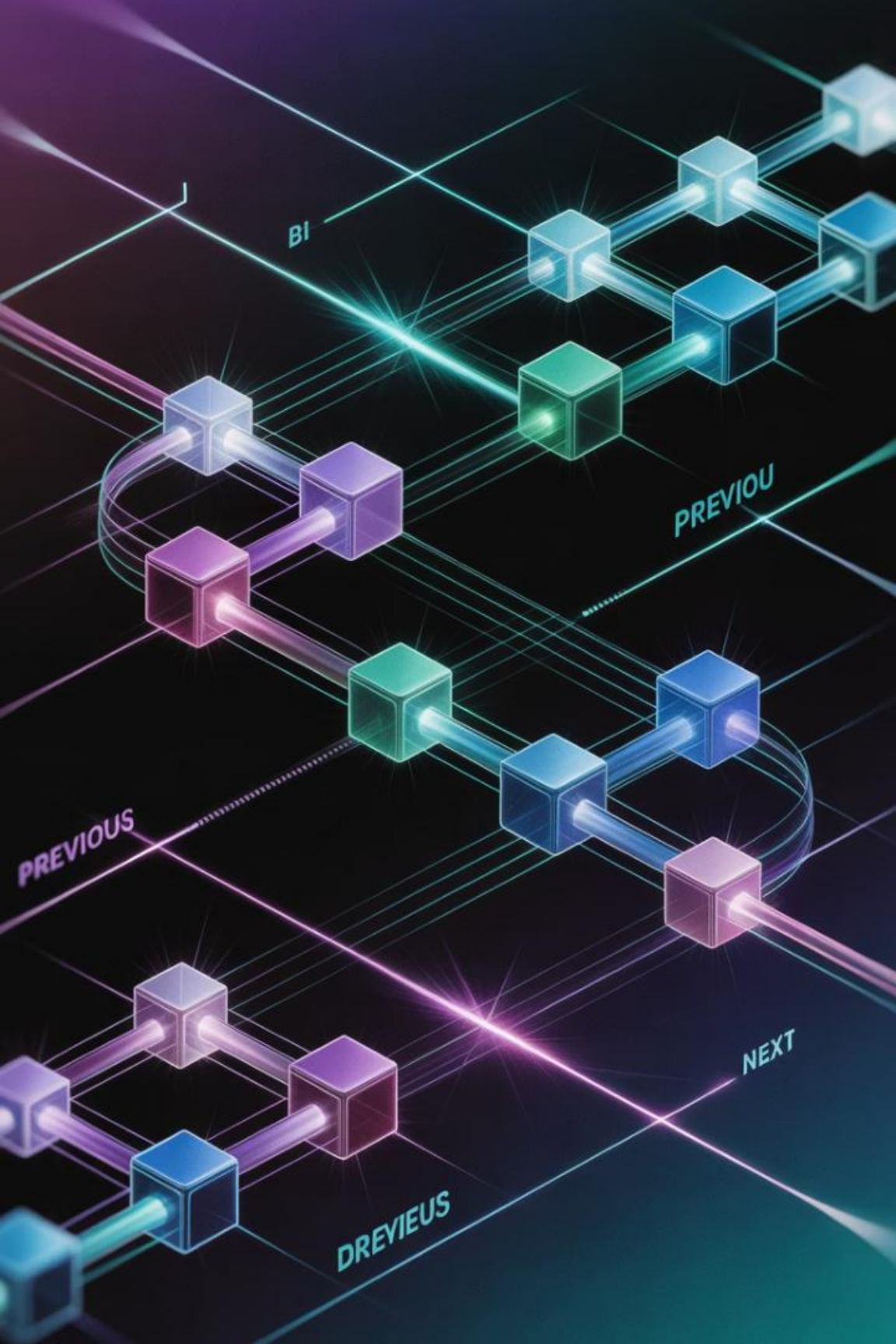


Maior overhead por nó devido ao ponteiro adicional.

Vantagem Principal



Remoção eficiente em qualquer posição com referência direta.



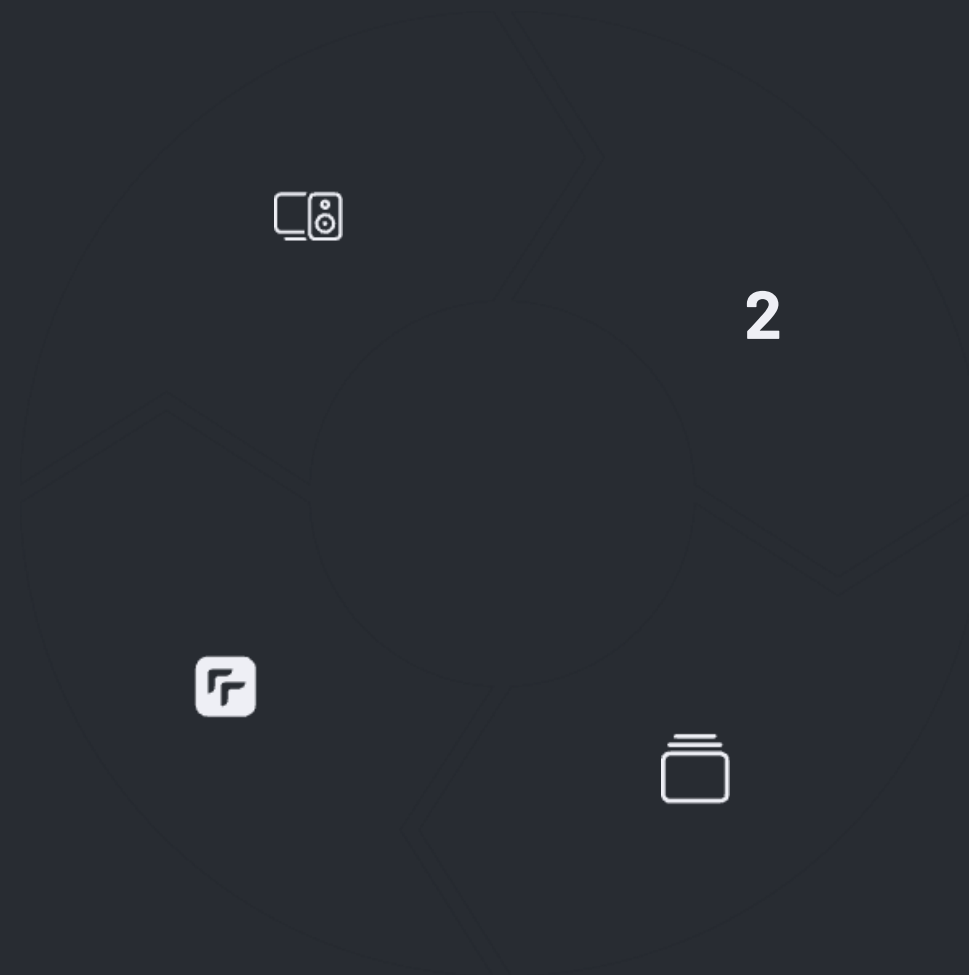
Lista Circular

Característica Principal

O último elemento aponta para o primeiro, sem NULL.

Percurso

Possibilidade de percorrer infinitamente sem parada.



Variantes

Pode ser simplesmente ou duplamente ligada.

Aplicações

Ideal para processos cíclicos como round-robin.

Inserção em Listas Ligadas

1

Inserção no Início

Operação $O(1)$, apenas ajusta a cabeça da lista.



Inserção no Meio

$O(n)$ para encontrar posição, $O(1)$ para inserir.

3

Inserção no Fim

$O(1)$ com ponteiro para cauda, $O(n)$ sem esse ponteiro.

Remoção em Listas Ligadas



Processo Básico

Atualização de ponteiros para contornar o nó removido.



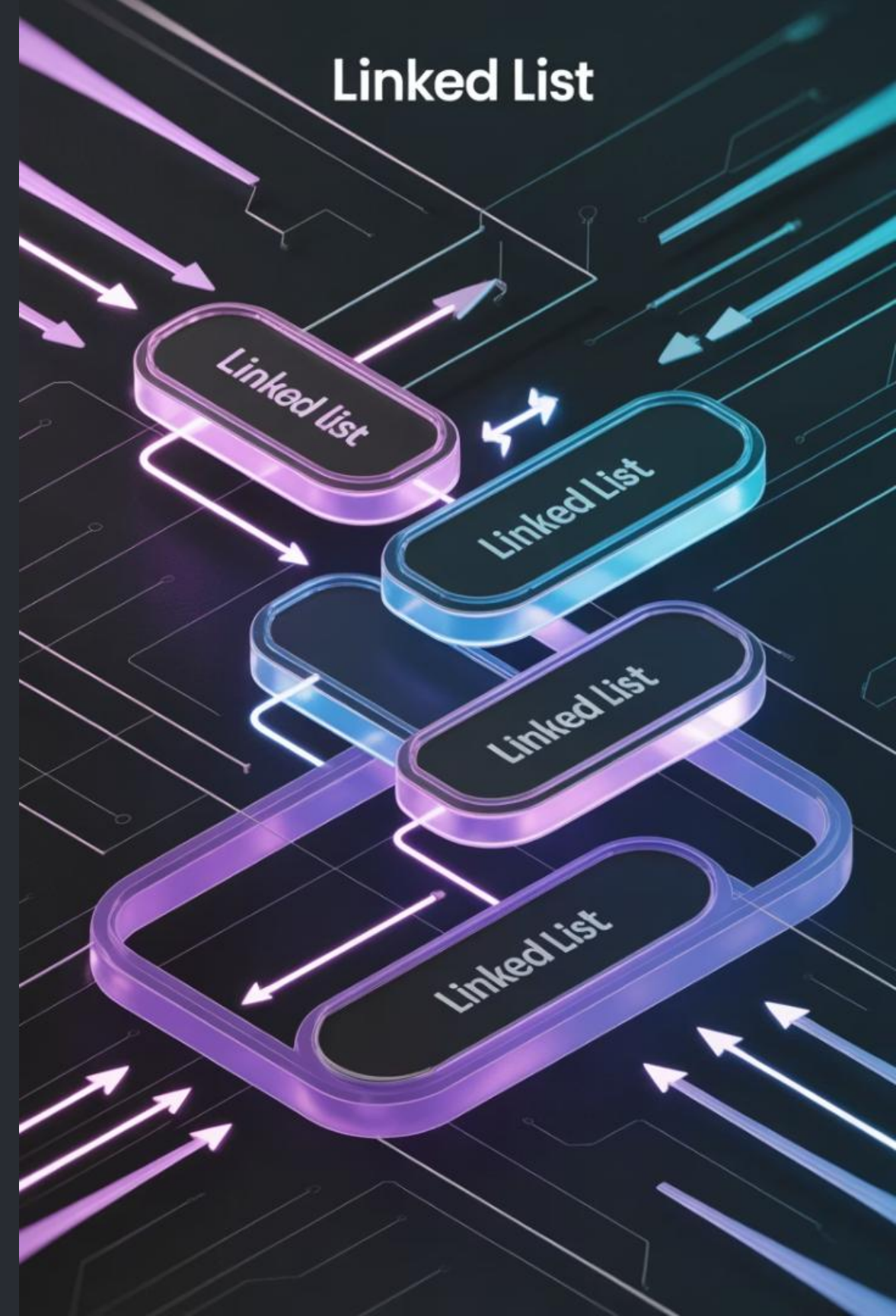
Casos Especiais

Remoção do primeiro ou último elemento requer tratamento específico.



Liberação de Memória

Necessário liberar memória do nó removido para evitar vazamentos.



Busca em Listas Ligadas



Processo

Percorre sequencialmente os nós, um por um.



Complexidade

$O(n)$ no pior caso, busca em todo o conjunto.



Acesso Aleatório

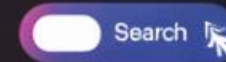
Não suporta acesso direto por índice.



Estratégias

Ordenação pode ajudar em alguns casos específicos.

Linked List Search Operation.



Vantagens de Listas Ligadas

Tamanho Dinâmico

Crescem e diminuem conforme necessário, sem realocação completa.

Alocação Flexível

Nós podem estar em qualquer lugar da memória, não precisam ser contíguos.

Inserção Eficiente

Inserir e remover elementos no início é extremamente rápido ($O(1)$).

Base para Outras Estruturas

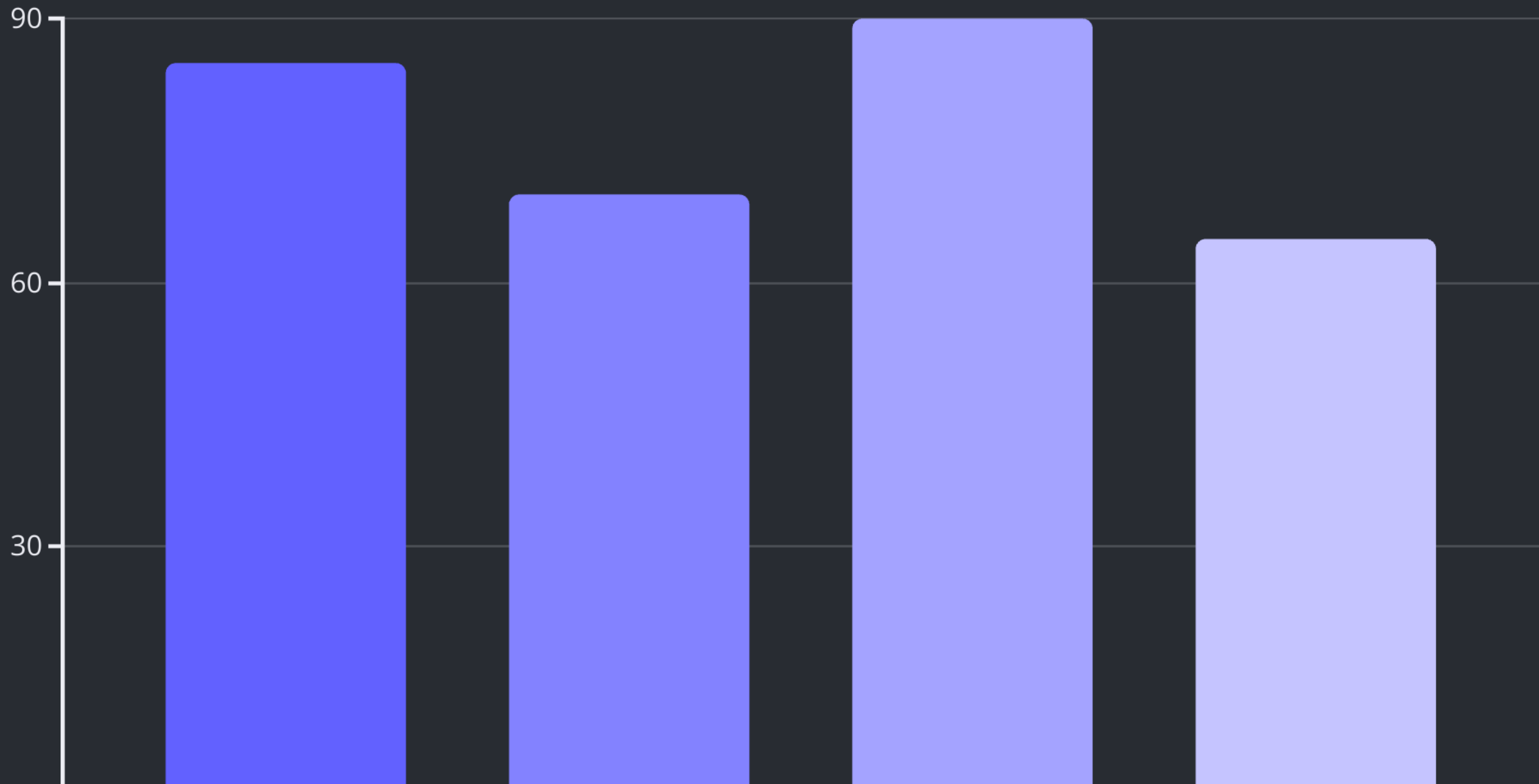
Servem como fundamento para implementar pilhas, filas e grafos.

Advantages

Linked List



Desvantagens de Listas Ligadas



Comparativo: Arrays vs. Listas Ligadas

Arrays

- Acesso direto $O(1)$
- Alocação contígua
- Tamanho fixo (arrays estáticos)
- Inserção/remoção no meio: $O(n)$
- Melhor uso de cache

Listas Ligadas

- Acesso sequencial $O(n)$
- Alocação dispersa
- Tamanho dinâmico
- Inserção/remoção no início: $O(1)$
- Mais overhead de memória

Aplicações de Listas Ligadas



Sistemas Avançados

Implementações de blockchain e distributed systems.



Estruturas Compostas

Base para grafos, árvores e tabelas hash.



Estruturas Fundamentais

Implementação de pilhas, filas e dequeues.

Algoritmo: Remoção de Duplicatas em Lista Ligada

1

Abordagem

Percorrer a lista mantendo conjunto de valores já encontrados.

#

Estrutura Auxiliar

Usar hash set para verificação rápida $O(1)$ de elementos repetidos.

✂

Remoção

Ajustar ponteiros para ignorar nós com valores duplicados.

🗑

Limpeza

Liberar memória dos nós removidos para evitar vazamentos.



Memory Management for Data Structures



Gerenciamento de Memória

Alocação Dinâmica

Solicitar memória do sistema operacional durante a execução.

- malloc/free em C
- new/delete em C++
- Automático em Python/Java

Desalocação

Liberar memória não mais necessária para evitar vazamentos.

- Manual em C/C++
- Automático em linguagens com GC

Coleta de Lixo

Processo automático de identificação e liberação de memória não utilizada.

- Implementado em Java, Python, JavaScript
- Reduz erros de gerenciamento

Erros Comuns em Estruturas Lineares



Segmentation Fault

Acesso a ponteiros nulos ou inválidos causando falha de memória.



Overflow/Underflow

Tentar inserir em estrutura cheia ou remover de estrutura vazia.



Memory Leak

Não liberar memória de nós removidos, causando vazamento.

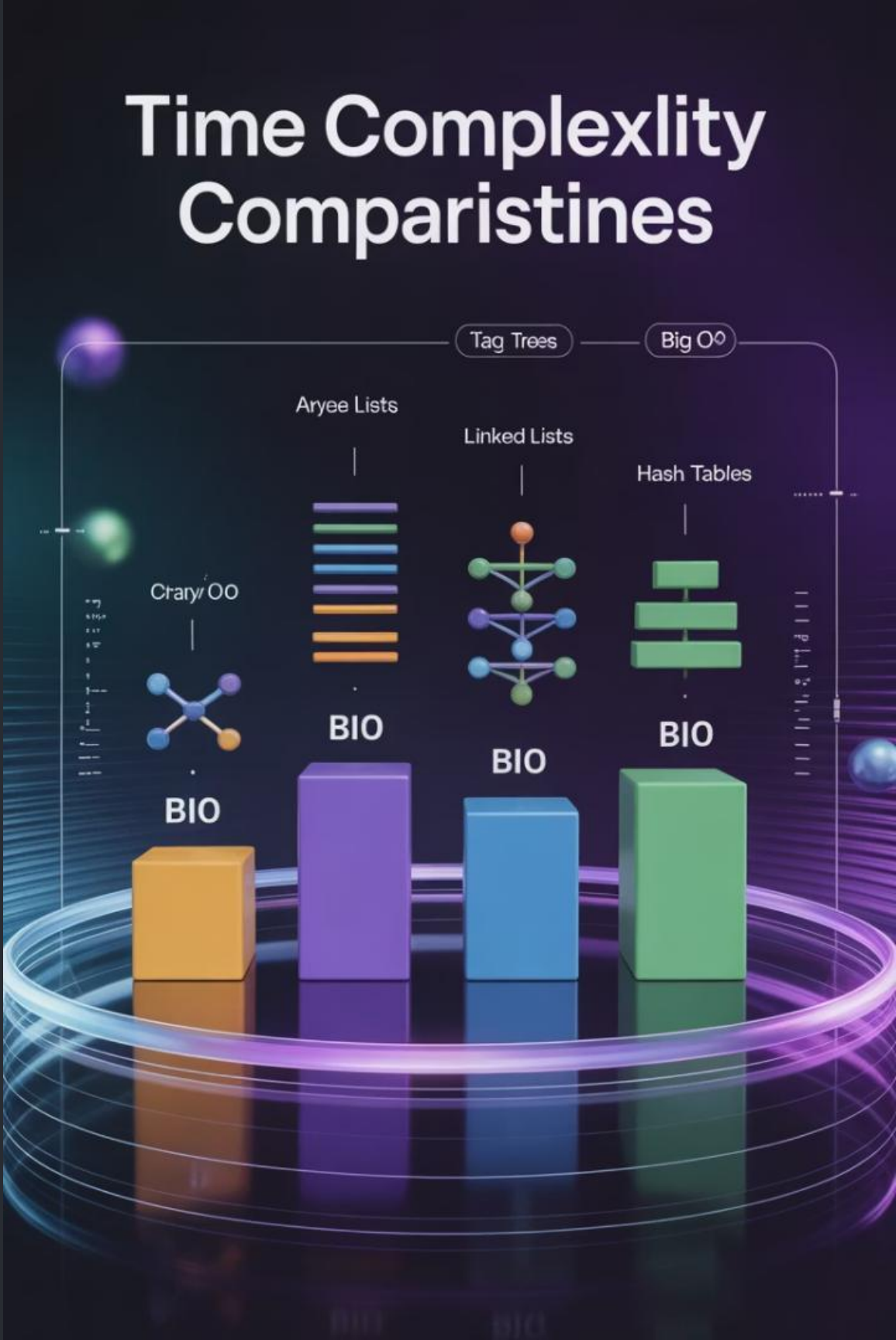
SEGMENTATION FAULT DATA STRUCTURE



Complexidade de Operações:

Tabela Resumo

Estrutura	Acesso	Busca	Inserção Início	Inserção Fim	Remoção Início	Remoção Fim
Array	$O(1)$	$O(n)$	$O(n)$	$O(1)^*$	$O(n)$	$O(1)$
Lista Ligada	$O(n)$	$O(n)$	$O(1)$	$O(n)^{**}$	$O(1)$	$O(n)^{**}$
Pilha	$O(n)$	$O(n)$	$O(1)$	N/A	$O(1)$	N/A
Fila	$O(n)$	$O(n)$	N/A	$O(1)$	$O(1)$	N/A



Estruturas Híbridas e Variantes

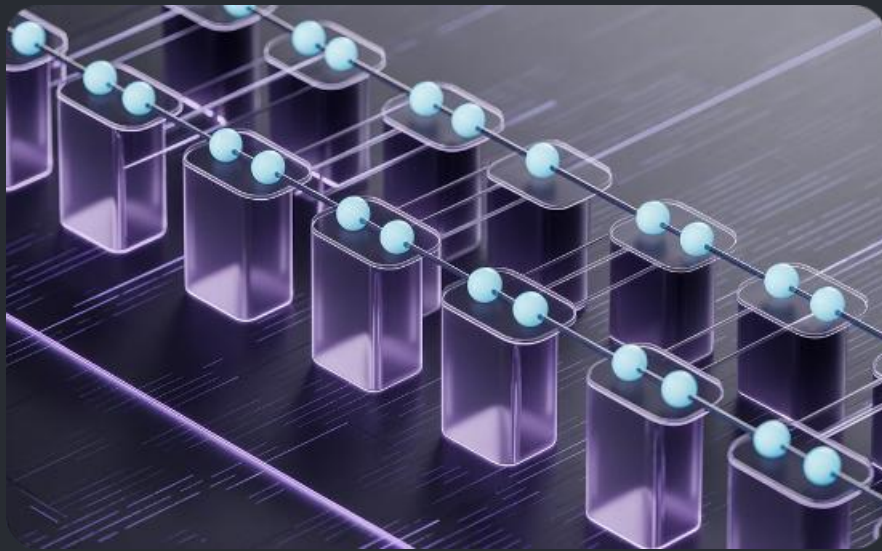
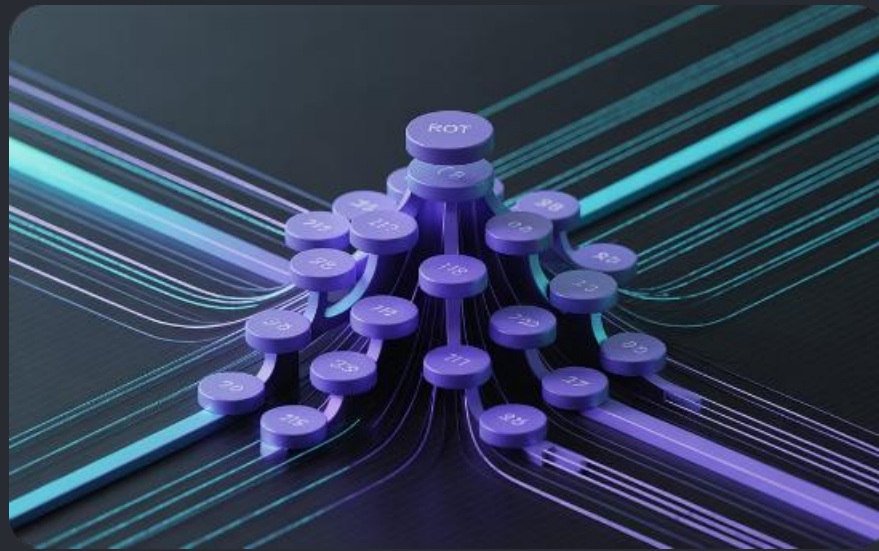


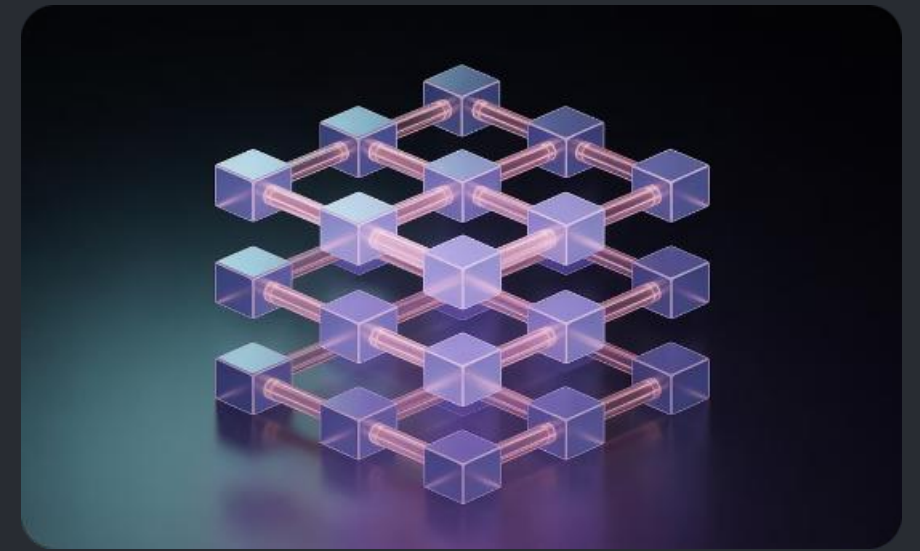
Tabela Hash

Vetor de listas ligadas para resolução de colisões.



Fila de Prioridade (Heap)

Combinação de propriedades de árvore e array.

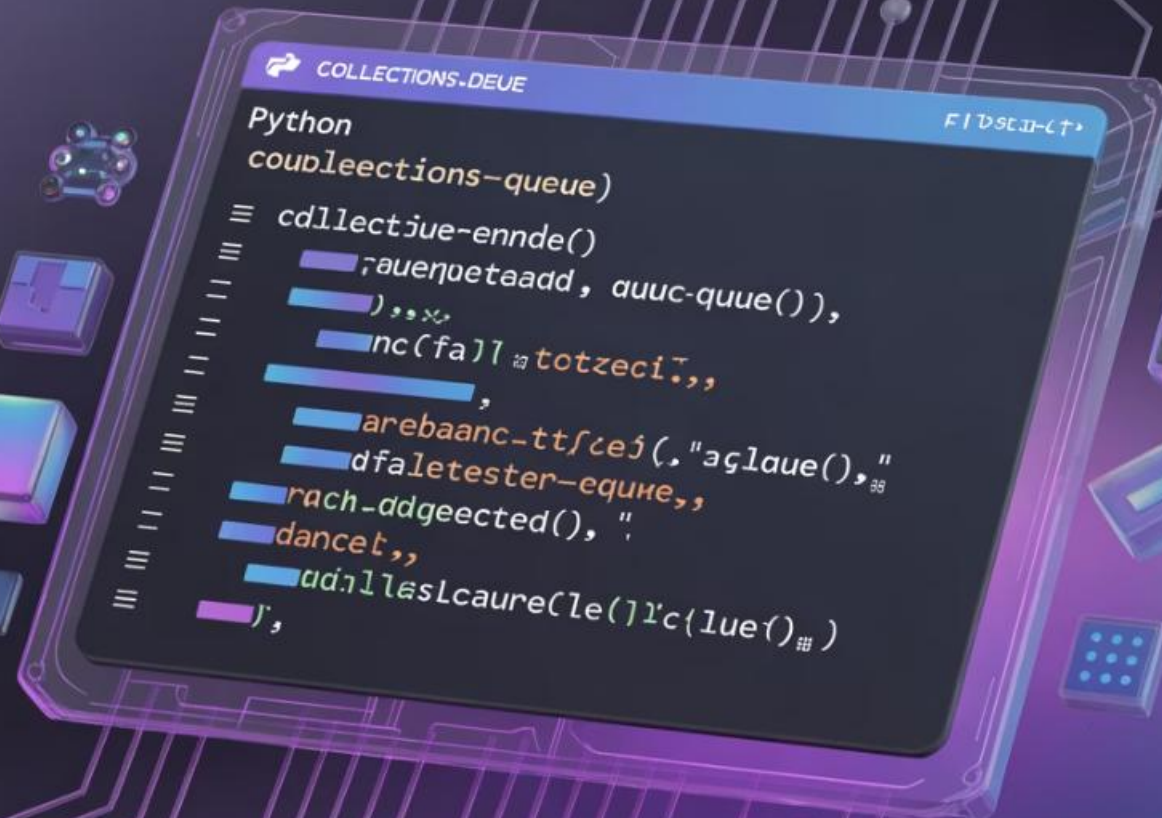


Skip List

Múltiplas listas ligadas em camadas para busca eficiente.


```
=== LIT,  
=== "FAPPAung-tine=)  
==== opcing,-lieratinn)  
KICARDI. ==
```

Exemplos em Python





Exemplos em C

0

Índice Inicial

Arrays em C são indexados a partir de zero.

*

Ponteiros

Usados para implementar estruturas dinâmicas.

sizeof()

Alocação

Gerenciamento manual de memória com malloc/free.

Exemplos em Java

ArrayList

Implementação dinâmica de array com redimensionamento automático.

Queue

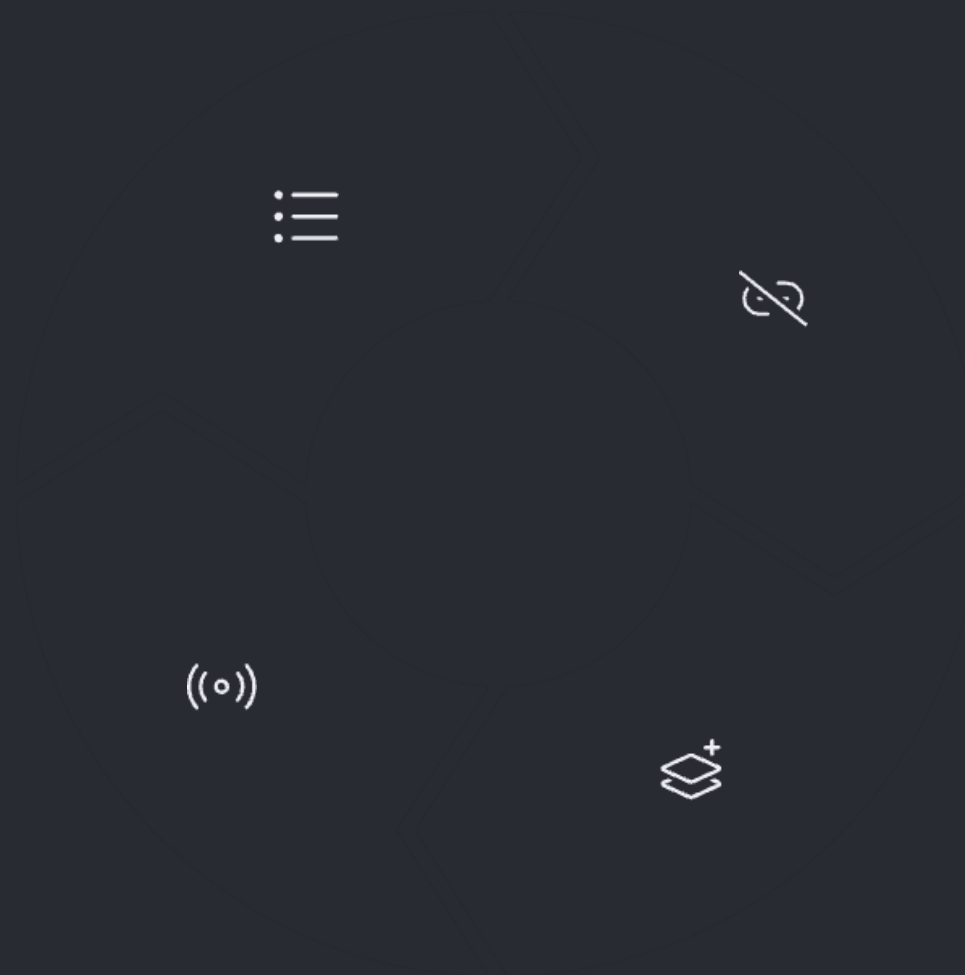
Interface para filas com implementações como LinkedList e PriorityQueue.

LinkedList

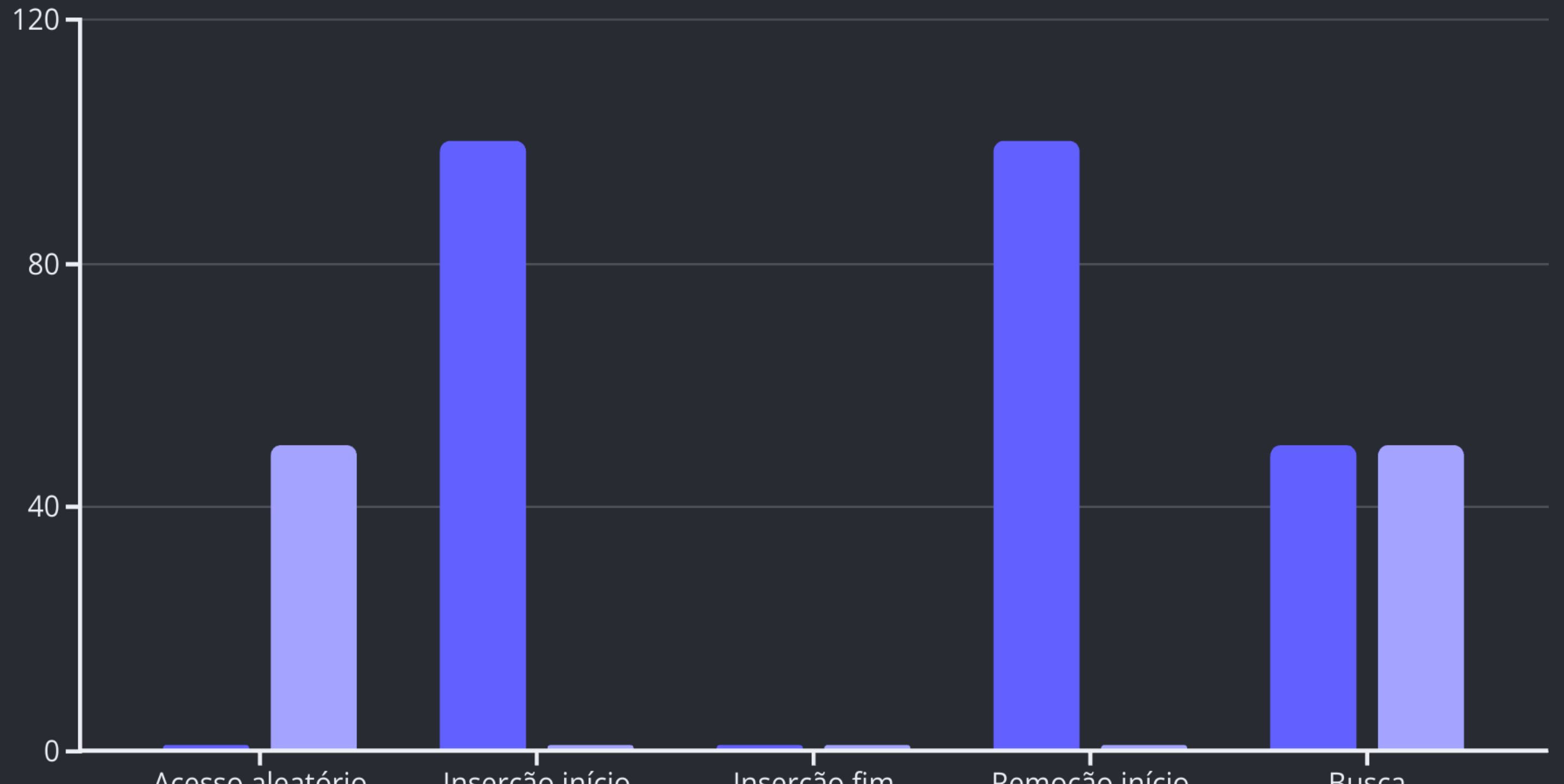
Lista duplamente ligada que implementa as interfaces List e Deque.

Stack

Implementação da estrutura de pilha (classe legada).



Análise de Performance



DATA STRUCTURE SELECTION GUIDE



Escolha da Estrutura Correta

Necessidade	Estrutura Recomendada	Motivo
Acesso rápido por índice	Array	$O(1)$ para acesso direto
Inserções/remoções frequentes no início	Lista Ligada	$O(1)$ para manipulação no início
Processamento LIFO	Pilha	Otimizada para operações no topo
Processamento FIFO	Fila	Otimizada para enfileirar/desenfileirar
Flexibilidade máxima nas extremidades	Deque	Operações $O(1)$ em ambas as pontas

Data Structure Optimization



Otimizações Possíveis



Pré-alocação

Reservar memória antecipadamente quando tamanho final é previsível.



Redução de Ponteiros

Usar XOR para armazenar dois ponteiros em um único campo.



Agrupamento

Alocar múltiplos nós em blocos contíguos para melhor uso de cache.



Estruturas Especializadas

Usar implementações híbridas otimizadas para casos específicos.

Casos Reais e Estudo de Caso



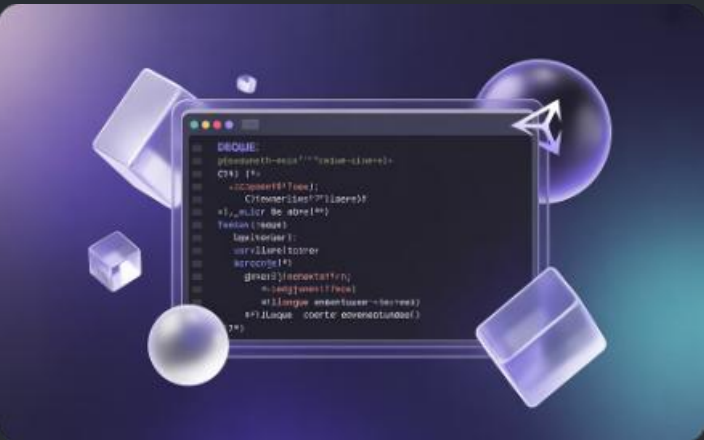
Navegadores Web

Chrome usa pilha de navegação para gerenciar histórico de páginas visitadas.



Servidores de Email

Gmail implementa filas de prioridade para processamento de mensagens.



Desenvolvimento de Jogos

Unity utiliza deque para gerenciar eventos e ações em tempo real.