

Estruturas de Dados e Algoritmos

Fundamentos essenciais da ciência da computação que impactam diretamente o desenvolvimento de software eficiente.

Nesta apresentação, exploraremos estruturas fundamentais, analisaremos seu desempenho e discutiremos implementações práticas.



por Mayana Duarte



A dark background featuring a complex, glowing network graph composed of numerous small, semi-transparent blue and purple dots connected by thin white lines, creating a sense of data flow and connectivity.

Introdução às Estruturas de Dados

Definição

Formas organizadas de armazenar dados para uso eficiente em computadores.

Design de Software

São fundamentais para criar aplicações bem estruturadas e eficientes.

Desempenho

Influenciam diretamente a velocidade e consumo de recursos de um programa.

Classificação de Estruturas de Dados

Estruturas Lineares

Arrays, listas, pilhas e filas

Elementos organizados em sequência

Estruturas Dinâmicas

Crescem e diminuem durante a execução



Estruturas Não Lineares

Árvores e grafos

Elementos com múltiplas conexões

Estruturas Estáticas

Tamanho fixo definido na compilação

Análise de Complexidade Computacional

Notação Big O

Descreve o comportamento limitante de uma função. Representa o pior caso de um algoritmo.

Exemplos comuns: $O(1)$, $O(\log n)$, $O(n)$, $O(n \log n)$, $O(n^2)$

Tempo de Execução

Mede quantas operações são necessárias para completar um algoritmo.

Crucial para escolher estruturas adequadas para cada aplicação.

Consumo de Memória

Avalia quanto espaço é necessário durante a execução.

Importante para sistemas com recursos limitados.



Fundamentos de Algoritmos

Estratégias de Resolução

Divide e conquista, programação dinâmica, algoritmos gulosos.

Tipos de Algoritmos

Ordenação, busca, recursão, grafos, probabilísticos.

Critérios de Eficiência

Tempo de execução, uso de memória, precisão, estabilidade.

Arrays



Conceito Básico

Coleção de elementos do mesmo tipo, armazenados em posições contíguas de memória.



Vantagens e Limitações

Acesso rápido indexado. Tamanho fixo em muitas linguagens.



Implementação

Sintaxe varia entre linguagens. Em Java: `int[] array = new int[10];`

Operações com Arrays



Inserção

Adiciona elementos em posições específicas



Remoção

Elimina elementos e reorganiza o array

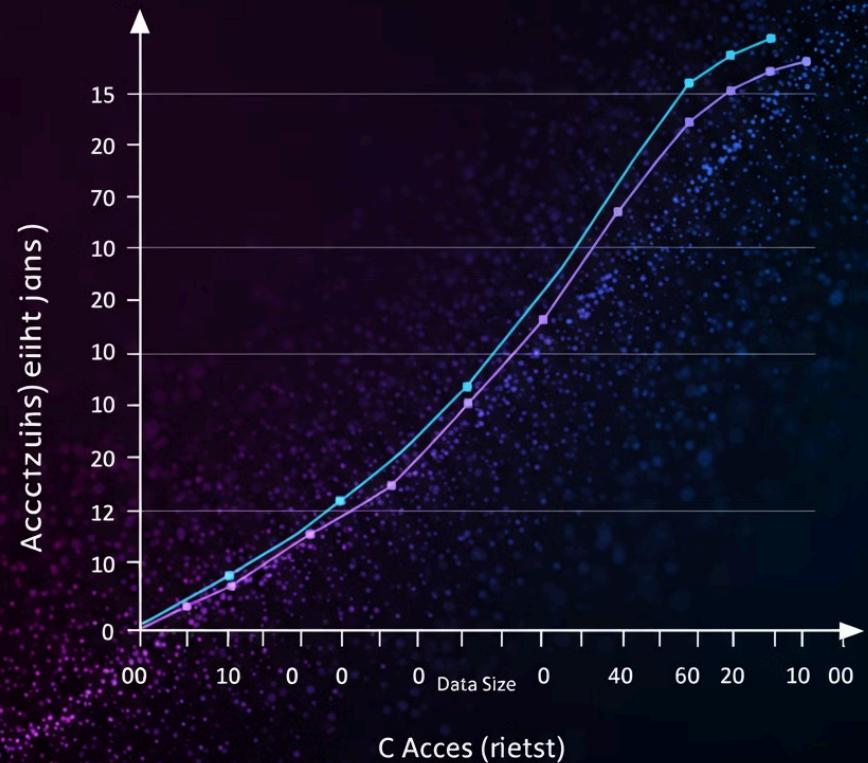


Busca

Localiza elementos por valor ou índice

Análise de Desempenho de Arrays

Operação	Complexidade de Tempo	Observações
Acesso	$O(1)$	Constante, independente do tamanho
Busca	$O(n)$	Linear no pior caso
Inserção/Remoção	$O(n)$	Exige deslocamento de elementos



Listas Encadeadas

Estrutura Dinâmica

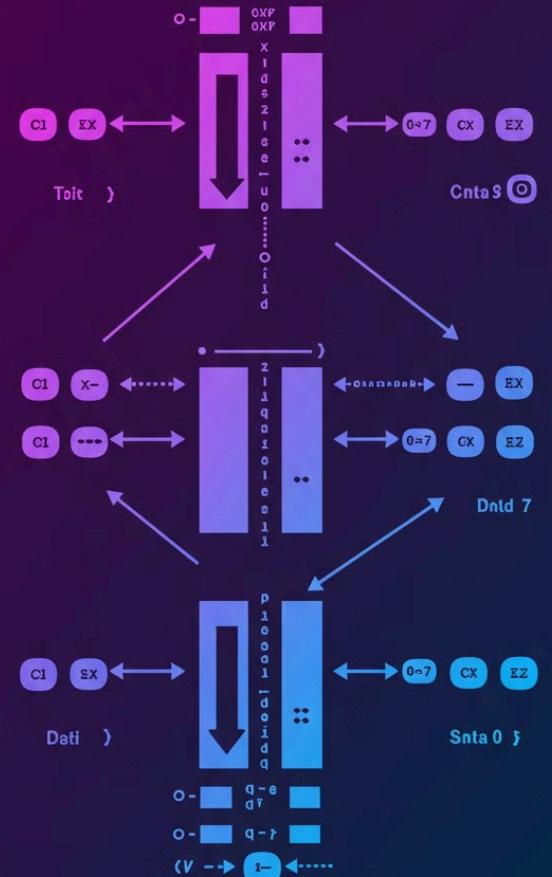
Elementos não contíguos, conectados por referências.

Tipos

Simples, duplas, circulares, com cabeça, ordenadas.

Implementação

Cada nó contém dado e referência ao próximo nó.



Operações em Listas Encadeadas

Inserção

Adiciona novo nó, atualizando referências

Busca

Localiza elementos por valor, verificando cada nó

Remoção

Elimina nó e reconecta os adjacentes

Navegação

Percorre a lista seguindo referências



Listas Duplamente Encadeadas

Estrutura Avançada

Cada nó contém referências para elementos anterior e posterior.

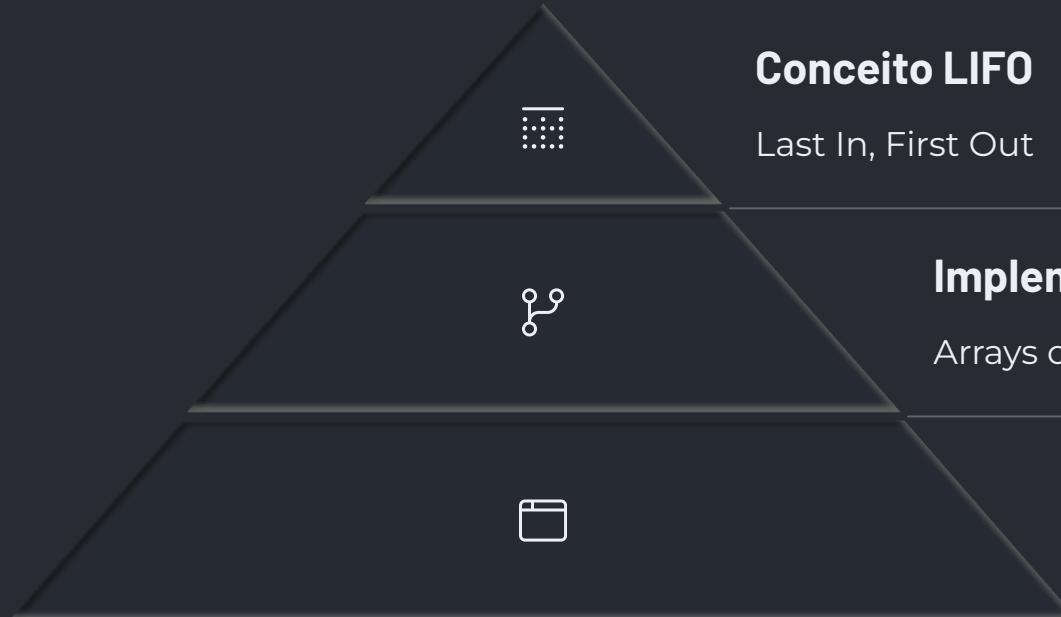
Permite navegação bidirecional através da lista.

Vantagens

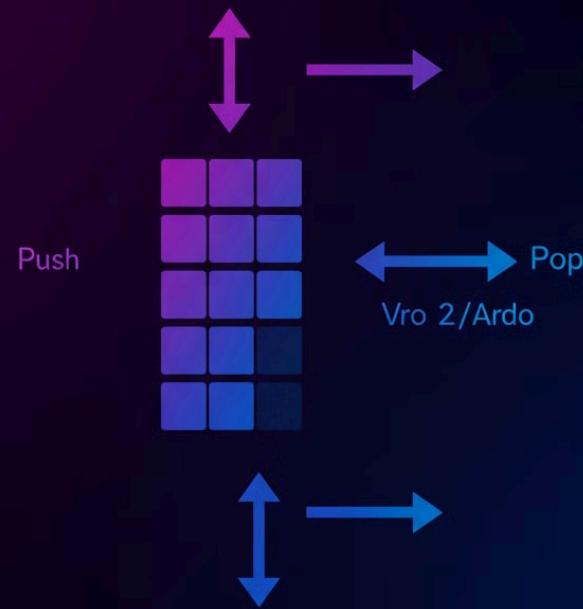
- Navegação em ambas direções
- Remoção mais eficiente
- Flexibilidade em operações



Pilhas (Stacks)



Stack Data structure



Vop Sep on atacc!

Operações em Pilhas



Push

Adiciona elemento ao topo da pilha.

Pop

Remove elemento do topo da pilha.

Peek

Verifica o elemento do topo sem removê-lo.

Filas (Queues)

Conceito FIFO

First In, First Out - primeiro elemento adicionado é o primeiro a sair.

Similar a uma fila do mundo real.

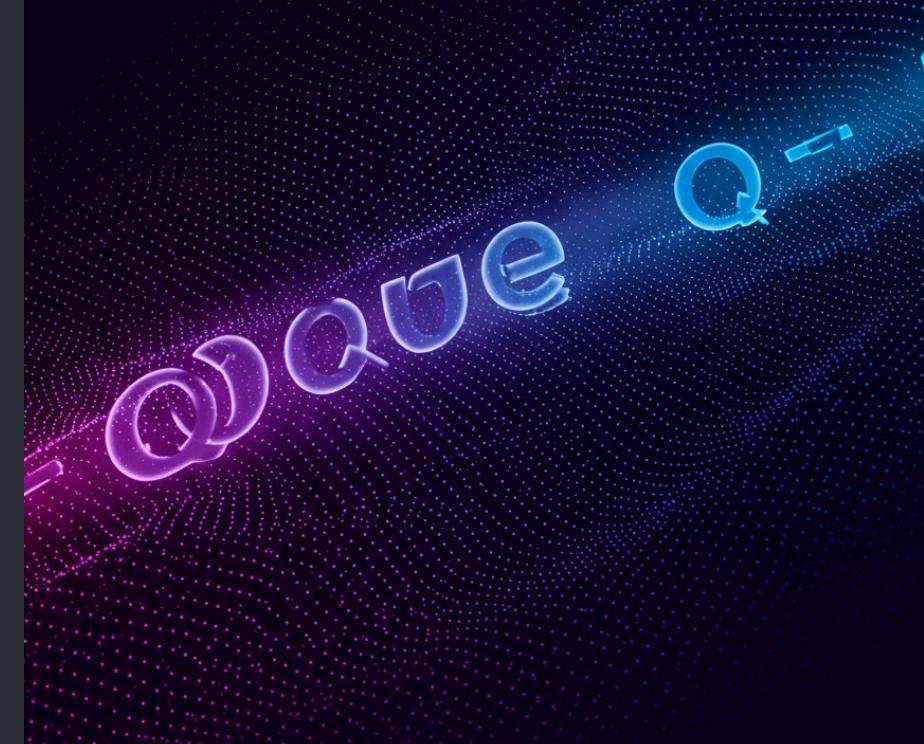
Tipos de Filas

- Simples
- Circulares
- De prioridade
- De dupla extremidade (deque)

Implementações

Arrays, listas encadeadas ou combinações.

Cada método tem vantagens específicas.



Operações em Filas

Enqueue

Adiciona elemento ao final da fila.

Complexidade $O(1)$ em implementações eficientes.

Dequeue

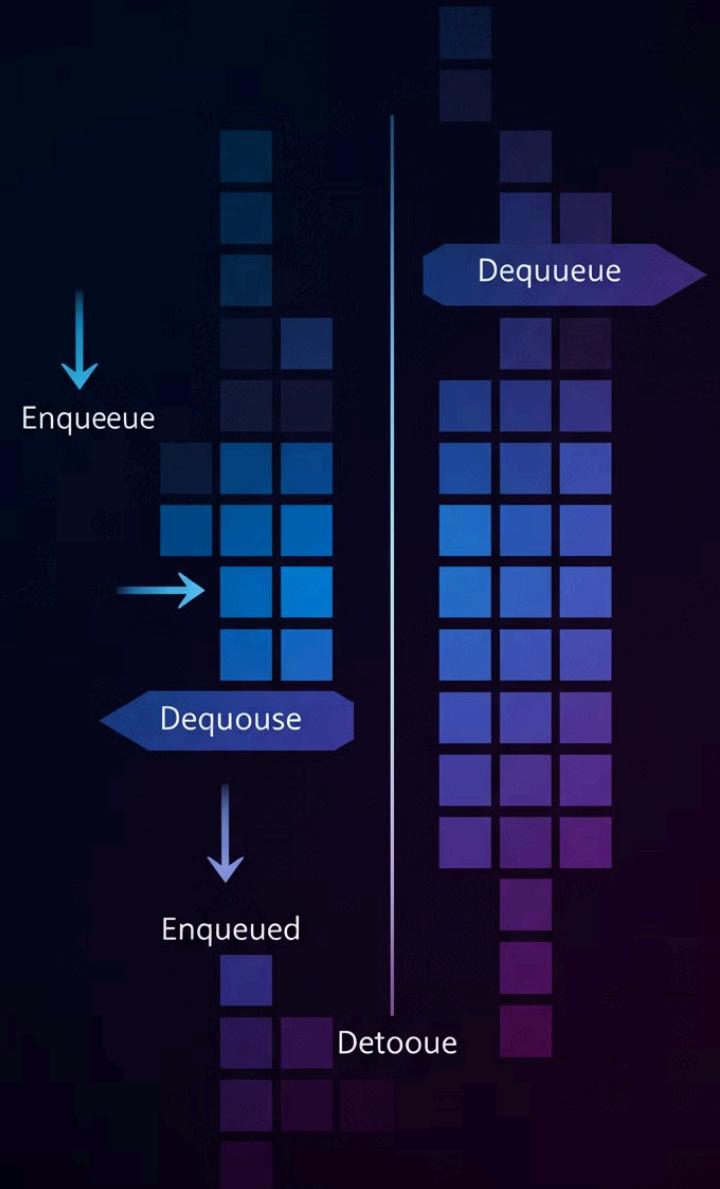
Remove elemento do início da fila.

Mantém a ordem de chegada intacta.

Peek/Front

Verifica o primeiro elemento sem removê-lo.

Útil para tomada de decisões baseadas no próximo item.



Árvores Binárias

Estrutura Hierárquica

Cada nó tem no máximo dois filhos:
esquerdo e direito.

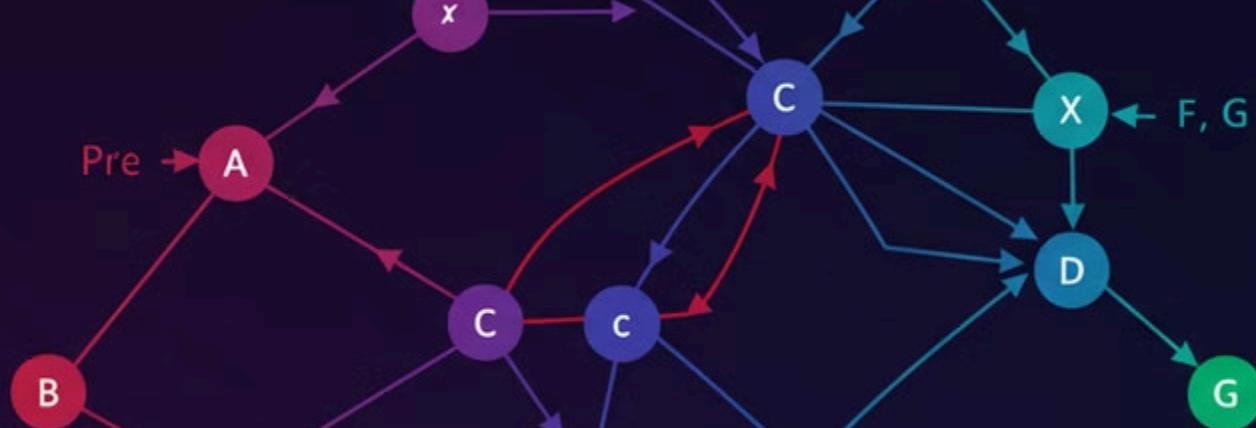
Organização que simula hierarquias
naturais.

Tipos de Árvores

- Completas
- Cheias
- Perfeitas
- Balanceadas
- De busca

Conceitos Fundamentais

- Raiz
- Nós internos
- Folhas
- Altura e profundidade



Travessia de Árvores



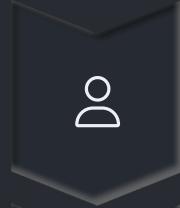
Pré-ordem

Visita raiz, subárvore esquerda, subárvore direita.



Em-ordem

Visita subárvore esquerda, raiz, subárvore direita.



Pós-ordem

Visita subárvore esquerda, subárvore direita, raiz.



Por nível

Visita nós por níveis, da raiz às folhas.



Árvores de Busca Binária



Propriedades

Nós à esquerda são menores que o pai. Nós à direita são maiores.



Remoção

Requer reorganização para manter propriedades da árvore.



Inserção

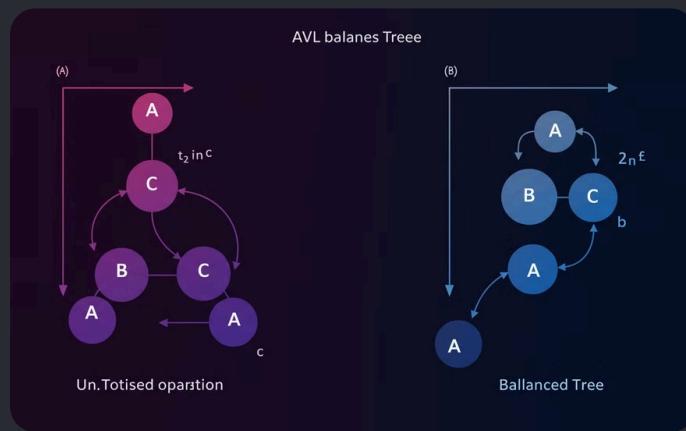
Localiza posição adequada seguindo regras de ordenação.



Busca

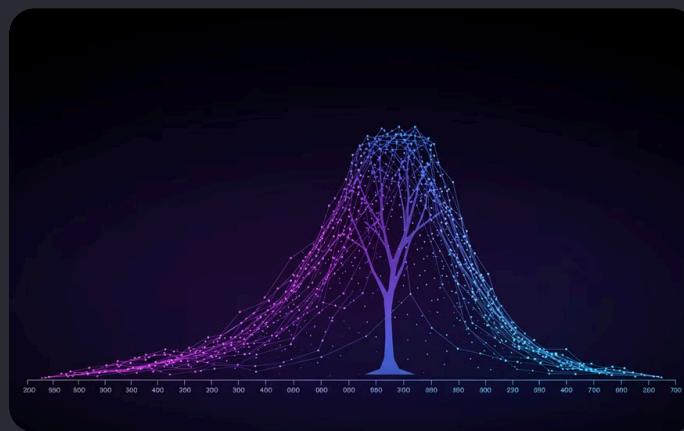
$O(\log n)$ em árvores平衡adas, $O(n)$ no pior caso.

Árvores AVL



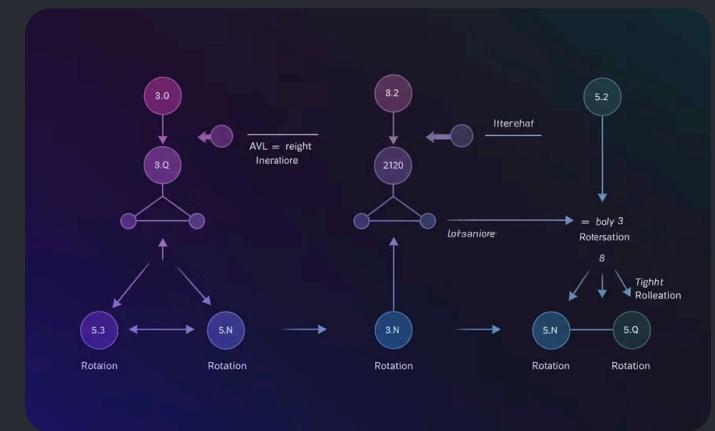
Rotações para Balanceamento

Operações que redistribuem nós para manter a altura equilibrada.



Fatores de Balanceamento

Diferença entre alturas das subárvore esquerda e direita.



Auto-balanceamento

Após cada inserção ou remoção, a árvore se reequilibra.

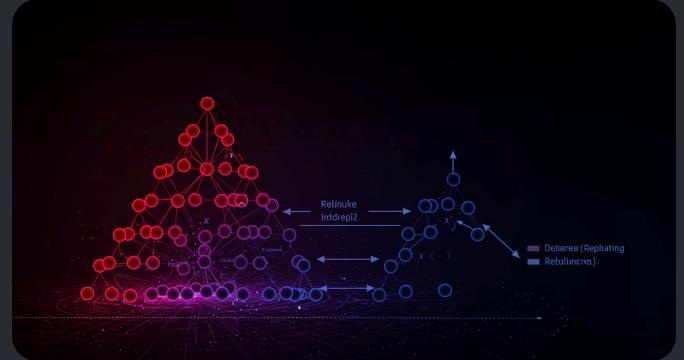
Árvores Rubro-Negras

Propriedades de Coloração

1. Nós são vermelhos ou pretos
2. Raiz é preta
3. Folhas nulas são pretas
4. Filhos de nós vermelhos são pretos
5. Mesmo número de nós pretos em todos caminhos

Vantagens

- Operações $O(\log n)$ garantidas
- Balanceamento mais eficiente que AVL
- Menos rotações na inserção/remoção



Heaps



Aplicações

Heapsort, filas de prioridade, Dijkstra

Algoritmos de Ordenação



Fundamentos

Reorganizam elementos em ordem específica (crescente ou decrescente).

Importância

Operação fundamental presente em quase todos sistemas de informação.

Critérios

Tempo, espaço,
estabilidade,
adaptabilidade e
comportamento com
dados.



Bubble Sort



Funcionamento

Compara pares adjacentes, trocando-os se estiverem fora de ordem.



Implementação Simples

Apenas dois loops aninhados e operação de troca.



Complexidade

$O(n^2)$ no pior e médio caso. $O(n)$ no melhor caso.



Uso Prático

Didático ou para conjuntos muito pequenos de dados.

Quick Sort

Divisão e Conquista

Escolhe um pivô e partitiona o array em elementos menores e maiores.

Repete o processo recursivamente para as duas partições.

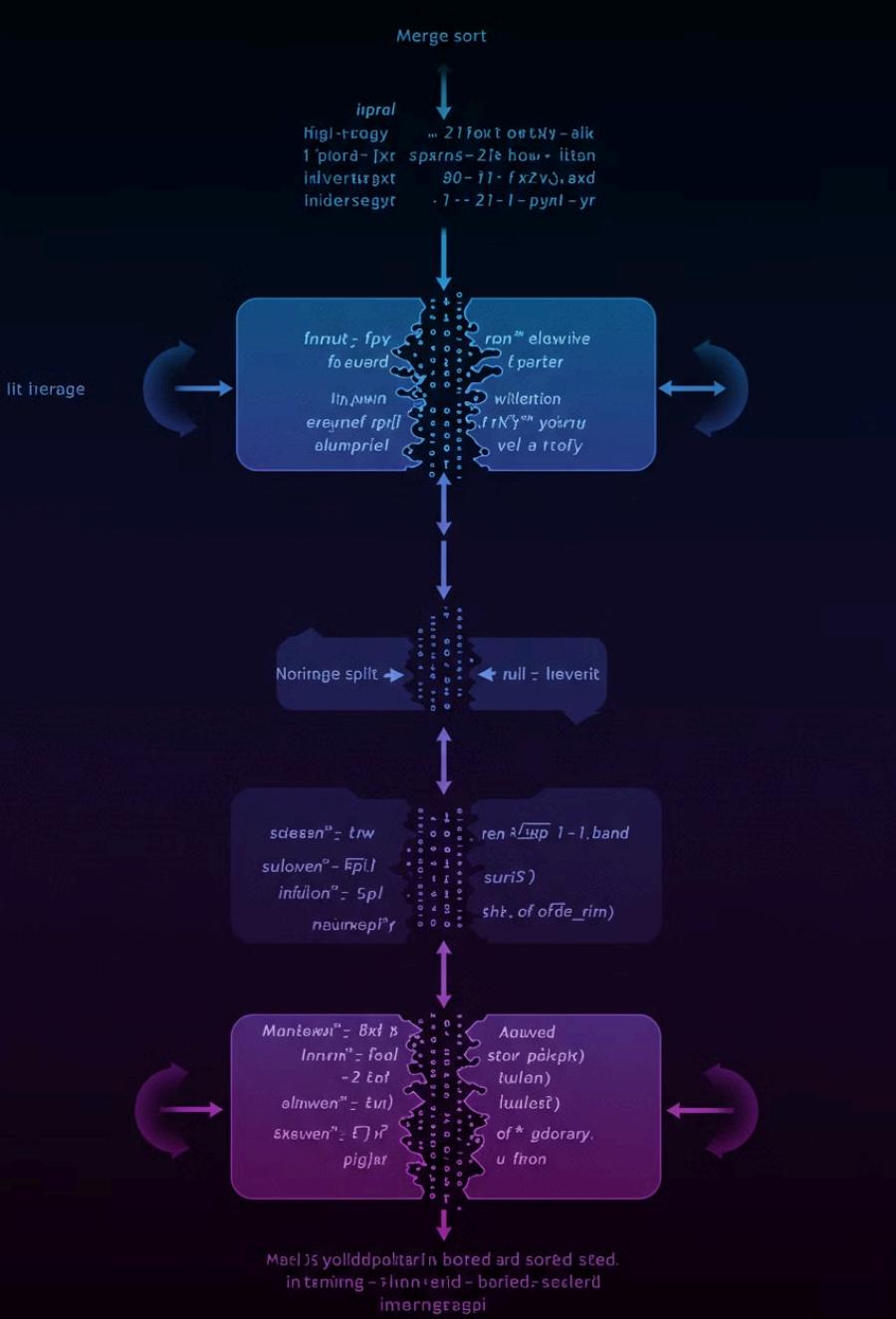
Particionamento

1. Seleciona pivô (geralmente último elemento)
2. Coloca elementos menores à esquerda
3. Coloca elementos maiores à direita
4. Posiciona pivô entre as partições

Complexidade

$O(n \log n)$ em caso médio e melhor.
 $O(n^2)$ no pior caso.

Muito eficiente na prática, utilizado em bibliotecas padrão.



Merge Sort

Divisão

Divide o array em metades até ter subproblemas triviais.

Processo recursivo até chegar a arrays de tamanho 1.

Conquista

Resolve os subproblemas triviais diretamente.

Arrays de um elemento já estão ordenados.

Combinação

Mescla as soluções parciais em ordem.

Junta as metades ordenadas para formar uma sequência completa.

o c n Б h i o p a j s t i c e x a t o o e s t a h g o l - b d

Insertion Sort

Conceito

Constrói uma sequência ordenada um elemento por vez.

Complexidade

$O(n^2)$ no pior e médio caso. $O(n)$ no melhor caso.



Implementação

Insere cada elemento na posição correta dentro da parte já ordenada.

Aplicações

Eficiente para conjuntos pequenos ou quase ordenados.

Grafos



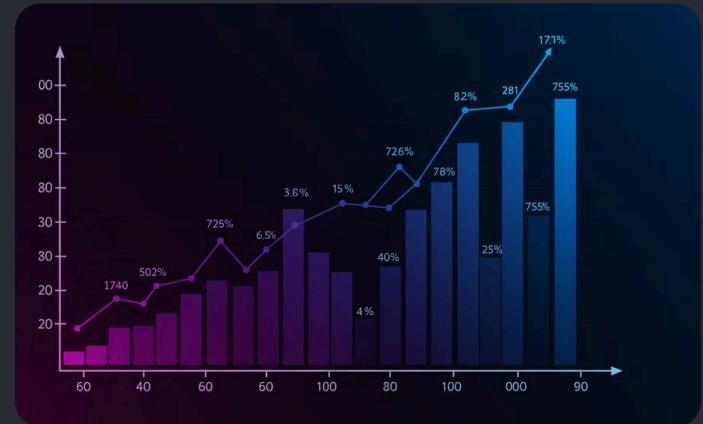
Grafos Direcionados

Arestas têm direção definida, indicando relação de origem-destino.



Grafos Não-direcionados

Arestas representam conexões bidirecionais entre nós.



Grafos Ponderados

Arestas possuem valores (pesos) associados, representando custos ou distâncias.

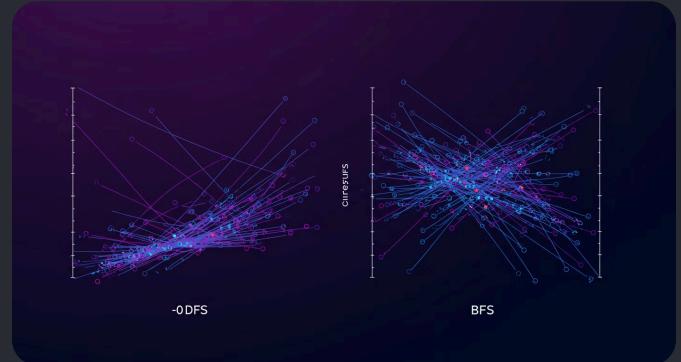
Algoritmos de Busca em Grafos

Busca em Profundidade (DFS)

- Explora um ramo completamente antes de retroceder
- Usa pilha (implícita na recursão)
- Aplicações: detecção de ciclos, ordenação topológica

Busca em Largura (BFS)

- Explora todos vizinhos antes de avançar em profundidade
- Usa fila para processamento
- Encontra caminhos mais curtos em grafos não ponderados





Algoritmo de Dijkstra



Inicialização

Define distância inicial do nó fonte como 0 e demais como infinito.



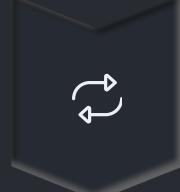
Seleção de Nó

Escolhe nó não visitado com menor distância atual.



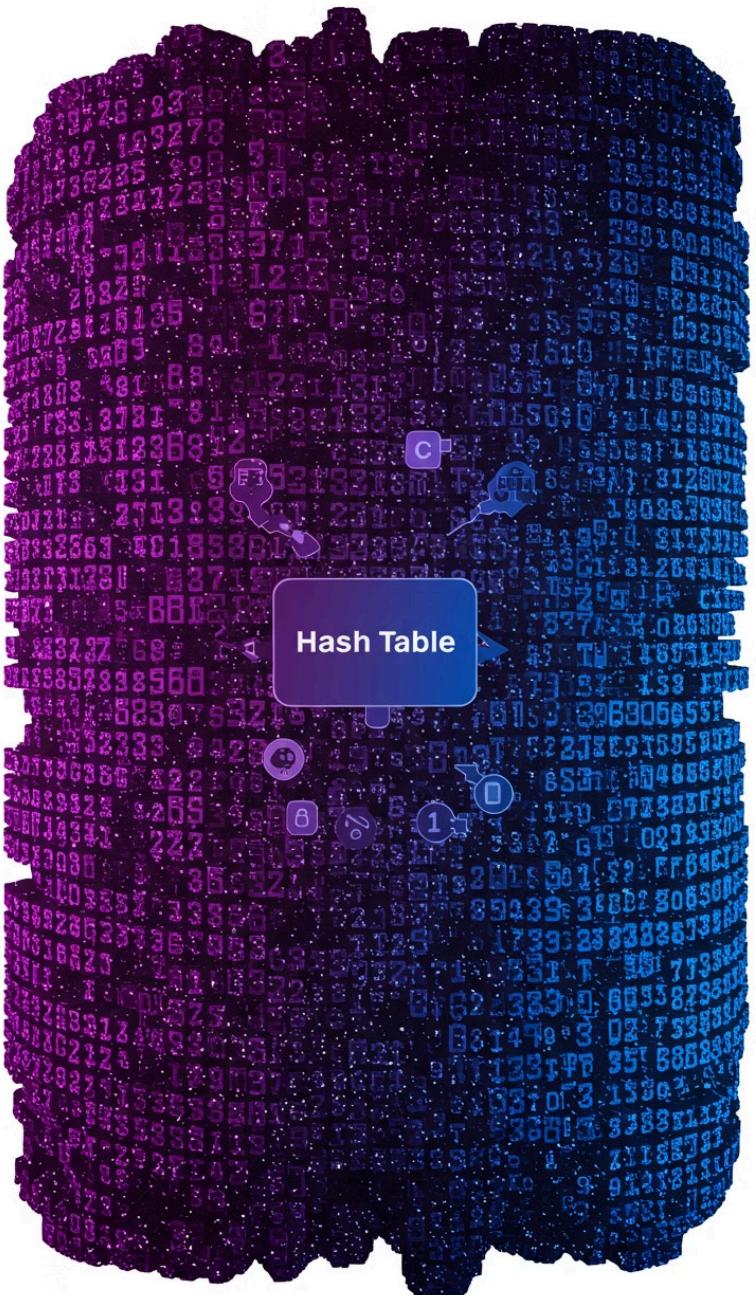
Atualização

Atualiza distâncias dos vizinhos se caminho via nó atual for menor.



Repetição

Repete até todos nós serem visitados ou destino encontrado.



Tabelas Hash

Funcionamento

Transforma chaves em índices de array usando funções hash.

Permite acesso direto aos dados sem busca sequencial.

Colisões

Ocorrem quando diferentes chaves produzem mesmo índice.

Soluções: encadeamento, endereçamento aberto, rehashing.

Função Hash

Deve distribuir valores uniformemente pelo espaço.

Exemplos: divisão, multiplicação, Knuth, criptográficas.

Operações em Tabelas Hash

O(1)

Complexidade de Inserção

Tempo constante em caso médio para adicionar elementos.

O(1)

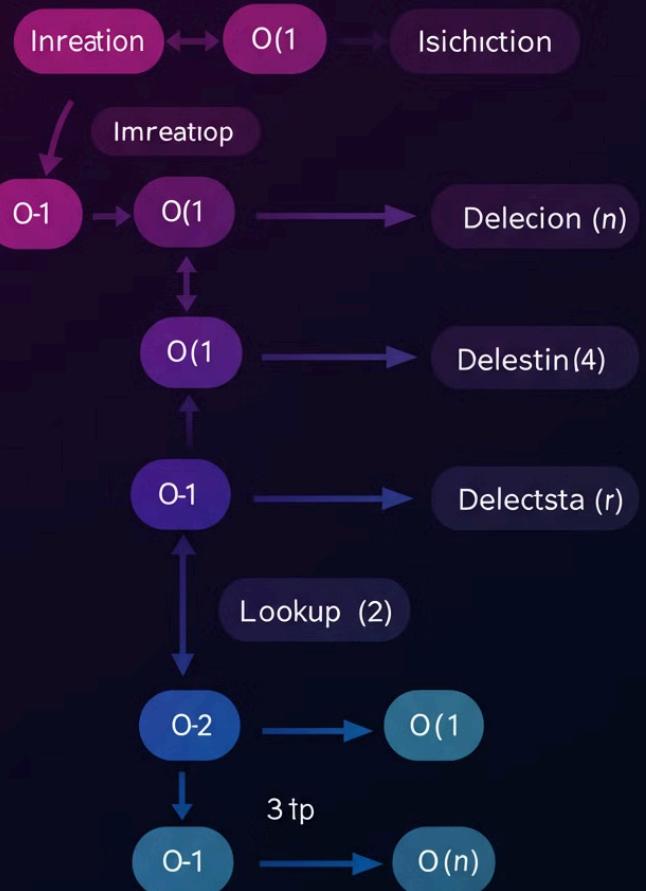
Complexidade de Busca

Acesso direto sem necessidade de busca sequencial.

O(1)

Complexidade de Remoção

Localização e exclusão em tempo constante no caso médio.



Nesedation

Opertion

Árvores de Tries

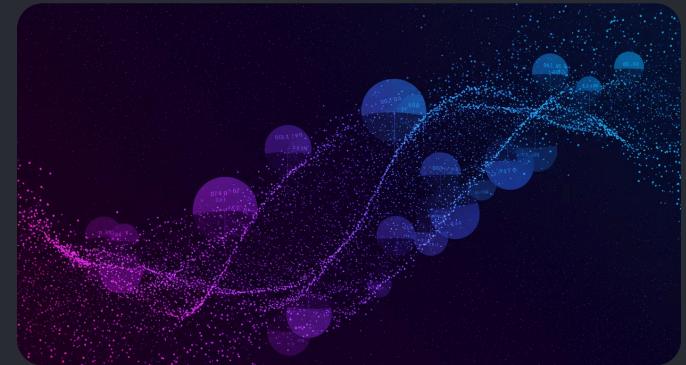
Estrutura Especializada

Árvore de prefixos para armazenamento eficiente de strings.

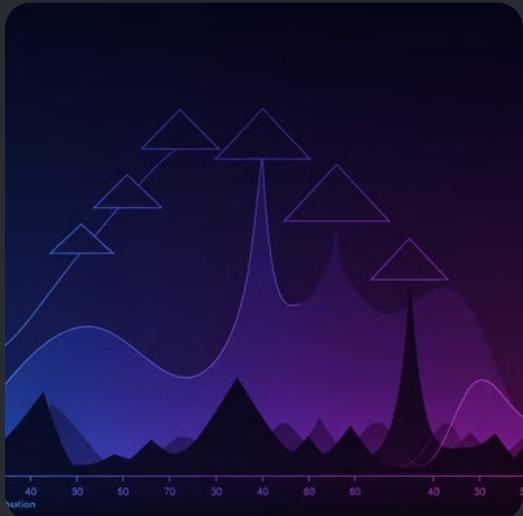
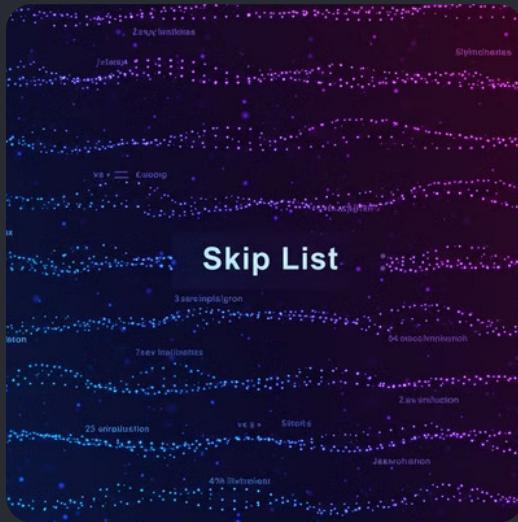
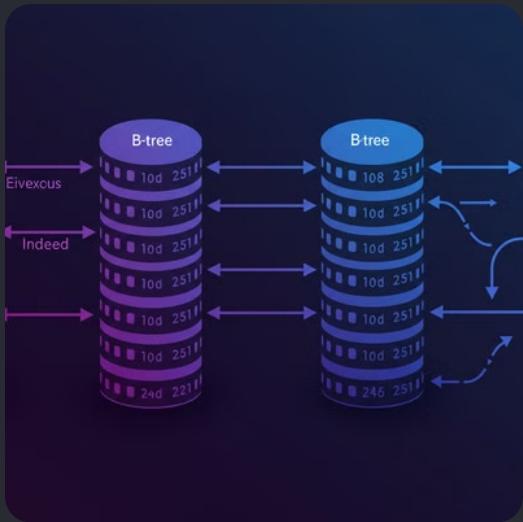
Cada nó representa uma letra ou caractere na sequência.

Vantagens

- Busca por prefixo em $O(m)$ onde m é tamanho da string
- Economiza espaço para palavras com prefixos comuns
- Ideal para dicionários e autocomplete



Estruturas de Dados Avançadas



Estruturas específicas para casos de uso avançados: B-trees, Skip Lists, Segment Trees, Bloom Filters e Quad Trees.



Compressão de Dados

Algoritmos de Compressão

Huffman, LZW, Run-length encoding, deflate.

Estruturas Eficientes

Árvores de Huffman, dicionários, tries.

Otimização

Balanceamento entre taxa de compressão e velocidade.

Memória e A alocação Dinâmica

Gestão de Memória

- Stack: alocação automática, LIFO
- Heap: alocação manual, flexível
- Garbage collection vs. liberação manual

Tipos de Alocação

- Estática: tamanho conhecido em compilação, limitada mas eficiente.
- Dinâmica: tamanho determinado em execução, flexível mas com overhead.

Otimização

- Pool allocation
- Memory chunking
- Fragmentação e compactação

Recursão

Caso Base

Condição de parada que resolve diretamente

Resultado

Combinação das respostas para formar solução final

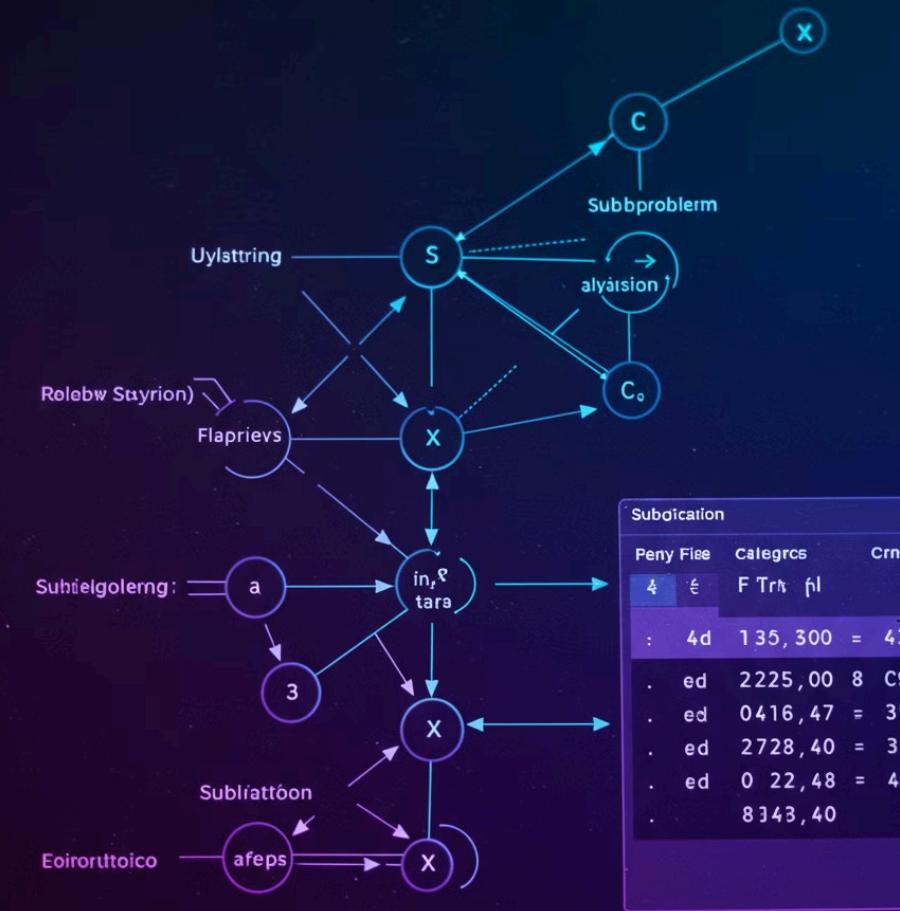


Caso Recursivo

Chama a própria função com subproblema

Pilha de Chamadas

Armazena contexto de execução



Programação Dinâmica

Subproblemas

Divide o problema original em partes menores sobrepostas.

Identifica estrutura de subproblemas recorrentes.

Memoização

Armazena resultados de subproblemas em tabela.

Evita recálculos desnecessários, otimizando desempenho.

Construção de Solução

Combina resultados dos subproblemas para resolver problema original.

Pode ser bottom-up (tabulação) ou top-down (recursivo com memoização).



Algoritmos Gulosos



Escolha Ótima Local

Toma decisão localmente ótima a cada passo.



Princípio da Esperança

Espera que escolhas locais levem a solução global ótima.



Exemplos Clássicos

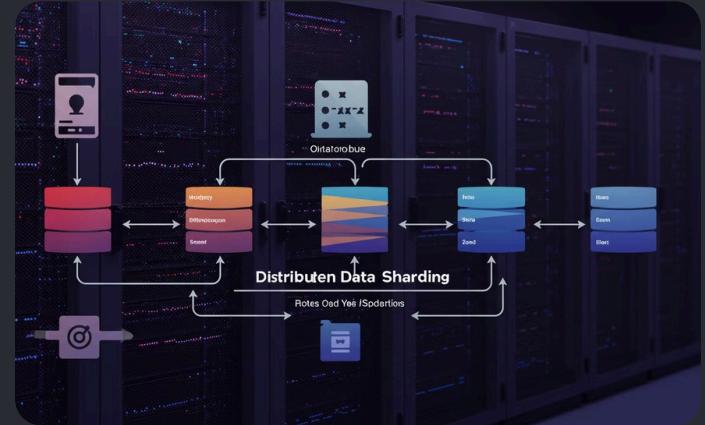
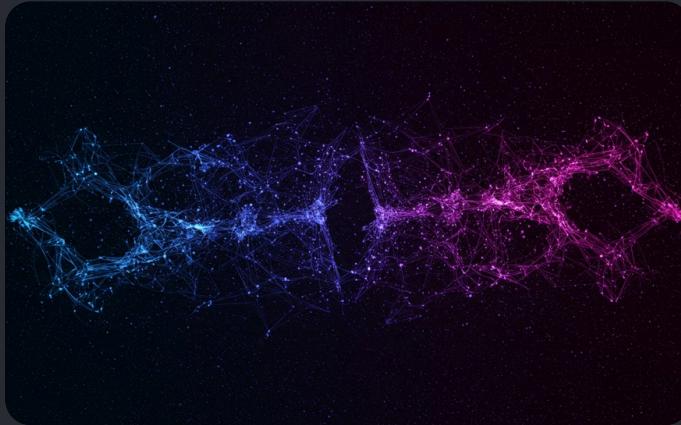
Algoritmo de Dijkstra, árvore geradora mínima, codificação de Huffman.



Limitações

Nem sempre encontra a solução ótima global.

Estruturas de Dados Distribuídas



Tabelas Hash Distribuídas

Permite armazenamento e acesso eficiente em redes P2P.

Hashing Consistente

Minimiza redistribuição de dados quando nós são adicionados ou removidos.

Sharding

Particiona dados entre múltiplos servidores para escalar horizontalmente.

Paralelismo e Estruturas de Dados

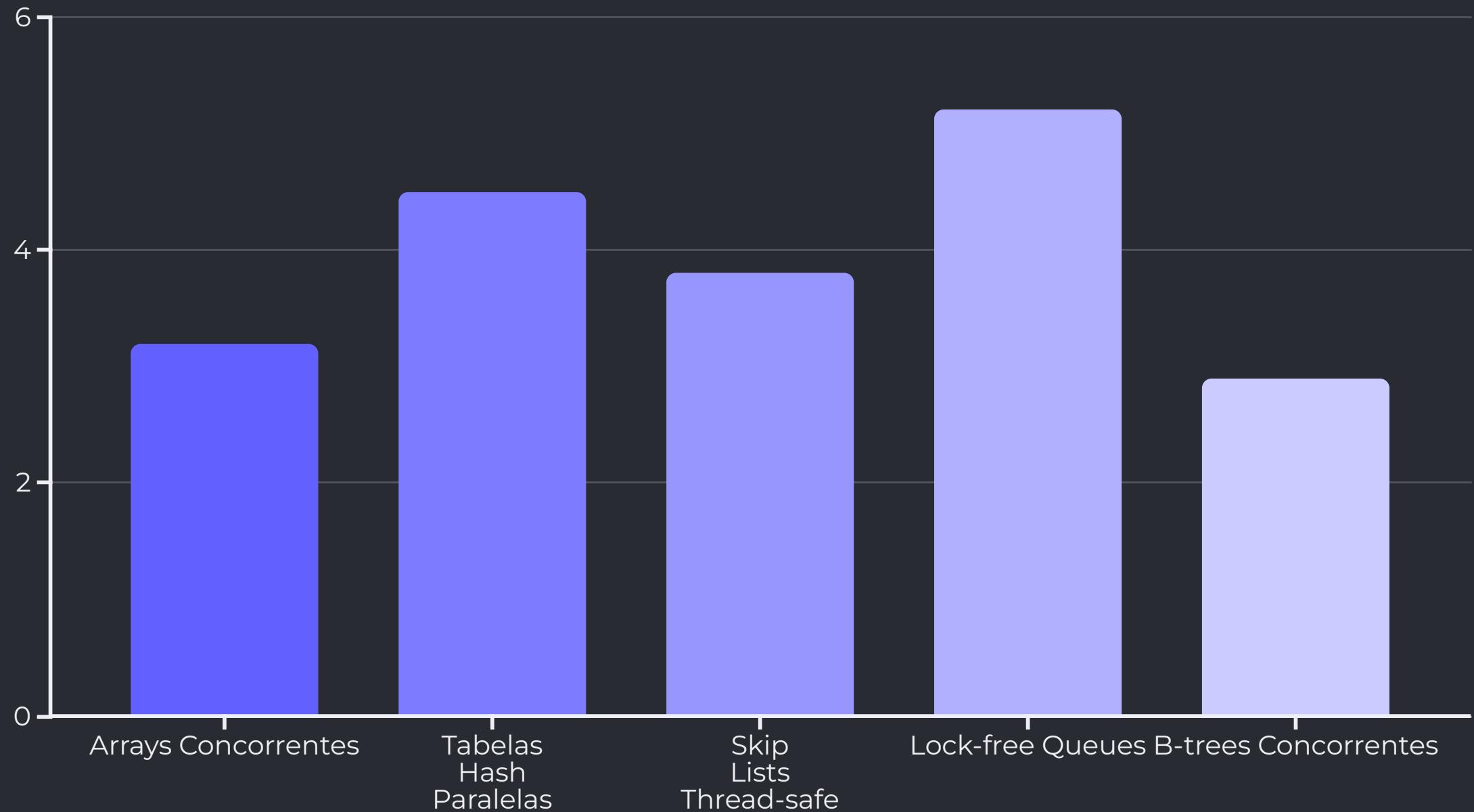


Gráfico comparativo de ganho de desempenho (speedup) em estruturas de dados paralelas com 8 threads em relação a versões sequenciais.

Estruturas de Dados em Machine Learning

Matrizes Esparsas

Armazemam eficientemente dados com muitos valores zero.

Essenciais para processamento de texto e redes neurais.

Grafos de Computação

Representam operações matemáticas para diferenciação automática.

Base de frameworks como TensorFlow e PyTorch.

KD-Trees

Aceleram buscas por vizinhos próximos em espaços multidimensionais.

Aplicações em clustering e classificação.

Estruturas de Dados em Big Data



Bloom Filters

Estrutura probabilística para testar pertinência com uso mínimo de memória.



Count-Min Sketch

Estima frequência de eventos em streams de dados com memória limitada.



HyperLogLog

Aproxima contagem de elementos distintos com precisão logarítmica.



LSM Trees

Otimiza escrita em bancos de dados com alta concorrência.

Segurança de Dados



Otimização de Desempenho



Análise

Identificação de gargalos com ferramentas de profiling

2

Algoritmos

Seleção de estruturas adequadas ao caso de uso



Memória

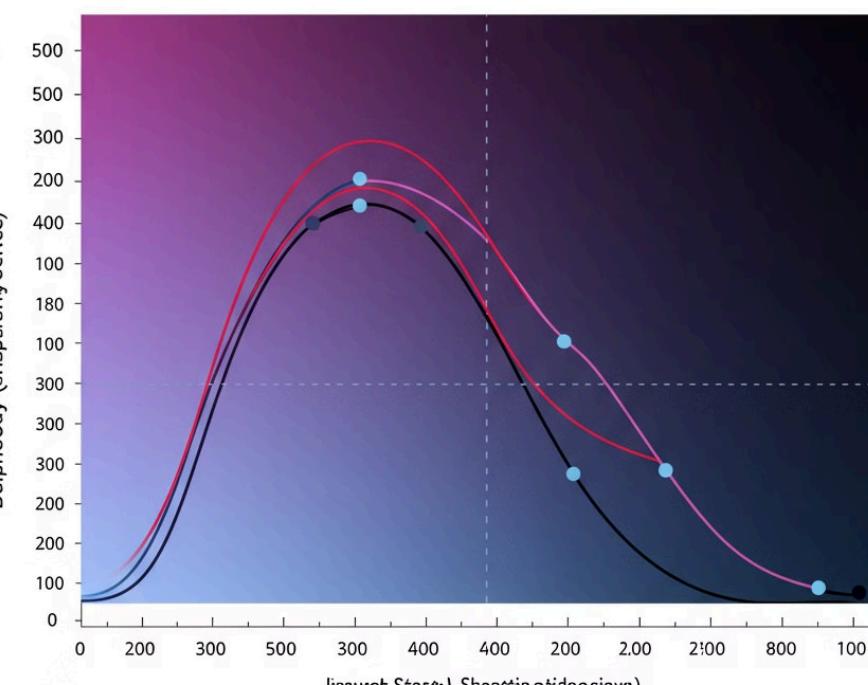
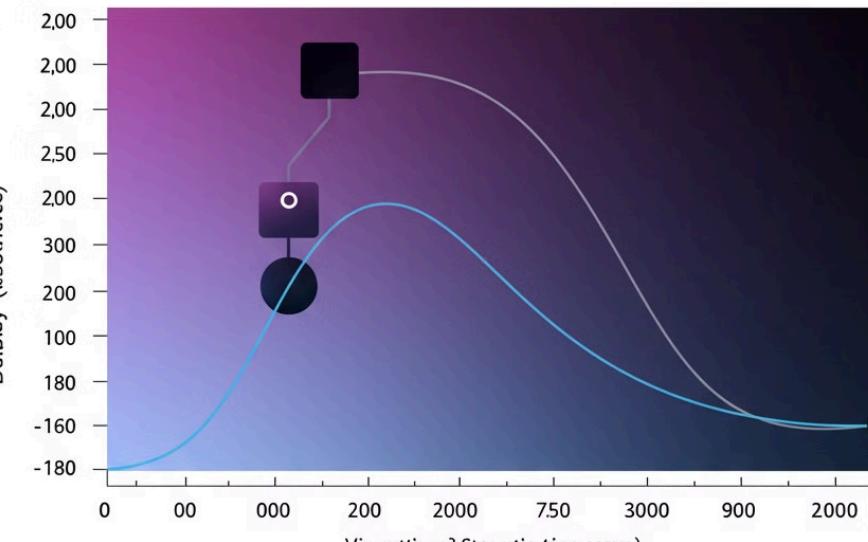
Otimização de alocação e aproveitamento de cache



Paralelismo

Utilização de múltiplos núcleos quando apropriado

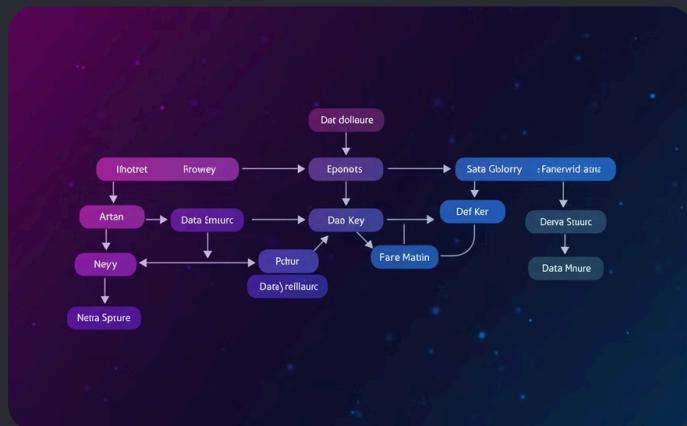
Binary Search Searóm



Algoritmos de Busca

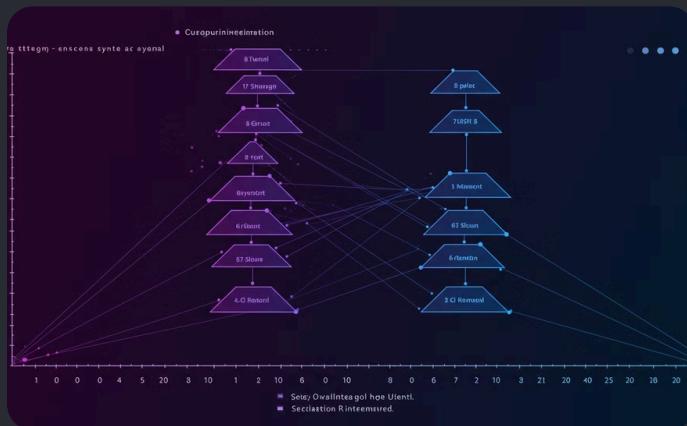
Algoritmo	Melhor Caso	Pior Caso	Requisitos
Busca Linear	$O(1)$	$O(n)$	Nenhum
Busca Binária	$O(1)$	$O(\log n)$	Dados ordenados
Busca por Hash	$O(1)$	$O(n)$	Função hash
Árvore Binária	$O(\log n)$	$O(n)$	Árvore balanceada

Estruturas de Dados em Banco de Dados



Índices B-Tree

Estrutura de árvore balanceada que acelera consultas e ordenação.



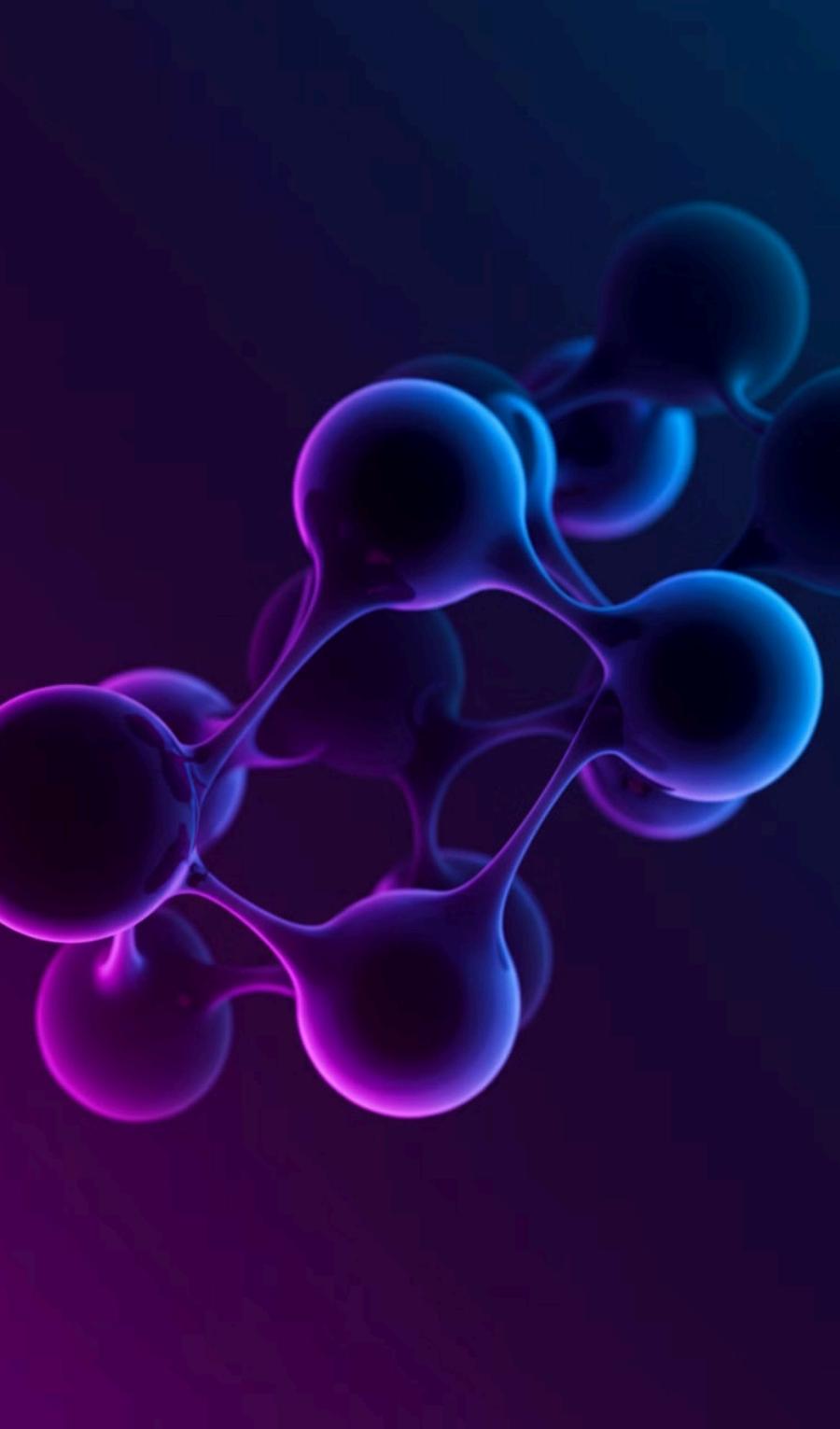
Índices Espaciais

R-trees e Quadtrees para dados geográficos e multidimensionais.



Índices Hash

Acesso $O(1)$ para busca exata de chaves em bancos NoSQL.



Tendências em Estruturas de Dados



Estruturas para IA

Otimizadas para redes neurais e processamento de grafos.

2

Computação Quântica

Novas estruturas explorando sobreposição e entrelaçamento.



Sistemas Distribuídos

CRDT, log estruturado e estruturas para consistência eventual.



Persistência

Estruturas otimizadas para novas tecnologias de armazenamento não-volátil.

Desafios Computacionais

Problemas NP-Completos

Caixeiro viajante, problema da mochila, coloração de grafos.

Requerem algoritmos de aproximação ou heurísticas.

Limitações de Hardware

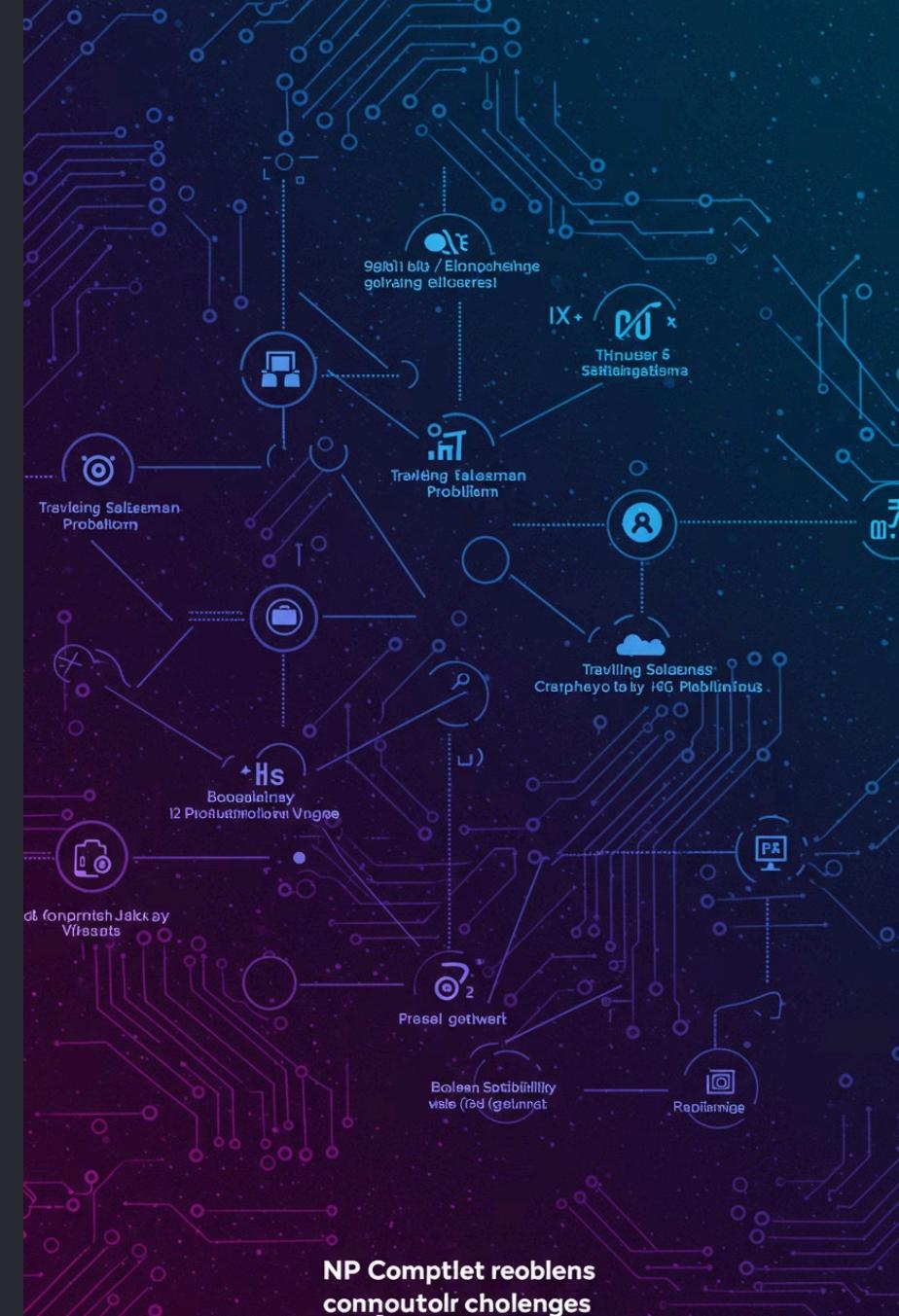
Lei de Moore desacelerando, arquiteturas alternativas emergindo.

Necessidade de estruturas otimizadas para novos paradigmas.

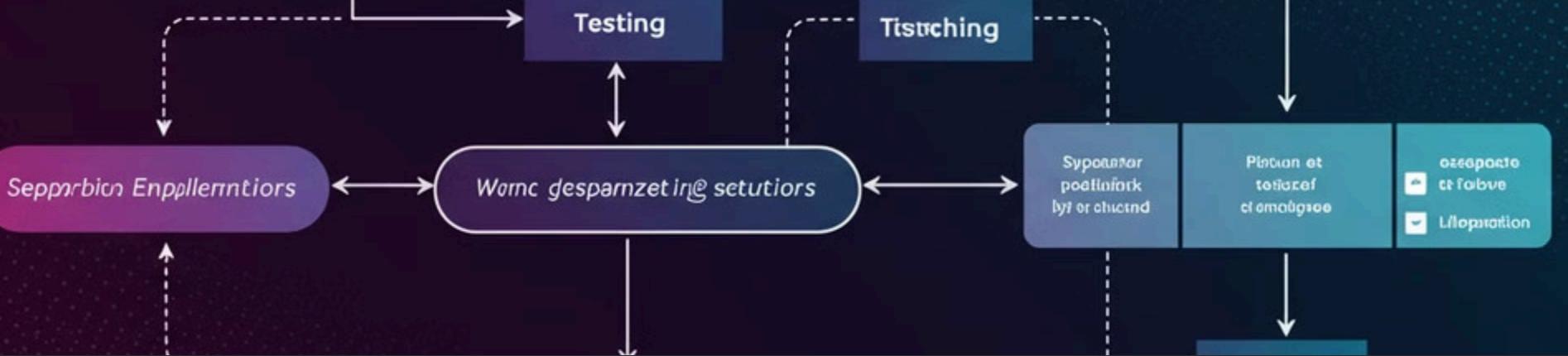
Escalabilidade

Dados crescendo exponencialmente, ultrapassando capacidade de processamento.

Estruturas distribuídas e aproximadas ganham relevância.



NP Complet reoblens
connoutor cholenges



Implementação Prática



Escolha Adequada

Selecione estruturas conforme requisitos de tempo, espaço e operações.



Abstração

Use interfaces para permitir trocas de implementação sem afetar código cliente.



Testes

Verifique comportamento em casos limite e grandes volumes de dados.



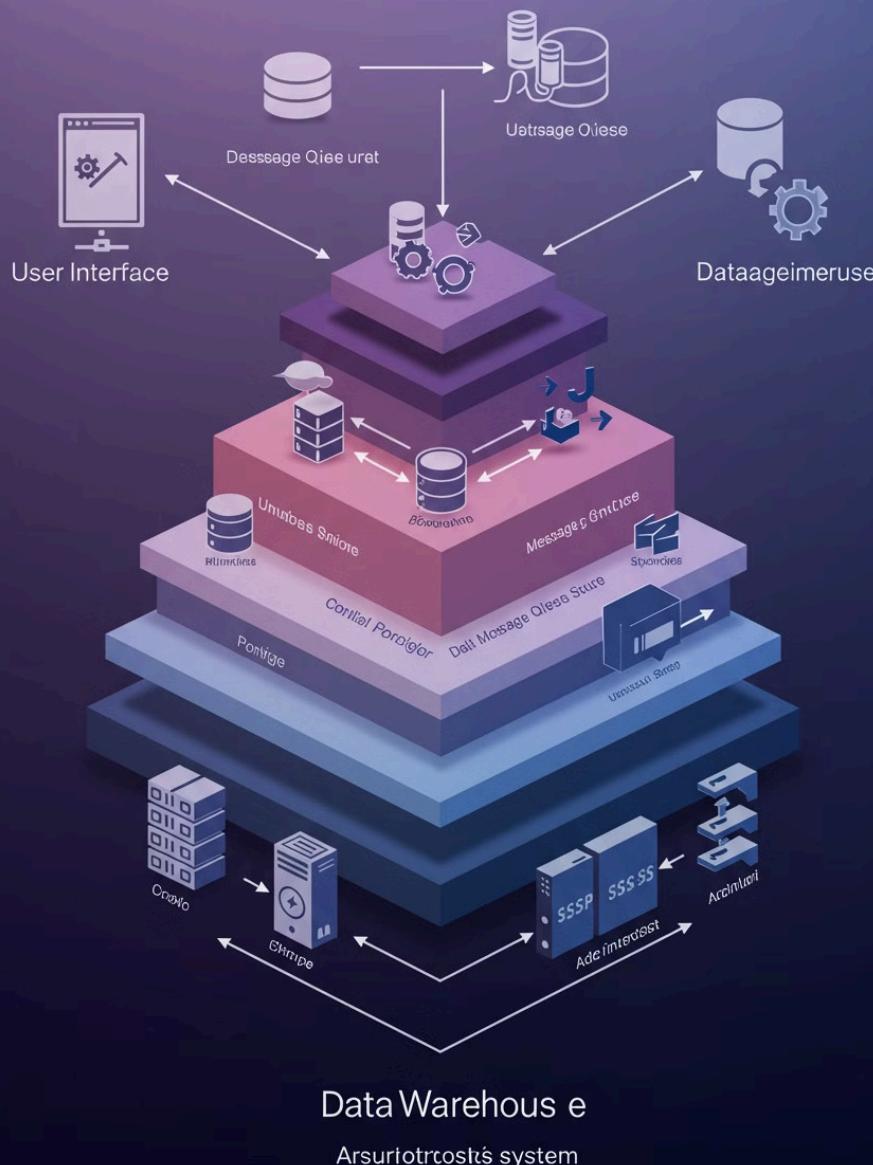
Otimização

Otimize apenas após identificar gargalos reais com profiling.

Ferramentas e Bibliotecas



Bibliotecas padronizadas oferecem implementações otimizadas e testadas de estruturas de dados comuns, economizando tempo de desenvolvimento e garantindo confiabilidade.



Arquitetura de Software

Modelagem de Dados

Estruturas escolhidas influenciam design das entidades.

2

Padrões de Projeto

Observer, Composite, Iterator dependem de estruturas específicas.

3

Arquitetura em Camadas

Estruturas diferentes otimizadas para acesso, lógica e persistência.

4

Performance

Escolhas impactam diretamente escalabilidade e responsividade.

Casos de Estudo

10x

Google Search

Índices invertidos para busca rápida em documentos.

99.9%

Amazon DynamoDB

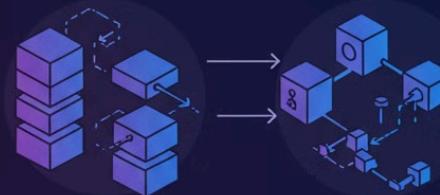
Consistência eventual com tabelas hash distribuídas.

500M+

Netflix Recommendation

Grafos bipartidos para modelar preferências de usuários.

Arrays

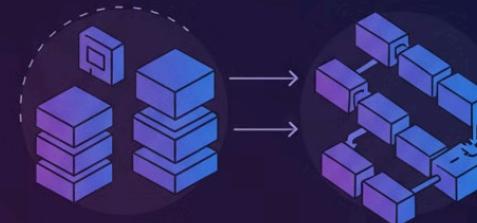


Orraayan Structure

Tine Tiep

Lorem ipsum dolor sit amet, tetim-comparicing eliq pülmurumy elh mod dœces venumod scatulis paqut.

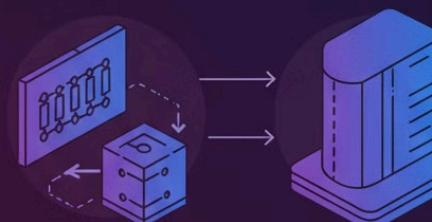
Linked lists



Hash Tables

Lorem ipsum dolor sit amet, tetim-comparicing end polinnumy silh mod dœcest venumme scatürle paqut.

Foyrron Ociure

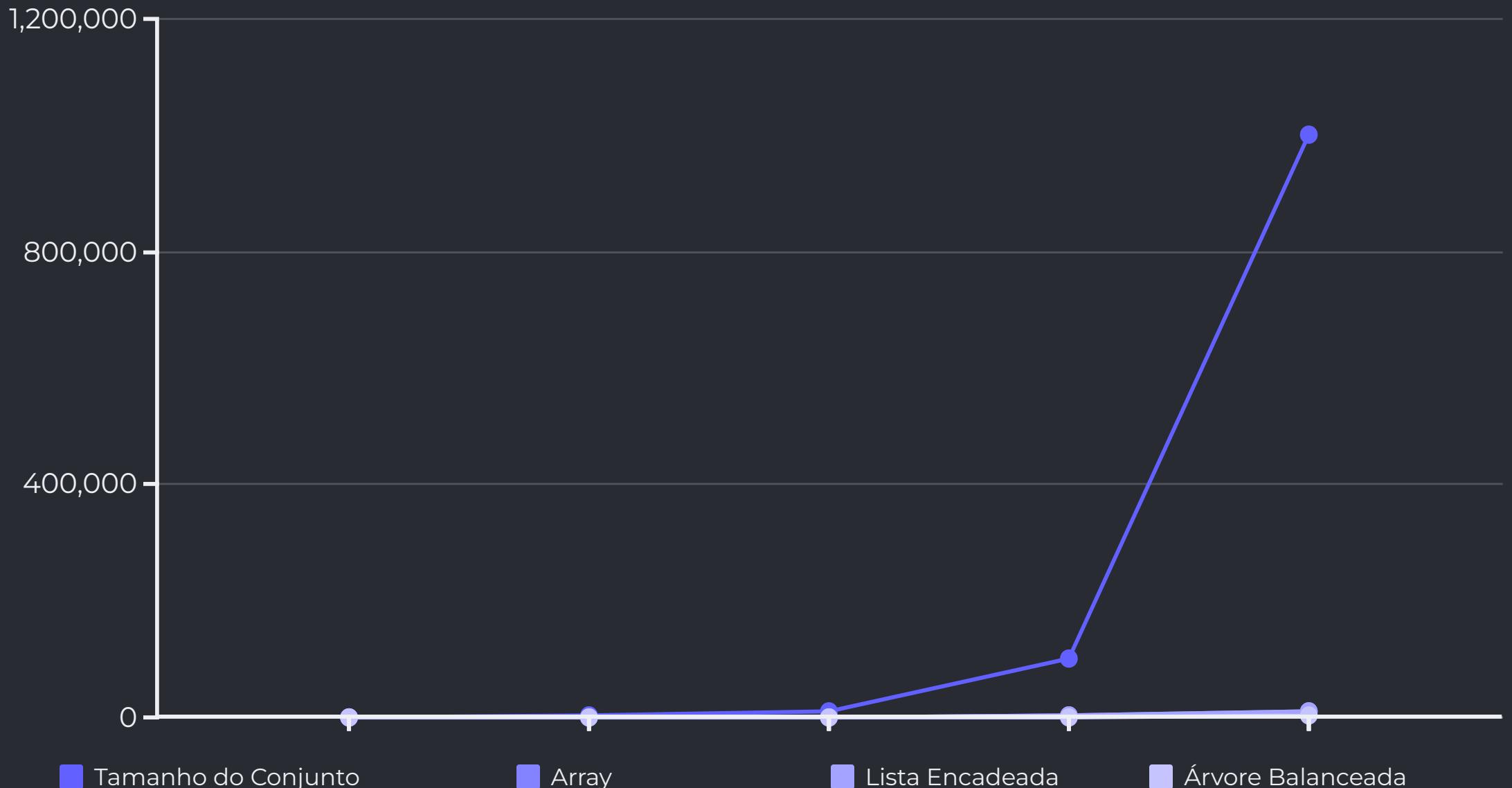


Hash Tables

Lorem ipsum dolor sit amet, tetim-comparicing eliq pülmurumy elh mod dœces venumod scatürle paqut.

Optamive_peformance

Comparação de Desempenho



Tempo de busca (ms) para diferentes estruturas conforme o volume de dados cresce. Note como árvores平衡adas se tornam mais eficientes em grandes conjuntos.

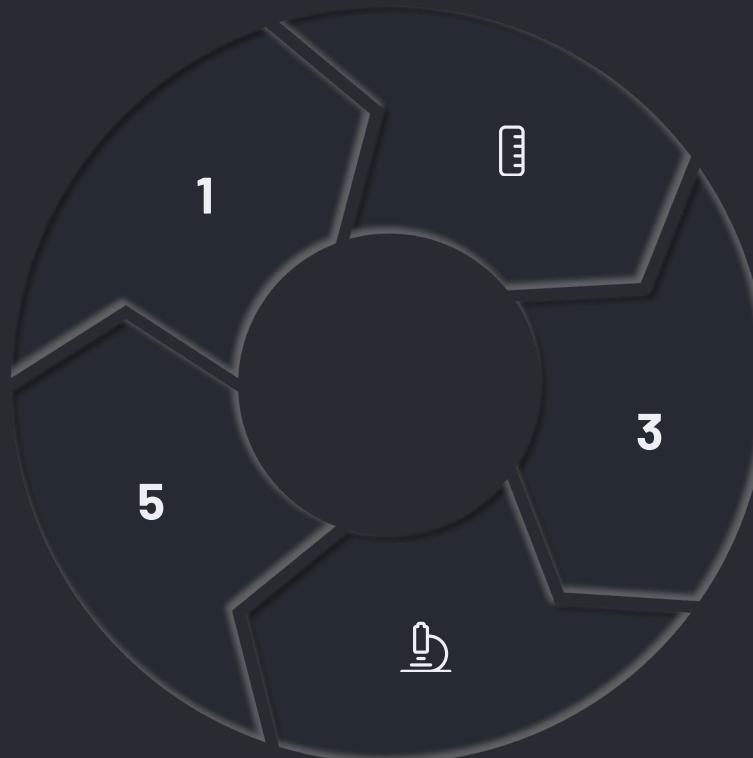
Desenvolvimento de Software

Requisitos

Análise de operações necessárias e volumes de dados

Otimização

Refinamento baseado em métricas reais



Design

Seleção de estruturas e algoritmos apropriados

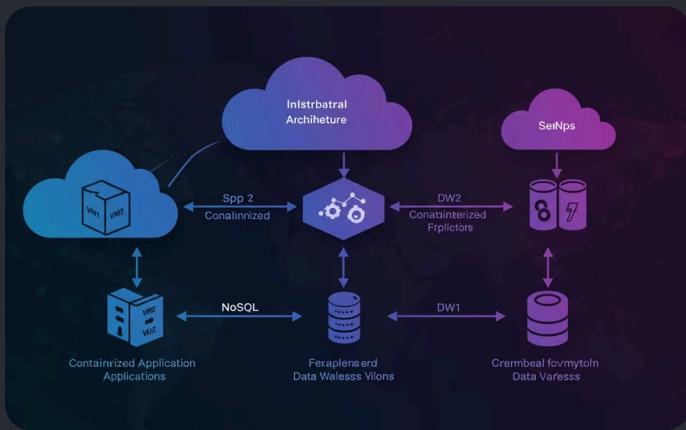
Implementação

Codificação eficiente ou uso de bibliotecas

Testes

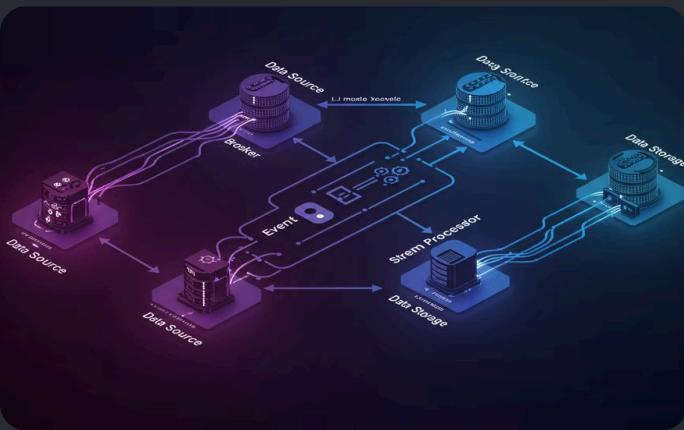
Verificação de comportamento e desempenho

Computação em Nuvem



Bancos Distribuídos

Estruturas otimizadas para consistência eventual e particionamento.



Streams de Eventos

Filas e logs para processamento assíncrono e resiliente.



Computação Serverless

Estruturas que minimizam latência de inicialização e otimizam memória.