Projeto Travô Aki

Análise de Algoritmos e Complexidade Computacional

José Vinicius Garcia Rodrigues | Algoritmos e Complexidade | 2025

Visão Geral do Sistema

Introdução e Escopo

O que é

Sistema Web de Help Desk para gerenciamento de filas de chamados.

Arquitetura

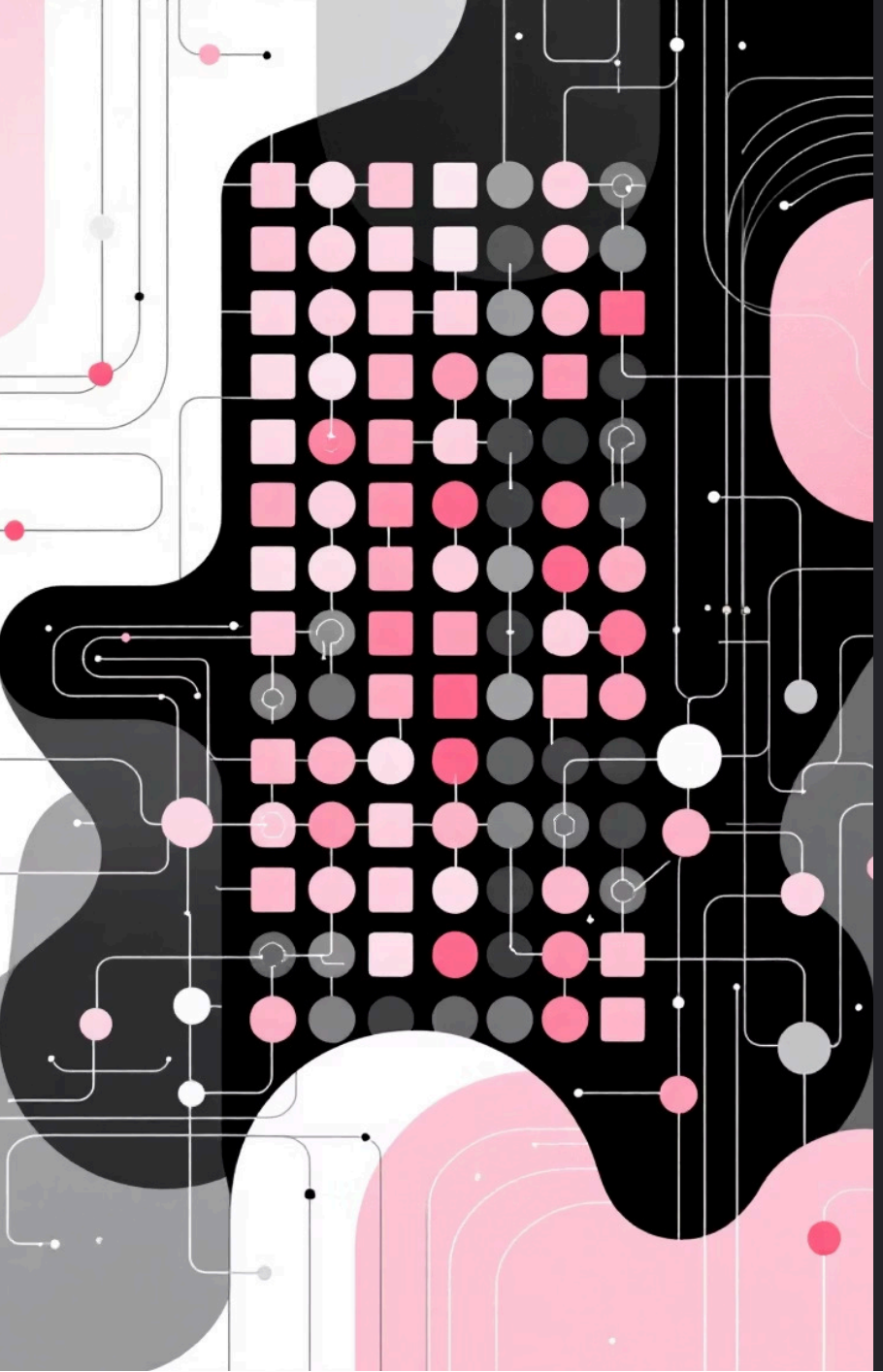
MVC (Model-View-Template) utilizando framework Django.

Banco de Dados

Relacional (MySQL) com indexação B-Tree.

Objetivo Acadêmico

Demonstrar aplicação prática de estruturas de dados e análise de eficiência em um cenário real.



Estruturas de Dados (1/2)

Estruturas de Acesso Rápido

Tabelas Hash (Hash Tables)

- Utilizadas nativamente pelo Python (dict) e pelo ORM do Django.
- *Aplicação:* Autenticação de usuários (Sessões) e Roteamento de URLs.
- *Complexidade:* Acesso médio em $O(1)$.

Árvores B+ (B-Trees)

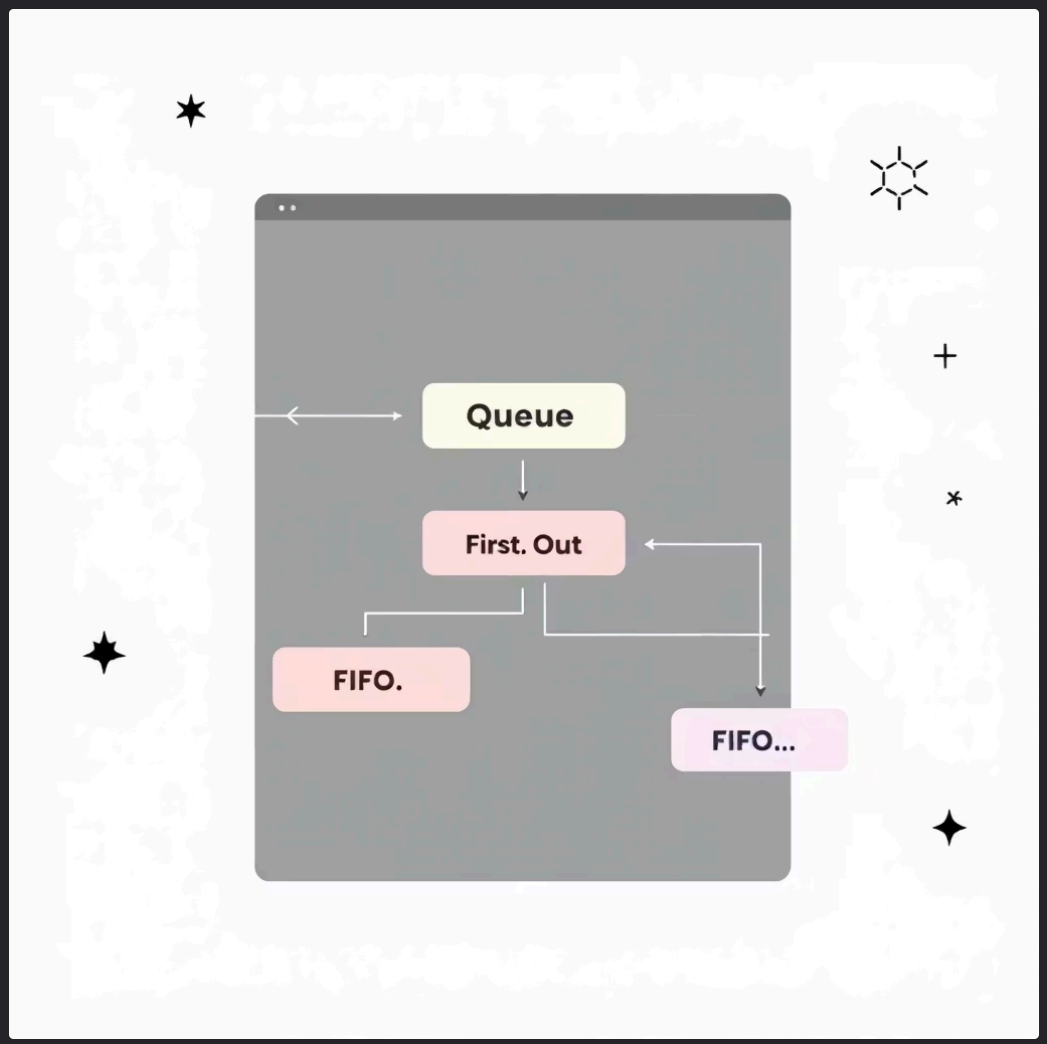
- Estrutura base dos índices do MySQL.
- *Aplicação:* Armazenamento e busca de Chaves Primárias (id) e UUIDs (ticket_id).
- *Vantagem:* Reduz operações de disco de linear para logarítmica.

Estruturas de Dados (2/2)

Lógica de Organização e Relacionamento

Filas (Queues)

- O sistema implementa uma **Fila de Prioridade** lógica.
- *Critério:* Ordenação por Status (Prioridade) e Data de Criação (FIFO - First In, First Out).



Grafos Direcionados

- O modelo relacional forma um grafo de dependências.
- *Nós:* Técnicos, Tickets, Interações.
- *Arestas:* Chaves Estrangeiras (Foreign Keys).
- *Integridade:* Remoção em cascata (Cascade Delete) mantém a consistência do grafo.

Algoritmos de Busca

Estratégia Híbrida de Busca

O sistema utiliza Polimorfismo na busca para decidir qual algoritmo usar:

Busca Indexada (Eficiente)

- *Gatilho:* Quando o input é um UUID válido.
- *Algoritmo:* Busca em Árvore B+.
- *Custo:* Baixo (Logarítmico).

Busca Textual (Exaustiva)

- *Gatilho:* Quando o input é texto (Nome/Empresa).
- *Algoritmo:* Varredura Linear (Table Scan) com *String Matching*.
- *Custo:* Alto (Linear).

Algoritmos de Ordenação

Algoritmo Timsort

Utilizado na ordenação do Dashboard (ORDER BY created_at DESC).

Definição

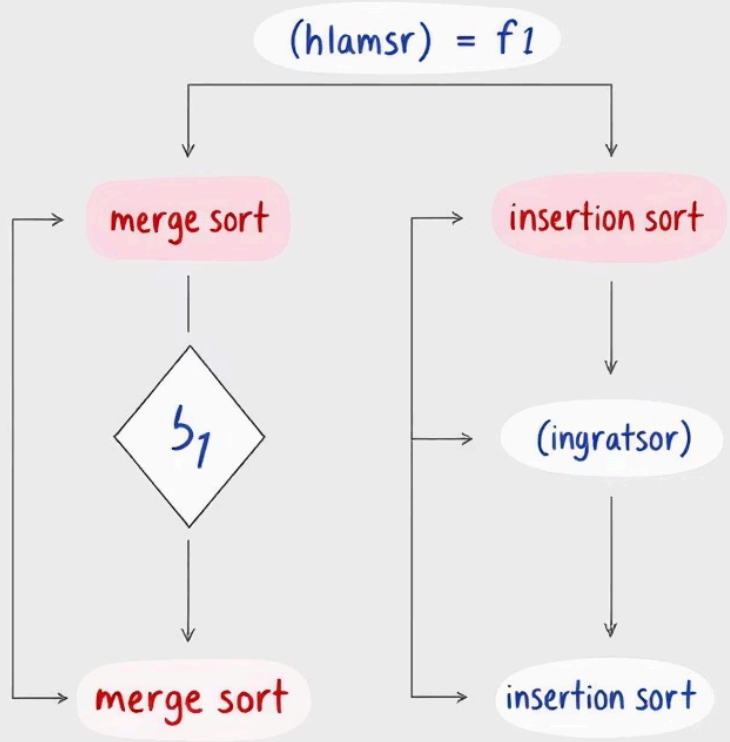
O Timsort é o algoritmo padrão do Python e de muitos SGBDs modernos.

Funcionamento

Híbrido de *Merge Sort* (Divisão e Conquista) e *Insertion Sort*.

Por que foi escolhido?

- É um algoritmo **Estável** (mantém a ordem relativa de itens iguais).
- Ótimo desempenho em dados "do mundo real" (que já possuem sequências parcialmente ordenadas).



Análise de Complexidade (Busca)

Complexidade Assintótica: Busca por Protocolo

Caso	Notação	Descrição
Melhor Caso	$\Omega(1)$	Elemento na raiz da árvore ou em cache de memória.
Caso Médio	$\Theta(\log n)$	Navegação padrão pelos nós da Árvore B+.
Pior Caso	$O(\log n)$	Elemento na folha mais profunda (árvore balanceada).

Onde n é o número de chamados no banco.

Análise de Complexidade (Texto)

Complexidade Assintótica: Busca Textual

Caso	Notação	Descrição
Melhor Caso	$\Omega(n)$	O banco lê a primeira página de dados (overhead mínimo).
Caso Médio	$\Theta(n \cdot k)$	Varredura linear comparando a string de tamanho k.
Pior Caso	$O(n \cdot k)$	O termo não existe ou está no último registro (Full Table Scan).

Nota: Esta busca é mais custosa, porém necessária para a usabilidade (UX).

Análise de Complexidade (Ordenação)

Complexidade Assintótica: Timsort

Caso	Notação	Descrição
Melhor Caso	$\Omega(n)$	Dados já ordenados (apenas uma verificação linear).
Caso Médio	$\Theta(n \log n)$	Custo padrão de ordenações baseadas em comparação.
Pior Caso	$O(n \log n)$	Dados totalmente aleatórios ou inversos.

Conclusão

Considerações Finais

- **Eficiência**

  O uso de UUIDs indexados garante escalabilidade para milhões de registros ($O(\log n)$).

- **Robustez**

  A escolha do MySQL (Árvores B+) e Python (Timsort) fornece uma base sólida para operações de ordenação e busca.

- **Requisitos Atendidos**
  - ✅ Busca Eficiente (Hash/B-Tree).
  - ✅ Ordenação e Ranking.
  - ✅ Simulação de Processos (Filas).