Centro Universitário de Excelência
Sistemas de Informação

Relatório Técnico - Científico Sistema de Gerenciamentos de Pedidos Tia Lu Food Delivery - Algoritmo de ordenação e Árvore Binária de Busca

Grupo Pernambuco

Autor: Jefte Martins
Pedro Silveira
Paulo Soares
David Cairo
Ian Neves

Vitória da Conquista, 2025

Agenda

O Objetivo dessa apresentação é

Este relatório discute a refatoração do sistema de pedidos da Tia Lu Food Delivery em Python, utilizando mapas, arquivos JSON e algoritmos de ordenação. O Quick Sort organiza pedidos eficientemente, enquanto a árvore AVL garante acesso estruturado às informações. Essas técnicas aumentam a confiabilidade do sistema e sua capacidade de atender diversas necessidades.

1. Introdução
2. Fundamentação Teórica
3. Metodologia de Implementação
4. Integração e Persistência
5. Considerações Finais

Introdução

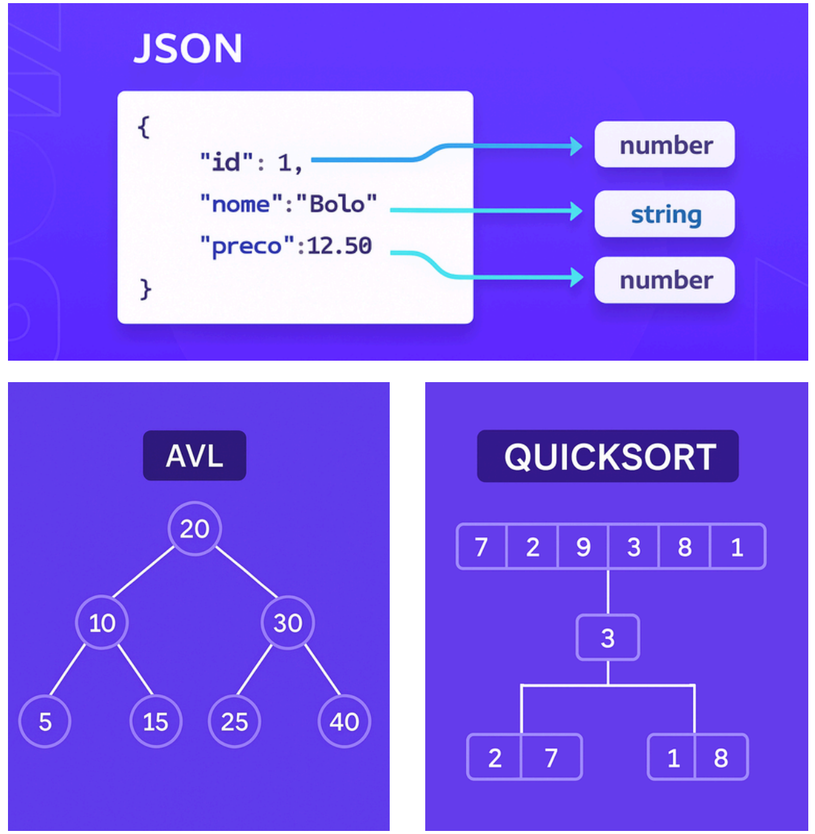
O Desafio Anterior

O sistema antigo usava listas e buscas sequenciais, ficando lento com muitos pedidos. Refatoramos usando QuickSort, Árvore AVL e JSON para melhorar desempenho, organização e persistência dos dados.

Algoritmo de ordenação e Árvore Binária de Busca

Fundamentação Teórica

QuickSort: algoritmo rápido de ordenação baseado em pivô e recursão (complexidade média O(n log n)).
Árvore AVL: árvore de busca balanceada que garante buscas rápidas em O(log n).
JSON: formato simples para armazenar e recuperar dados entre execuções.



Algoritmo de ordenação e Árvore Binária de Busca

Metodologia de Implementação

Carregamos os dados do JSON, inserimos tudo nas árvores AVL e usamos QuickSort para organizar listagens. O sistema foi modularizado e cada operação (itens, clientes, pedidos) usa busca rápida pela AVL.

```python
def criar_no(id, dado):
    return {
        'id': id,
        'dado': dado,
        'esquerda': None,
        'direita': None,
        'altura': 1
    }
```

```python
def inserir(no, id, dado):
    if not no:
        return criar_no(id, dado)

    if id < no['id']:
        no['esquerda'] = inserir(no['esquerda'], id, dado)
    elif id > no['id']:
        no['direita'] = inserir(no['direita'], id, dado)
    else:
        no['dado'] = dado
        return no

    no['altura'] = 1 + max(get_altura(no['esquerda']), get_altura(no['direita']))
    balanceamento = get_balanceamento(no)
```

Algoritmo de ordenação e Árvore Binária de Busca

RESULTADOS E DISCUSSÕES

As buscas ficaram imediatas, as listagens passaram a sair ordenadas e os arquivos JSON mantêm tudo sincronizado. O sistema ficou mais rápido, organizado e preparado para crescer.

```python
def buscar_elemento(no, id):
    if not no:
        return None
    if id == no['id']:
        return no['dado']
    if id < no['id']:
        return buscar_elemento(no['esquerda'], id)
    return buscar_elemento(no['direita'], id)
```

Algoritmo de ordenação e Árvore Binária de Busca

Considerações Finais

A refatoração atingiu o objetivo: melhorar desempenho e estrutura do sistema. A AVL acelerou as buscas, o QuickSort organizou as listagens e o JSON garantiu persistência confiável.

```python
def rotacionar_esquerda(x):
    y = x['direita']
    T2 = y['esquerda']

    y['esquerda'] = x
    x['direita'] = T2

    x['altura'] = 1 + max(get_altura(x['esquerda']), get_altura(x['direita']))
    y['altura'] = 1 + max(get_altura(y['esquerda']), get_altura(y['direita']))

    return y
```