CURSO DE CIÊNCIA DA COMPUTAÇÃO

RELATÓRIO SRHP

Estrutura de Dados

MACEIÓ - AL

2025

ANTÔNIO GABRIEL DOS SANTOS BARBOSA
EDUARDA RAFAELLA SILVA MELO
EUDES DE OLIVEIRA ROCHA
DANILO SOARES MATOS
NATALIE CAVALCANTE
VINICIUS AUGUSTO

RELATÓRIO SRHP

Trabalho da disciplina de Estrutura de Dados, do curso de Ciência da Computação da Unima - Afya, sob a orientação dos Instrutores Ícaro Ferreira e Marcos Bento, apresentado como pré-requisito para a avaliação do aprendizado realizado na disciplina.

MACEIÓ - AL
2025

RESUMO

O presente relatório técnico descreve o desenvolvimento do Sistema de Recomendação Hierárquica de Produtos (SRHP), criado para atender às necessidades de organização, busca eficiente e recomendação automática em um marketplace com milhares de itens cadastrados. O projeto foi motivado pela necessidade de estruturar categorias e subcategorias de forma escalável, garantindo desempenho mesmo com o crescimento contínuo da base de produtos. Para isso, foi utilizada uma árvore AVL como estrutura de dados principal, possibilitando inserções, remoções e buscas em tempo O(log n), além de permitir reorganização automática e navegação hierárquica. O sistema foi desenvolvido em Python 3.10+, seguindo boas práticas de modularização, uso de recursividade e aplicação de testes unitários. O relatório apresenta a justificativa das decisões arquitetônicas, a análise de complexidade, os módulos implementados e o impacto do balanceamento AVL no desempenho geral da aplicação.

Palavras-chave: Relatório. Árvore AVL. Python. Dados.

SUMÁRIO

1. INTRODUÇÃO

Com o avanço do comércio eletrônico e o aumento do volume de produtos disponíveis em marketplaces, torna-se essencial o uso de estruturas de dados eficientes para garantir uma experiência de busca rápida e precisa. Categorias mal estruturadas, excesso de registros e falta de organização hierárquica resultam em lentidão, dificuldade de navegação e perda de conversões.

Nesse contexto, o projeto Sistema de Recomendação Hierárquica de Produtos (SRHP) foi solicitado pela diretoria de tecnologia com o objetivo de propor uma solução robusta, escalável e tecnicamente sólida. A proposta foi desenvolver uma estrutura baseada em árvores AVL, capazes de manter o balanceamento automático e assegurar desempenho adequado mesmo em ambientes de grande escala. Paralelamente, o sistema inclui módulos de recomendação, análise de tempo de execução, impressão hierárquica e mecanismos de cadastro organizados.

Entre as opções que foram avaliadas pela equipe, como árvores não balanceadas e arrays, a Árvore AVL, que seria uma árvore binária de busca auto-balanceada, foi a escolhida por sua eficiência e por garantir a complexidade logarítmica O(log n) em todas as duas operações críticas, mantendo um balanceamento automático.

No projeto é estabelecido como meta a redução de pelo menos 90% no tempo de busca em relação à implementação atual, suportando conjuntos de mais de 100.000 produtos e mantendo o tempo de resposta inferior a 20 ms.

O desenvolvimento foi orientado por boas práticas de engenharia de software, arquitetura modular, uso de recursividade nas operações de navegação da árvore e execução de testes unitários. Este relatório reúne o raciocínio técnico da equipe, suas decisões de projeto e a fundamentação teórica das estruturas utilizadas.

1.1 OBJETIVOS

1.1.1 OBJETIVOS GERAIS

Criar um Sistema de Recomendação Hierárquica de Produtos (SRHP) capaz de organizar e sugerir produtos de um marketplace de forma eficiente e escalável, utilizando árvores e árvores AVL como base para o armazenamento hierárquico, com operações implementadas de forma recursiva para navegação, busca e recomendação de produtos.

1.1.2 OBJETIVOS ESPECÍFICOS

Estruturar uma árvore AVL para armazenar categorias e subcategorias de produtos de forma balanceada, garantindo inserções, remoções e buscas em O(log n).

Implementar operações recursivas para navegação da árvore, incluindo busca hierárquica, impressão da estrutura e recomendação automática de produtos.

Desenvolver o módulo de negócios para cadastro, gerenciamento e recuperação de produtos associados às categorias.

Criar o módulo de recomendação, capaz de sugerir itens relacionados com base na posição hierárquica e em consultas recorrentes.

Construir o módulo de análise, responsável por medir desempenho, tempos de execução e comparar complexidades teóricas e práticas.

Aplicar arquitetura modular em Python, seguindo padrões de legibilidade, organização de pastas, separação de camadas e reutilização de código.

Garantir a qualidade do software por meio de testes unitários com cobertura mínima de 80%, validação funcional e boas práticas de QA.

Produzir um relatório técnico completo, explicando escolhas arquitetônicas, justificativas de uso da AVL e o impacto da estrutura no desempenho do sistema.

2. ARQUITETURA DO SISTEMA

2.1 Visão Geral da Arquitetura

2.1.1. Tecnologias utilizadas

- Python 3
- Bibliotecas:
  - ➔ Para a criação Front-end utilizando Python foi implementado o Tkinter, que é uma biblioteca usada para a criação de interfaces gráficas de usuário.

2.2. Estrutura dos Módulos

Os módulos são divididos com base no padrão MVC (Model-View-Controller) com alguns adicionais. o Model foi o products.py, o View foi o front_end.py e o Controller foi o productController.py. Além desses, a classe da árvore AVL ficou no módulo srhp.py, junto com parte da lógica da recomendação de produtos e os códigos foram reunidos para execução no main.py e testados no benchmark.py.

3. RECURSIVIDADE

A recursividade nada mais é do que uma função que chama a si mesma dentro do código. Na recursão o problema maior do código é dividido em problemas menores, resolvendo as pequenas partes até chegar ao **caso base**.

Para que a função seja considerada uma função recursiva e funcione de maneira correta, deve-se ter duas partes de grande importância:

O caso Base: A condição que determina quando a função deve parar, retornando o resultado conhecido. Sem esse caso base, a função entraria em loop infinito, causando um erro de stack overflow.

Passo Recursivo: Etapa do código onde a função chama a si mesma, com seus problemas simplificados.

A recursão foi utilizada no código por causa das vantagens que ela oferece, especialmente em problemas de estrutura de dados naturalmente recursivos. Dentro dessas vantagens estão a simplicidade e clareza do código, divisão de problemas complexos e gerenciamento automático de estado.

3.1. Aplicações no Projeto

A inserção, remoção, busca e recomendação hierárquica da árvore AVL executam usando a recursividade.

Inserção: A função se chama nos sub-nós esquerdo ou direito até encontrar uma posição vaga. Após o retorno da chamada recursiva, ela gerencia o balanceamento automático.

Remoção: Recursivamente busca o nó a ser removido e dependendo do caso (0, 1 ou 2 filhos) realiza a operação de remover ele e, no reorganizar a árvore.

Busca: Recursivamente percorre a árvore até achar o dado que está sendo procurado.

Recomendação: Uma das funções que auxiliam à recomendação é a _get_produtos_recursivo, onde a subárvore com base no nó escolhido é recursivamente percorrida com objetivo de pegar os filhos e netos para recomendação.

Travessia: A fim de obter os nós em ordem alfabética ele percorre a árvore como ilustrado a seguir:

```
def travessia_em_ordem(self, no_atual):
    resultado = []
    if no_atual:
        resultado.extend(self.travessia_em_ordem(no_atual.esquerda))
        resultado.append(no_atual.dado)
        resultado.extend(self.travessia_em_ordem(no_atual.direita))
    return resultado
```

4. ANÁLISE DE COMPLEXIDADE

4.1. Complexidade Big O

A notação Big O é uma ferramenta padrão para descrever o comportamento de crescimento de um algoritmo, medindo seu tempo de execução ou uso de memória se comporta à medida que o tamanho da entrada (n) se aproxima de infinito.

4.1.1 Classes de Complexidade

Notação Big O	**Nomenclatura**	**Taxa de Crescimento (Eficiência)**
O(1)	Constante	Excelente. Tempo fixo, independente de (n)
O(log n)	Logarítmico	Ótimo, cresce muito lentamente
O(n)	Linear	Bom, cresce de acordo com o tamanho de (n)
O(n log n)	Linear Logarítmico	Médio, eficiente para algoritmo de ordenação
O(n²)	Quadrático	Ruim, escalona muito rapidamente com (n)
O(2^n)	Exponencial	Péssimo, cresce drasticamente, caso impossível para operações com (n) grande.

4.2. Complexidade da árvore AVL

4.2.1 Operações Básicas

OPERAÇÃO	COMPLEXIDADE TEMPORAL	COMPLEXIDADE ESPACIAL
INSERÇÃO	O(log n)	O(log n) - pilha recursiva
BUSCA	O(log n)	O(log n) - pilha recursiva
REMOÇÃO	O(log n)	O(log n) - pilha recursiva
TRAVESSIA	O(n)	O(n) - visita todos os nós
ROTAÇÃO	O(1)	O(1)

5. RESULTADOS

5.1 Ambiente de Teste

- **Sistema Operacional: Windows 11**
- **Linguagem: Python 3.10+**
- **IDE: Vscode**

5.1.1. Teste

```python
import time
from algorithms.productController import ProductController

def medir_performance_avl(controller, num_itens=50000):
    print(f"\nTeste de Performance da Inserção AVL ({num_itens} Categorias)---")

    arvore_teste = controller.arvore_categorias
    dados = [f"Categoria_{i:05d}" for i in range(num_itens)]

    inicio = time.time()
    for dado in dados:
        arvore_teste.raiz = arvore_teste._inserir_recursivo(arvore_teste.raiz, dado)
    fim = time.time()

    tempo = fim - inicio
    print(f"\nTempo total de inserção AVL para {num_itens} itens: {tempo:.4f} segundos.")
    print("Complexidade: O(log n)")

    return tempo

if __name__ == "__main__":
    app_controller = ProductController()
    medir_performance_avl(app_controller)
```

5.2. Teste de Performance

O benchmark utilizado foi tempo de inserção, mas como busca, inserção e remoção compartilham a mesma notação Big O, é possível analisar o crescimento de todos com base nesse teste. Aqui estão os resultados de um crescimento do número de categorias dobrando:

50 000 Categorias: 0.1835s

100 000 Categorias 0.3819

200 000 Categorias 0.8305

400 000 Categorias 1.8049

800 000 Categorias 3.8694

1 600 000 Categorias 8.3241

Baseado nisso é possível afirmar que o fator de crescimento varia entre 2.08 e 2.17, o que informa que está nas expectativas da anotação O log n.

6. CONCLUSÃO E REFLEXÃO

6.1. O que foi alcançado

Todos os objetivos foram alcançados, o sistema é capaz de recomendar produtos por meio de uma árvore AVL eficientemente (O log n) que se auto balanceia em toda inserção e exclusão. Além disso, o código em si foi modularizado por meio do MVC (Model-View-Controller) e a recursividade foi usada extensivamente.

6.1.1 Lições Aprendidas

Diversas lições foram aprendidas ao decorrer do projeto, principalmente em relação à MVC, árvores e CRUD. Especialmente sobre balanceamento e rotações de árvores AVL.

6.1.2. Dificuldades

A principal dificuldade foi, sem dúvidas, a implementação da árvore AVL no arquivo srph.py. Onde coisas como a rotação (LL,RR,LR,RL) entre outras foram obstáculos consideráveis. Além de especificamente a árvore, a integração do controller também foi consideravelmente complexa na parte de recomendação de produto.

6.1.3. Impacto no mercado

Na hipótese de uso real no mercado, o sistema é eficiente o suficiente para lidar com milhões de categorias diferentes, algo comparativamente muito maior do que seria necessário para lidar com milhões de produtos, pois muitos fariam parte das mesmas categorias. Dessa forma escalabilidade e eficiência para uso no mercado são garantidos.