Implementação de Árvore B+ em C

Aluno: Gabriel Baeta Pereira

1. Visão Geral do Projeto

Este projeto consiste em uma implementação completa de uma estrutura de dados de Árvore B+ em C, projetada para gerenciar de forma eficiente um grande volume de registros em memória secundária. A estrutura é composta pelas operações fundamentais de inserção, busca e remoção, mantendo a árvore balanceada e garantindo a performance das Árvores B+.

A implementação foi modularizada em três arquivos (TARVB.h, TARVBP.c, main.c).

2. Decisões de Design e Arquitetura

2.1. Estruturas de Dados

A fundação do projeto reside em duas structs principais definidas em functions.h:

- Dados: Modela o registro apresentado. A escolha por arrays de char de tamanho fixo (cpf[13], nome[50]) é uma abordagem direta para o escopo do trabalho, evitando a complexidade do gerenciamento dinâmico de memória para cada registro.
- TARVBP: Define a estrutura de um nó da árvore:
    - eh_folha: Um flag crucial que permite à mesma struct representar tanto nós internos quanto nós folha.
    - ponteiro_pai: A inclusão de um ponteiro para o nó pai foi fundamental para a implementação da remoção. Sem esse ponteiro, a navegação "para cima" na árvore para operações de balanceamento seria extremamente complexa, exigindo recursão ou estruturas de dados auxiliares, o que simplifica drasticamente o algoritmo de remoção.
    - proximo_no: Este campo, usado apenas nas folhas, cria uma lista encadeada que conecta todas as folhas em sequência.
    - endereco: Armazena o endereço do nó no arquivo de índice, servindo como sua ID.

2.2. Gerenciamento de Arquivos

- **Arquitetura de Dois Arquivos**: Foi adotada uma separação clássica entre o índice e os dados:
    - ArqIdx.bin: Armazena apenas os nós da TARVBP.
    - ArqDados.bin: Contém os registros de Dados completos.
- **Endereçamento de Nós com Contador Global**: Variável global nos_atual para calcular o endereço de novos nós (nos_atual * sizeof(TARVBP)) é uma estratégia simples e eficaz para este projeto.

3. Análise dos Algoritmos Implementados

3.1. Inserção (TARVBP_insere)

A inserção utiliza a estratégia de divisão. Ao descer na árvore, o algoritmo verifica se o próximo nó a ser visitado está cheio. Se estiver, ele é dividido **antes** da descida. Esta abordagem garante que o nó pai sempre terá espaço para uma nova chave.

3.2. Busca (busca_cpf)

Começando na raiz e descendo nível por nível. As chaves nos nós internos guiam até chegar em uma folha. Ao chegarmos na folha, fazemos uma busca simples para encontrar o que queremos ou retornamos -1 se não acharmos.

3.3. Remoção (TARVBP_remove)

A remoção é a operação mais complexa e implementa os seguintes passos:

1. **Localização:** A chave é localizada no nó folha apropriado.
2. **Remoção Simples:** A chave é removida do nó folha.
3. **Verificação de Underflow:** O algoritmo verifica se o nó ficou com menos chaves que o mínimo permitido (TAM - 1).
4. **Balanceamento (balancear_no_apos_remocao):** Se houver underflow, a estratégia de **concatenação** é acionada (priorizando o esquerdo).

3.4. Testes (main.c)

A função main foi projetada não apenas como um ponto de entrada, mas como um **ambiente de testes robusto**. A metodologia de primeiro gerar um arquivo com 10.000 registros, construir a Árvore B+ a partir dele. Além disso, o sistema permite que o usuário teste as funcionalidades de inserir, buscar e remover alunos por CPF em tempo real.

https://github.com/GabrielBaetaP/Trab_EDA/tree/main/TRAB_EDA_BP