João Felipe Veras Dantas Rocha / DRE:117202753
Lucas dos Santos Melo / DRE:123312281
Rafael Augusto Santos / DRE:124155331
Rayan Lucas de Paula Carvalhal / DRE:124180742
Link para o GitHub dos arquivos com os códigos, assim como o arquivo de texto: https://github.com/jfvdrocha/BD-2025.1-ABB

ABB e índices de SBD

Professor: Milton Ramos Ramirez

Rio de Janeiro

2025

Comentários sobre o Desenvolvimento da Solução com ABB e EDL

O projeto proposto consistiu no desenvolvimento de um sistema de indexação e manipulação de registros baseado em Árvore Binária de Busca (ABB) para otimização de buscas por CPF, simulando um mecanismo de índices primários, conforme estudado na disciplina de Sistemas de Banco de Dados (SBD). A estrutura foi complementada por uma Estrutura de Dados Linear (EDL) (lista) para armazenar os registros originais, possibilitando integração entre busca eficiente e acesso direto aos dados.

Estrutura de dados utilizadas:

- Lista (EDL):
  -Armazena os registros na ordem de inserção.
  -Cada registro possui um campo deletado para controle lógico de remoções.
  -Complexidade de acesso: O(1) por índice.

- Árvore Binária de Busca (ABB):
  -Utilizada para indexar os registros da EDL por CPF, otimizando operações de busca.
  -Cada nó armazena um objeto da classe Registro, a posição correspondente na EDL e ponteiros para os filhos esquerdo e direito.

Complexidades:

-Inserção, busca e remoção: O(h), onde h é a altura da árvore (O(log n) em caso ideal, O(n) no pior caso se desbalanceada).
-Percursos (pré-ordem, in-ordem, pós-ordem e em largura): O(n) no tempo e O(h) no espaço de pilha ou fila auxiliar.

Organização de Módulos e Rotinas

- Classe Registro: encapsula os dados de cada pessoa, com métodos de comparação baseados no CPF.
- Classe NoABB: representa os nós da árvore, armazenando a chave e a posição na EDL.
- Classe ABB: provê métodos para inserção, remoção, busca e diferentes percursos. Também implementa a cópia profunda da árvore e sua deleção completa.
- Função consultar_por_cpf(): realiza a consulta de um CPF indexado, retornando os dados diretamente da EDL.
- Função gerar_edl_ordenada(): utiliza percurso in-ordem para reordenar os dados conforme a ordenação da ABB.

As funcionalidades foram organizadas em módulos com clara separação entre dados (EDL), índice (ABB) e interface de consulta.

Complexidade e Eficiência

A principal vantagem da ABB é garantir eficiência em consultas, desde que a árvore esteja balanceada.

Por não usar AVL ou Red-Black Trees, o sistema pode atingir degradação para O(n) em casos patológicos (ex: inserções em ordem crescente).

A escolha de manter os registros fora da árvore, apenas com índices armazenados nos nós, reduz o uso de memória duplicada e evita inconsistências.

Problemas e Observações

A ausência de balanceamento automático pode gerar árvores desbalanceadas com degradação de desempenho.

O campo deletado na EDL serve para simular deleções lógicas — o ideal seria realizar um "garbage collection" ao final.

A integração entre EDL e ABB exige atenção ao manter os índices sincronizados após remoções.

Conclusão:

A implementação cumpriu todos os requisitos propostos, permitindo acesso eficiente aos registros por CPF, além de suportar inserções, remoções, buscas e ordenação. A integração entre ABB e EDL mostrou-se eficaz e didática, evidenciando os conceitos de índices primários e estruturas hierárquicas.

Apesar de a ABB não garantir balanceamento, a estrutura foi suficiente para ilustrar a importância da separação entre dados e índices em um sistema gerenciador de dados. A atividade também reforçou a compreensão prática sobre operações em árvore binária, percursos e manipulação de registros reais.