Relatório do Desenvolvimento da Árvore Binária de Busca (ABB) para Gerenciamento de Registros

Trabalho de Estrutura de Dados

Universidade Federal do Rio de Janeiro

Charles RIbeiro Chaves - 122086950
Filipe Viana da Silva - 121050053
Vinícius Brasil de Oliveira Barreto - 120029237

6 de julho de 2025

Sumário

1 Introdução

Este relatório descreve o desenvolvimento de uma estrutura de dados hierárquica do tipo Árvore Binária de Busca (ABB) para gerenciar registros de pessoas, contendo campos como CPF, nome e data de nascimento. O objetivo foi implementar a ABB como índice para um arquivo de registros linear, permitindo buscas eficientes e operações de inserção e remoção.

2 Estruturas de Dados Utilizadas

- **Registro:** classe que encapsula os dados de uma pessoa, incluindo CPF (chave de ordenação), nome e data de nascimento, além de uma flag para marcação de exclusão lógica.
- **Nó da ABB:** cada nó contém um objeto `Registro` e a posição correspondente no arquivo linear, além de referências para os filhos esquerdo e direito.
- **Árvore Binária de Busca (ABB):** estrutura hierárquica que mantém os nós ordenados pela chave CPF, suportando inserção, remoção, busca e diversos percursos.
- **Arquivo de Registros (EDL linear):** lista simples que armazena os registros, permitindo acesso por índice em tempo constante.

3 Divisão de Módulos

O código foi organizado em três módulos principais:

- `registro.py`: definição da classe Registro, incluindo operadores de comparação e métodos auxiliares.
- `arvore.py`: implementação da ABB, com métodos para inserção, remoção, busca e percursos (pré-ordem, em ordem, pós-ordem e largura).
- `sistema.py`: implementação do sistema gerenciador que integra o arquivo de registros e a ABB como índice, além das operações de inserção, remoção e busca.

4 Descrição das Rotinas e Funções

- `Registro.__lt__`: permite comparar registros pela chave CPF para manter a ABB ordenada.
- `ArvoreBinariaBusca.inserir`: insere um novo nó na ABB, posicionando-o conforme a chave.
- `ArvoreBinariaBusca.remover`: remove um nó da ABB, tratando os casos de nó folha, com um filho ou dois filhos.
- `ArvoreBinariaBusca.buscar`: busca um nó pela chave CPF.

- `ArvoreBinariaBusca.percursos`: métodos para percorrer a ABB em pré-ordem, em ordem, pós-ordem e largura.
- `ArquivoRegistros.inserir`: adiciona um registro ao arquivo linear em tempo constante.
- `ArquivoRegistros.deletar`: marca um registro como deletado sem alterar a posição dos demais.
- `SistemaGerenciadorBD`: coordena as operações entre ABB e arquivo linear, garantindo consistência.

5 Complexidade de Tempo e Espaço

- **Inserção e busca na ABB:** em média, tempo $O(\log n)$, onde n é o número de registros, devido à propriedade da árvore binária balanceada (não implementado balanceamento automático, portanto no pior caso pode ser $O(n)$).
- **Remoção na ABB:** também em média $O(\log n)$, com complexidade adicional para reorganizar a árvore.
- **Acesso ao arquivo linear:** busca por índice em tempo constante $O(1)$.
- **Espaço:** armazenamento linear para os registros e espaço proporcional ao número de nós na ABB.

6 Problemas e Observações

Durante o desenvolvimento, destacam-se:

- A necessidade de tratar registros deletados para evitar erros na impressão, especialmente ao lidar com datas nulas.
- A ausência de balanceamento automático na ABB pode levar a degradação de desempenho para conjuntos de dados ordenados.
- A remoção lógica no arquivo linear simplifica a manutenção da posição dos registros, mas pode causar fragmentação e desperdício de espaço.
- A implementação do percurso em largura foi um desafio adicional, mas permitiu maior flexibilidade na manipulação da ABB.

7 Conclusão

A implementação da Árvore Binária de Busca como índice para um arquivo linear de registros demonstrou ganhos significativos em eficiência de busca, reduzindo a complexidade de $O(n)$ para $O(\log n)$ em condições médias. A separação clara entre as estruturas de dados e o sistema gerenciador facilitou a manutenção e extensibilidade do código. Apesar das limitações, como a falta de balanceamento, a solução atende aos requisitos propostos e serve como base para aprimoramentos futuros, como a implementação de árvores balanceadas ou múltiplos índices para diferentes campos.