Relatório do Estudo Dirigido de Teoria de Grafos

Aluno: Susan Kaori Izawa

**1. Introdução**

Este trabalho teve como objetivo implementar um **sistema de gerenciamento de arquivos** utilizando **árvores AVL** para garantir eficiência nas operações de busca, inserção e remoção.

Cada arquivo no sistema possui:

* **Identificador único (id)** numérico
* **Nome** (string)
* **Extensão** (string)
* **Tamanho** (inteiro não negativo)

As operações principais são:

* Inserir arquivo na árvore
* Buscar arquivo pelo identificador (retornando se existe ou não)
* Remover arquivo pelo identificador
* Manter a árvore balanceada conforme as regras da AVL

Além da lógica, foi desenvolvida uma **interface gráfica** em C++ utilizando a biblioteca **raylib**, permitindo interação por meio de um menu e diferentes telas.

**2. Desenvolvimento**

**2.1 Implementação da árvore AVL**

Inicialmente, o foco foi a implementação de uma árvore binária de busca simples, para depois estender com o balanceamento da AVL.

* Foi criada uma estrutura de nó contendo o identificador e ponteiros para filhos, além de um ponteiro para o **pai** (necessário nas rotações).
* No começo, a função getHeight calculava a altura dinamicamente. Depois, para eficiência, a altura passou a ser armazenada diretamente no nó.
* A função de balanceamento foi implementada de forma recursiva, usando **percurso pós-ordem** para balancear de baixo para cima.
* Durante a implementação das rotações, percebeu-se a necessidade de atualizar corretamente as conexões entre nós e seus pais.
* Foram implementadas funções para imprimir a árvore em **pré-ordem, em-ordem, pós-ordem e por nível**. Isso ajudou nos testes de corretude.
* Além das versões recursivas, foram também implementadas **versões não recursivas** de funções de busca, inserção e remoção, com o objetivo de aprendizado.

**2.2 Estrutura de dados do arquivo**

Foi criado um struct File contendo os dados de cada arquivo, separado da lógica da árvore AVL. Assim, cada nó armazena um id (chave de busca) e um ponteiro para a estrutura de arquivo. Essa decisão permitiu reutilizar a árvore para diferentes contextos, mantendo o código modular.

**2.3 Organização do código**

O projeto foi dividido em múltiplos arquivos .cpp e .h, cada um agrupando funcionalidades semelhantes (ex.: bst.cpp para a AVL, stack.cpp para pilha, etc.). Essa separação deixou o código mais legível e fácil de manter, funcionando como pequenas bibliotecas internas.

O código foi escrito em inglês para seguir padrões globais, embora isso tenha exigido refatoração ao integrar com código do professor.

**3. Interface gráfica com Raylib**

**3.1 Motivações**

Como imprimir árvores no console tem limitações (principalmente na visualização de arestas), optou-se por utilizar a biblioteca **raylib**.

**3.2 Estrutura das telas**

Foi adotada uma máquina de estados simples (enum SystemScreen) para alternar entre telas:

* Menu principal
* Armazenar arquivo
* Buscar arquivo
* Remover arquivo
* Mostrar árvore

Cada tela segue um padrão de **initScreen**, **handleScreen** e **drawScreen**, para separar inicialização, lógica e desenho.

**3.3 Implementação**

* Foram criados botões com retângulos (Rectangle) e cores diferentes dependendo do estado (normal, selecionado, hover).
* Inputs de texto foram implementados do zero, exigindo que o usuário mantenha o mouse sobre a caixa para digitar.
* Foi priorizada a **funcionalidade** em vez da sofisticação (não foi implementado foco de input nem botões prontos de confirmação).
* Foram criadas **telas de erro e confirmação** para dar feedback ao usuário.
* A função que desenha os nós da árvore usa percurso **pós-ordem**, garantindo que as arestas sejam desenhadas antes dos nós.

**4. Dificuldades encontradas**

* Integração com o código do professor levou a refatoração total.
* Implementar inputs de texto foi mais trabalhoso do que esperado.
* A falta de componentes prontos no raylib (botões, caixas de texto, popups) fez com que grande parte da interface tivesse que ser criada manualmente.
* Passar informações entre telas (ex.: arquivo encontrado na busca) se mostrou um desafio, exigindo variáveis globais ou armazenamento temporário.

**5. Configuração do ambiente**

Para rodar o projeto é necessário:

1. Instalar o compilador **mingw** incluso no raylib (w64devkit).
2. Instalar o **raylib** e adicioná-lo ao PATH.
3. (Opcional) Usar Code::Blocks para configurar o projeto.
4. Incluir todos os .cpp e .h no projeto.
5. Configurar o compilador global para apontar para o raylib.
6. Linkar a biblioteca libraylib.a no projeto.
7. Adicionar as opções de link: -lraylib -lgdi32 -lopengl32 -lwinmm.
8. Compilar e executar.

**6. Conclusão**

O projeto alcançou os objetivos principais de implementar uma **AVL funcional** e integrá-la a um **sistema de gerenciamento de arquivos** com interface gráfica.

Como pontos fortes, destacam-se:

* Estrutura modular e reutilizável da AVL
* Separação clara entre lógica e interface gráfica
* Implementação tanto recursiva quanto não recursiva de funções, reforçando o aprendizado

Como pontos a melhorar:

* Melhor organização da passagem de dados entre telas
* Uso de bibliotecas gráficas mais completas para inputs e botões
* Interface mais polida e intuitiva para o usuário

O trabalho foi uma boa oportunidade para consolidar conhecimentos de **estruturas de dados, C++ e desenvolvimento de interfaces gráficas**.