ATIVIDADE PRÁTICA 2

Valor: 1,0

Especificações:

- Esse trabalho pode ser realizado em duplas **ou** individual;
- O valor total do trabalho representará 10% da nota do semestre;
- O trabalho deve ser entregue em formato digital no Moodle (data da entrega: 03/02/2025 – 23:59 Moodle)
- Arquivos .c compactados.
- Não serão aceitos, em hipótese alguma, trabalhos enviados via e-mail.
- Cópias de trabalhos serão penalizadas com a perda total do valor do trabalho.
- O trabalho enviado deve estar claramente identificado, com nome do aluno.

Sistema de Gerenciamento de Reservas de Salas

1 Introdução

Gerenciar reservas de salas de maneira eficiente é essencial para evitar conflitos de horários e garantir uma melhor utilização dos recursos disponíveis. Estruturas de dados como AVL, Heap e Hash permitem que operações de reserva, cancelamento e consulta sejam realizadas rapidamente, otimizando a experiência dos usuários.

Este projeto consiste em desenvolver um sistema de gerenciamento de reservas de salas que permite aos usuários criar, cancelar e consultar reservas, bem como priorizar reservas baseadas em critérios específicos. O sistema deverá realizar todas as operações necessárias lendo e manipulando dados fornecidos em arquivos .txt.

2 Objetivos

Implementar as seguintes estruturas de dados para o sistema de gerenciamento de reservas de salas:

1. **Árvore AVL** para gerenciar informações das salas.
2. **Heap** para gerenciar prioridades de reservas.
3. **Tabela Hash** para armazenar estados das salas.

O sistema deve:

- Ler as informações de entrada de arquivos .txt.
- Processar e executar as operações especificadas nos arquivos.
- Gerar arquivos de saída .txt contendo logs detalhados das ações realizadas.

2.1 Entrada e Saída de Dados

1. **Arquivos de Entrada (entrada.txt)**:
   O arquivo de entrada conterá uma lista de operações e dados a serem processados. Exemplos de operações incluem:
   - CRIAR_SALA <código> <capacidade>
   - RESERVAR_SALA <código_sala> <prioridade>
   - BLOQUEAR_SALA <código_sala>
   - DESBLOQUEAR_SALA <código_sala>
   - CANCELAR_RESERVA <código_sala>
   - CONSULTAR_SALA <código_sala>
   - LISTAR_SALAS

2.2 Arquivos de Saída (saida.txt):

- O arquivo de saída deverá conter os resultados das operações executadas, como mensagens de sucesso ou falha, e o estado final das estruturas de dados após cada operação.

2.3 Funcionalidades

2.3.1 Gestão de Salas (Árvore AVL)

- **Inserir sala**: Adicionar informações de salas (código e capacidade).
- **Remover sala**: Excluir uma sala da estrutura AVL.
- **Buscar sala**: Localizar uma sala específica pelo código.
- **Listar salas**: Exibir todas as salas disponíveis em ordem crescente de código.

Quando uma sala vai para a AVL?

A Árvore AVL é utilizada para organizar e armazenar as informações das salas de forma balanceada, permitindo busca, inserção e remoção eficientes.

- Entrada na AVL:
    - Quando uma sala é criada (CRIAR_SALA), suas informações (código, capacidade) são inseridas na Árvore AVL.
    - Exemplo: A operação CRIAR_SALA 101 50 adiciona a sala 101 com capacidade de 50 lugares à AVL.
- Operações típicas na AVL:
    - Inserção: Adiciona novas salas.
    - Remoção: Exclui salas existentes do sistema (REMOVER_SALA).
    - Busca: Permite localizar uma sala pelo seu código.
    - Listagem: Exibe as salas em ordem crescente do código.
- Propósito da AVL:
    - Organizar as salas de forma estruturada e balanceada.
    - Permitir buscas e listagens rápidas e eficientes.

2.3.2 Gestão de Prioridades (Heap)

- **Adicionar reserva**: Inserir uma reserva na heap de prioridades.
- **Atualizar prioridade**: Alterar a prioridade de uma reserva existente.
- **Listar reservas**: Exibir todas as reservas em ordem de prioridade.

Quando uma sala vai para a Heap?

A Heap é utilizada para gerenciar as reservas das salas, ordenando-as com base em prioridades.

- Entrada na Heap:
    - Quando uma sala é reservada com uma prioridade específica, essa reserva é adicionada à Heap de Máximo.
    - Exemplo: A operação RESERVAR_SALA 101 10 adiciona uma reserva da sala 101 com prioridade 10 à Heap.
- Propósito da Heap:
    - Permitir que as reservas sejam organizadas por prioridade, garantindo que as mais urgentes ou importantes sejam atendidas primeiro.

- A Heap é utilizada apenas enquanto houver reservas ativas associadas às salas.

2.3.3 Gestão de Estados (Tabela Hash)

- **Bloquear sala**: Alterar o estado de uma sala para "bloqueada".
- **Desbloquear sala**: Alterar o estado de uma sala para "disponível".
- **Consultar estado**: Listar todas as salas por estado atual.

Quando uma sala vai para a Hash?

A Hash é utilizada para armazenar e gerenciar os estados das salas.

- Entrada na Hash:
  - Sempre que uma sala é criada (CRIAR_SALA), seu estado inicial (por exemplo, "disponível") é registrado na Tabela Hash.
  - Quando o estado de uma sala é alterado, como em operações BLOQUEAR_SALA ou DESBLOQUEAR_SALA, a Tabela Hash é atualizada.
- Propósito da Hash:
  - Permitir consultas rápidas ao estado atual das salas (disponível, bloqueada, etc.).
  - Facilitar a listagem e categorização de salas por estado.

2.3.4 Cancelar Reserva

Remover uma reserva específica de todas as estruturas de dados.

2.3.5 Encerrar Sistema

Remover todas as informações de salas e reservas antes de finalizar o sistema.

3 Estrutura de Dados

3.1 Árvore AVL para Salas

Implementar operações de inserção, remoção e busca que garantam o balanceamento da árvore, otimizando consultas e modificações.

3.2 Heap para Prioridades

Implementar um heap de máximo para gerenciar prioridades de reservas, garantindo que as mais importantes sejam atendidas primeiro.

3.3 Tabela Hash para Estados

Implementar uma tabela hash eficiente para armazenar estados das salas e realizar consultas rápidas. Incluir tratamento de colisões, como encadeamento ou endereçamento aberto.

4. Requisitos Técnicos

1. Implementação em C.
2. Entrada e saída baseadas em arquivos .txt.
3. Evitar ao máximo o uso de variáveis globais.
4. Implementar funções separadas para cada operação.
5. Gerar logs detalhados das ações realizadas em arquivos de saída (saida.txt).
6. Documentar adequadamente o código-fonte, descrevendo a funcionalidade de cada módulo, função e estrutura de dados.