Trabalho Prático: Estrutura de Dados II

Sistema de Análise de Perfis de Jogadores 2025

Alunos:
Diego Pereira Betti - 202376037 - diego.pereira@estudante.ufjf.br
Vanessa Palmeira Kelmer Trajano - 202376035 - vanessa.trajano@estudante.ufjf.br
Eduarda Pereira Mourão Nunes - 202376015 - eduarda.nunes@estudante.ufjf.br

1. Descrição da Implementação

- O projeto foi desenvolvido em C++ utilizando as bibliotecas padrão da linguagem, com organização modular em diferentes diretórios para facilitar a manutenção e a escalabilidade. A estrutura do projeto segue o padrão:

- **src**: Contém os arquivos-fonte organizados em subpastas por funcionalidade (ex: Árvores B, Hash, Leitor de Planilha, Menu, Objetos, Utils).
- **steam**: Armazena os arquivos de dados em formato CSV, essenciais para o funcionamento do sistema.
- **build**: Diretório destinado aos arquivos gerados durante a compilação.

- O sistema realiza a leitura de arquivos CSV contendo informações sobre jogadores, jogos, conquistas, histórico de partidas, preços e compras. Para manipulação eficiente dos dados, foram implementadas estruturas como Árvores B e Tabelas Hash, além de listas encadeadas para gerenciamento interno.

- A compilação é feita via CMake, com suporte ao MinGW no Windows, conforme instruções detalhadas no README.

2. Decisões de Projeto

- **Estrutura Modular**: O código foi dividido em módulos para separar responsabilidades, facilitando testes e futuras expansões.

- **Uso de Árvores B**: Escolhidas para indexação eficiente de dados que exigem buscas rápidas e ordenadas, como histórico de partidas e conquistas.

- **Tabelas Hash**: Utilizadas para acesso rápido a informações de jogadores, reduzindo a complexidade de busca.

- **Leitura de CSV**: Implementação de um módulo dedicado para leitura e parsing dos arquivos de dados, garantindo flexibilidade para diferentes formatos.

- **Compatibilidade**: O projeto foi pensado para ser facilmente compilado em ambientes Windows, utilizando ferramentas amplamente disponíveis (CMake e MinGW).

3. Documentação do Código

Documentação do Sistema de Gerenciamento de Jogadores, Jogos e Conquistas

Visão Geral
O sistema gerencia jogadores, jogos e conquistas, permitindo consultas e inserções eficientes usando tabelas hash e árvores B. Os dados são lidos de arquivos CSV e organizados em estruturas otimizadas para buscas e estatísticas.

Estruturas de Dados e Utilitários
Utils (utils.h)
split(str, delimitador): Divide uma string em tokens, removendo espaços extras.
verificaIdJogador(s): Verifica se a string é um ID válido de jogador (17 dígitos).

3.1 Player

Player (Player.h, Player.cpp)
Representa um jogador.

Atributos:
- id, pais, dataDeCriacao
- jogos: lista de ponteiros para jogos adquiridos
- conquistas: lista de conquistas obtidas

Principais métodos:
- getId(), getPais(), getDataDeCriacao()
- getJogos(), getConquistas()
- addConquista(conquista): adiciona conquista ao jogador

- setJogos(jogos): define os jogos do jogador
- operator<<: imprime informações do jogador

3.2 Jogo

Jogo (Jogo.h, Jogo.cpp)
Representa um jogo.

Atributos:
- id, titulo, desenvolvedores, publishers, generos, idiomas, dataDeLancamento
- conquistas: lista de conquistas do jogo

Principais métodos:
- getId(), getTitulo(), getDesenvolvedores(), getPublishers(), getGeneros(), getIdiomas(), getDataDeLancamento()
- getConquistas()
- adicionaConquista(conquista): adiciona conquista ao jogo
- operator<<: imprime título do jogo

3.3 Conquista

Conquista (Conquista.h, Conquista.cpp)
Representa uma conquista de um jogo.

Atributos:
id, idJogo, titulo, descricao

Principais métodos:
- getId(), getIdJogo(), getTitulo(), getDescricao()
- operator<<: imprime título da conquista

3.4 Tabela Hash

Estruturas Genéricas
Enum MetodoDeColisao (hash_generico.h)
Define o método de tratamento de colisão na tabela hash:
ENCADEAMENTO: usa listas encadeadas
SONDAGEM_LINEAR: usa sondagem linear

Tabela Hash<T> (hash_generico.h)
Tabela hash genérica para qualquer tipo de objeto.

Atributos:

- tamanho, numeroDeElementos, numeroDeColisoes
- metodoDeColisao: define o método de colisão
- encadeamento: vetor de listas encadeadas (para encadeamento)
- sondagemLinear: vetor de entradas (para sondagem linear)

Principais métodos:

- insere(obj): insere objeto na hash
- busca(id): busca objeto pelo ID
- exibirEstatisticas(): mostra estatísticas da hash
- forEach(func): aplica função a todos os elementos

3.5 Hash Entry

HashEntry<T> e Enum EntryState (hash_entry.h)
HashEntry: Estrutura para armazenar dados e estado de uma posição na hash (vazio, ocupado, removido).

3.6 Linked List

LinkedList<T> (linked_list.h)
Lista encadeada genérica, usada para encadeamento em hash.

Principais métodos:

- insere(data): insere no início
- busca(id): busca elemento pelo ID
- forEach(func): aplica função a todos os elementos

JogadoresDoJogo (hash_players.h)
Struct que associa um jogo a uma lista de IDs de jogadores que o possuem.

Principais métodos:

- getId(): retorna o ID do jogo
- addJogador(id): adiciona ID de jogador
- imprimeJogadores(): imprime todos os jogadores do jogo

3.7 Hash de Players

HashPlayers (hash_players.h)
Classe utilitária para popular tabelas hash com jogadores, jogos e conquistas.

Principais métodos:
populaTabelaComPlayers(tabela, metodo): carrega jogadores na hash

criaHashJogosParaJogadores(hashJogos, tabelaPlayers): cria hash de jogos para jogadores

3.8 Árvore B

Estruturas de Árvore B
Enum TipoDeIndexacao (arvoreB.h)
Define o tipo de indexação da árvore B:

JOGOS: indexa jogadores por quantidade de jogos
CONQUISTAS: indexa jogadores por quantidade de conquistas

ArvoreB (arvoreB.h, arvoreB.cpp)
Árvore B para indexação eficiente de jogadores.

Atributos:

- ordem: ordem da árvore
- raiz: nó raiz

Principais métodos:

- busca(chave): busca jogador pela chave (ex: número de jogos)
- insere(chave, jogador): insere jogador na árvore
- indexarPorJogos(tabelaJogadores): indexa jogadores por jogos
- indexarPorConquistas(tabelaJogadores): indexa jogadores por conquistas
- buscaTopJogadores(qtd): retorna os jogadores com mais jogos/conquistas
- buscaPorIntervalo(min, max): retorna jogadores em um intervalo de chaves

3.9 Nó B

NoB (noB.h)
Nó da árvore B.

Atributos:

- chaves: valores de indexação
- filhos: ponteiros para filhos
- jogadores: ponteiros para jogadores
- chavesPreenchidas, eFolha: controle do nó

Principais métodos:

- addChave(chave, jogador): adiciona chave e jogador
- addFilho(filho): adiciona filho
- setEFolha(eFolha): define se é folha

3.10 Leitura de Dados

Leitura de Dados
LeitorDePlanilha (leitorDePlanilha.h, leitorDePlanilha.cpp)
Classe utilitária para ler e processar arquivos CSV.

Principais métodos:

- limparCampoCSV(campo): limpa campo de CSV
- contadorCSV(caminho): conta linhas de um CSV
- processarCSV(caminho, func): processa cada linha do CSV com uma função callback

3.11 Menu

Menu e Fluxo Principal
Menu (menu.h, menu.cpp)
Gerencia a interação com o usuário e integra todas as estruturas.

Atributos:

- tabelaHash: hash de jogadores
- hashJogos: hash de jogos para jogadores
- arvoreBJogos, arvoreBConquistas: árvores B para indexação

Principais métodos:

- menuInicial(): inicia o menu principal
- menuDeConsultas(): menu de consultas (buscas, estatísticas)
- menuBuscaHash(), menuInsercaoHash(): menus de busca e inserção
- imprimeTopJogadores(qtd, tipo): imprime top jogadores por jogos/conquistas
- imprimeIntervaloDeJogadores(min, max, tipo): imprime jogadores em intervalo
- imprimeEstatisticasJogos(): estatísticas dos jogos
- imprimeTopJogos(qtd): imprime top jogos
- imprimeJogadoresDoJogo(id): imprime jogadores de um jogo
- imprimeJogosOuConquistasDoJogador(id, tipo): imprime jogos ou conquistas de um jogador

3.12 Fluxo de Inicialização

- Leitura dos CSVs: Utiliza LeitorDePlanilha para carregar dados.
- Popular Hashes: Usa HashPlayers para preencher as tabelas hash.
- Indexação em Árvore B: Indexa jogadores por jogos/conquistas.
- Menu: Usuário interage via Menu, realizando buscas, inserções e consultas.
- Integração

- As classes de objetos (Player, Jogo, Conquista) são usadas nas estruturas genéricas (TabelaHash, ArvoreB).
- O menu centraliza a lógica de interação, utilizando as funções das estruturas para responder às ações do usuário.
- Utilitários (Utils, LeitorDePlanilha) facilitam o processamento de dados e validações.

4. Análise de Desempenho

- **Leitura de Arquivos:** O sistema realiza a leitura sequencial dos arquivos CSV linha a linha ao iniciar, processando milhares de registros em poucos segundos em um computador padrão (ex: Intel i5, 8GB RAM, Windows 10). O tempo de carregamento é linear em relação ao tamanho dos arquivos. Sendo otimizado para criar os objetos de forma dinâmica a cada linha lida fazendo com que o consumo de memória seja menor.

- **Busca em Árvores B:** As Árvores B foram utilizadas para indexar dados que exigem ordenação e buscas rápidas, como conquistas e quantidade de jogos.

- **Acesso via Tabela Hash:** Para operações envolvendo jogadores, a tabela hash permite inserção e busca em tempo praticamente constante (O(1)), mesmo com grandes volumes de dados. O sistema manteve desempenho estável devido ao uso de listas encadeadas ou de sondagem linear, a escolha do usuário, para resolução de colisões.

- **Consumo de Memória**: O uso de estruturas dinâmicas garante que a memória seja alocada conforme a necessidade, fazendo toda consulta e acesso a objetos já criados por ponteiros.

- **Escalabilidade:** O sistema foi projetado para suportar o aumento do volume de dados sem degradação significativa do desempenho. A modularização permite a inclusão de novos módulos ou otimizações sem impacto negativo nas funcionalidades existentes.

5. Conclusões

O sistema desenvolvido atendeu aos requisitos propostos, apresentando desempenho eficiente e organização modular. A escolha das estruturas de dados foi

fundamental para garantir rapidez nas operações e flexibilidade para expansão futura.

Comparativo: Árvore B vs. Tabela Hash

- **Árvore B:**
  - Vantagens: Ideal para buscas ordenadas, intervalos de valores e operações que exigem ordenação (ex: listar conquistas ou histórico em ordem cronológica). Mantém desempenho estável mesmo com grandes volumes de dados e permite inserções e remoções eficientes.
  - Limitações: Pode ser menos eficiente que a tabela hash para buscas pontuais, pois a complexidade é O(log n).
  - Melhor uso: Quando é necessário manter os dados ordenados ou realizar buscas por intervalos.

- **Tabela Hash**:
  - Vantagens: Excelente para buscas diretas por chave (ex: buscar um jogador pelo ID), com complexidade média O(1). Inserções e buscas são extremamente rápidas, mesmo com grandes volumes de dados.
  - Limitações: Não mantém os dados ordenados e pode sofrer com colisões, que afetam o desempenho se não forem bem tratadas. Não é adequada para buscas por intervalos ou ordenação.
  - Melhor uso: Quando o acesso é feito por chave única e não há necessidade de ordenação.

- **Recomendações:**
  - Use a Árvore B para operações que envolvem ordenação, intervalos ou quando a estrutura precisa ser balanceada e escalável.
  - Use Tabela Hash para buscas rápidas por chave exata, como consultas diretas a registros específicos.
  - Evite a Tabela Hash quando for necessário percorrer os dados em ordem ou realizar buscas por intervalos.
  - Evite Árvore B quando o acesso for sempre direto e não houver necessidade de ordenação, pois a tabela hash será mais eficiente.

- **Considerações Finais**: A combinação dessas estruturas no projeto permitiu explorar o melhor de cada abordagem, garantindo desempenho e flexibilidade.