Trabalho Prático I

Implementação de Árvore B com Virtualização para Indexação de um Sistema de Locadora de Veículos

Disciplina: Estrutura de Dados II

Objetivo

Desenvolver um sistema de gerenciamento de veículos para uma locadora, utilizando uma Árvore B para indexação com virtualização. O sistema deve implementar uma fila de tamanho **P** para manter as **P** últimas páginas utilizadas em memória principal, otimizando o acesso aos dados e reduzindo a necessidade de carregamento de páginas do disco. O usuário poderá realizar operações de busca, inserção e remoção de veículos na árvore usando placa do veículo como chave primária.

Descrição do Trabalho

1. Implementação da Árvore B

- **Estrutura da Árvore B**:
  - A Árvore B deve ser de ordem **M** definida por `#define`. Desta forma, se esse valor mudar, a árvore deve ser reconstruída.
  - Cada página (nó) da árvore contém:
    - **Vetor de Chaves**: Placas dos veículos (*string*).
    - **Vetor de RNNs**: endereço de Registro no arquivo de dados correspondente a cada chave.
    - **Vetor de RNNs Ponteiros para os filhos** das chaves.
    - Outras informações pertinentes para implementação como número de chaves dentro da página e quem é a página pai.
  - Use registros de tamanho fixo para as páginas.
  - A árvore deve ser mantida em um arquivo binário `btree_M.idx` que armazenará no seu cabeçalho o RNN da raiz seguida das demais chaves.
  - Quando executado o código, a seguinte verificação deve ser feita
    - Se Árvore B com ordem M **existe** em arquivo (verificar se existe o arquivo `btree_M.idx`), carregue apenas a raiz na memória virtual (coloque na fila) e mostre as opções de operações ao usuário.
    - Se Árvore B com ordem M **não existe**, crie-a lendo o registros do arquivo de dados um a um e inserindo as suas chaves primárias na Árvore B. No processo de construção use a fila e salve as páginas no arquivo

`btree_M.idx`). Por fim, carregue a raiz em memória principal e mostre as opções de operações ao usuário.

2. Virtualização e Gerenciamento de Páginas

- **Fila de Páginas na Memória**:
  - Utilize uma **fila de tamanho P** para manter as páginas utilizadas na memória principal.
  - **Se uma página já está na fila**:
    - Não é necessário carregá-la do arquivo.
    - Mova a página para o final da fila (indicando acesso recente).
  - **Se uma página não está na fila**:
    - Carregue a página do arquivo binário de indexação `btree_M.idx`.
    - Adicione a página ao final da fila.
    - Se a fila atingiu o tamanho máximo **P**, remova (desaloque) a página do início da fila (página menos recentemente usada).
    - O valor de **P** deve ser definido com um `#define`.
    - Lembre-se que caso uma página seja alterada (split, inserção, remoção), ela precisa ser modificada no memória virtual (principal) e também em arquivo.

3. Operações Disponíveis

- **Busca de Veículo**:
  - Permitir que o usuário procure por um veículo específico na Árvore B, utilizando a placa como chave.
  - Através do RNN encontrado na Árvore B, carregar os dados do veículo do arquivo de dados e imprimir para o usuário as informações do veiculo.
  - Pontuação se funcionando corretamente: 5 pts
- **Inserção de Veículo**:
  - Inserir um novo veículo no sistema.
  - Atualizar a Árvore B e o arquivo de dados conforme necessário.
  - Tratar casos de divisão de páginas e ajuste de ponteiros na árvore.
  - Pontuação se funcionando corretamente: 3 pts
- **Remoção de Veículo**:
  - Remover um veículo existente no sistema.
  - Ajustar a estrutura da Árvore B conforme necessário (fusão ou redistribuição de páginas).
  - Atualizar o arquivo de dados para refletir a remoção. Para isso pode manter um arquivo extra com os RNNs de páginas removidas ou ajustar o cabeçalho do arquivo de dados. Essas páginas podem ser reutilizadas em novas inserções.
  - Pontuação funcionando corretamente: 2 pts

4. Arquivo de Dados e Chave Primária

- **Arquivo de Dados** (`veiculos.dat`):
  - Arquivo binário que contém registros de veículos disponíveis para locação.
  - Esse arquivo já está disponível com cerca de 100 veículos
  - Cada registro tem 88 bytes.
  - Um exemplo de leitura de um registro desse arquivo pode ser visto no apêndice.
  - Cada registro possui:
    - **Placa do Veículo** (*string*) - **Chave Primária**
    - Modelo (*string*)
    - Marca (*string*)
    - Ano (*inteiro*)
    - Categoria (*string*) (ex: econômico, luxo, SUV)
    - Quilometragem (*inteiro*)
    - Status de Disponibilidade (*string*) (disponível, alugado, em manutenção)
- **Chave Primária**:
  - A **Placa do Veículo** será utilizada como chave primária na Árvore B.

Entrega

- **Código Fonte**:
  - Arquivos com o código-fonte estruturado e comentado.
- **Arquivos Gerados**:
  - `btree_M.idx`: Arquivo binário da Árvore B.
  - `veiculos.dat`: Arquivo de dados dos veículos.
  - outros arquivos auxiliares.
- **Instruções**:
  - Passos para compilação e execução do programa.
  - Dependências e requisitos do sistema.
- **Entrega**: arquivo `Nome_RA.zip`
- **Data de entrega:** 05/11 via *classroom*. Trabalhos fora do prazo são desconsiderados.

Avaliação

- Todos trabalhos devem ser apresentados.
- Apresentação também é parte da avaliação.
- Qualquer tipo de plágio resultado em nota zero.
- Trabalhos podem ser feitos em dupla. Nesse caso, será decrementado -1 da nota final.
- Para trabalhos em duplas, deve ser informado até dia 10/10 através de comentário na postagem da atividade no *classroom* colocando o nome da dupla. Após essa data, não serão aceitas mudanças.

Bom Trabalho!

1 Apêndice

1.1 Ler um registro do arquivo de dados

O código abaixo ilustra como ler um registro de um veículo do arquivo de dados `veiculos.dat`.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define TAMANHO_PLACA 8
#define TAMANHO_MODELO 20
#define TAMANHO_MARCA 20
#define TAMANHO_CATEGORIA 15
#define TAMANHO_STATUS 16

typedef struct {
    char placa[TAMANHO_PLACA];
    char modelo[TAMANHO_MODELO];
    char marca[TAMANHO_MARCA];
    int ano;
    char categoria[TAMANHO_CATEGORIA];
    int quilometragem;
    char status[TAMANHO_STATUS];
} Veiculo;

int main() {
    FILE *datFile = fopen("veiculos.dat", "rb");
    if (datFile == NULL) {
        perror("Erro ao abrir o arquivo veiculos.dat");
        return 1;
    }
```

```
    size_t tamanho_registro = sizeof(Veiculo);
    printf("Tamanho de um registro em bytes: %zu\n",tamanho_registro);

    Veiculo veiculo;

    // Ler o registro
    size_t registros_lidos = fread(&veiculo, tamanho_registro, 1, datFile);
    if (registros_lidos != 1) {
        printf("Erro ao ler o registro");
        fclose(datFile);
        return 1;
    }

    // Imprimir os dados do veículo
    printf("Registro de RNN 0\n");
    printf("Placa: %s\n", veiculo.placa);
    printf("Modelo: %s\n", veiculo.modelo);
    printf("Marca: %s\n", veiculo.marca);
    printf("Ano: %d\n", veiculo.ano);
    printf("Categoria: %s\n", veiculo.categoria);
    printf("Quilometragem: %d\n", veiculo.quilometragem);
    printf("Status: %s\n", veiculo.status);
    printf("--------------------------\n");

    fclose(datFile);

    return 0;
}
```

Será impresso:

```
Tamanho de um registro em bytes: 88
Registro de RNN 0
Placa: GIA5915
Modelo: Civic
Marca: Renault
Ano: 2000
Categoria: Hatch
Quilometragem: 124098
Status: Em manutenção
--------------------------
```