UNIVERSIDADE FEDERAL DE LAVRAS [Introdução aos Algoritmos] [Marluce]

Projeto Prático: Sistema de Cadastro em Arquivos com Ordenação

Tema: Cadastro de Frutas Tropicais

Alunos:

- Matheus Fellipe Araujo Marques
- Henrique Furtini

Lavras, [DATA] de 2025

1. Introdução

O objetivo deste trabalho foi a implementação de um sistema de cadastro completo, desenvolvido na linguagem C++, capaz de realizar as quatro operações básicas (CRUD: Criar, Ler, Atualizar e Deletar) sobre um conjunto de dados, com armazenamento persistente em arquivos. O sistema foi projetado para carregar os dados de um arquivo CSV em um vetor alocado dinamicamente na memória, permitindo que o usuário realize diversas operações sobre esses dados antes de, opcionalmente, salvá-los de volta no arquivo.

O tema escolhido para a base de dados foi "Frutas Tropicais", que inclui cinco campos distintos, satisfazendo os requisitos de conter múltiplos tipos de dados (numérico e textual) e ao menos um campo textual com espaços.

O desenvolvimento focou no domínio de conceitos centrais da disciplina, como a alocação dinâmica e redimensionamento de vetores, e a implementação manual de algoritmos eficientes de ordenação e busca, sem o uso de bibliotecas de containers (como `<vector>`) ou funções prontas (como `sort()`). O programa foi desenvolvido e testado em ambiente Windows, com a devida atenção para garantir a portabilidade e execução em um sistema Linux, conforme exigido.

2. Descrição das Estruturas e Lógica do Programa

O sistema foi estruturado em torno de uma `struct` principal e um conjunto de funções modulares que operam sobre um vetor dinâmico.

Estruturas de Dados:

- **`struct Frutas`**: É a estrutura central que agrupa os dados de cada registro. Ela contém cinco campos: `id` (inteiro), `calorias` (inteiro), `nome` (string), `continente` (string) e `coloração` (string).
- **Vetor Dinâmico (`Frutas* vetor`)**: O armazenamento principal dos dados em memória é um vetor alocado dinamicamente no *heap*. Este vetor é um ponteiro para um bloco de memória do tipo `Frutas`.
- **Variáveis de Controle:** Duas variáveis inteiras (`capacidade_total` e `n_elementos_vetor`) são usadas para gerenciar o vetor. A primeira rastreia o tamanho total alocado, e a segunda, quantos elementos estão de fato em uso.
- **Flag de Ordenação (`string ordenadoPor`)**: Uma variável de estado (string) armazena o critério de ordenação atual ("nenhum", "id" ou "nome") para validar a execução da busca binária.

Lógica do Programa:

1. **Inicialização:** Ao iniciar, o `main` aloca o vetor dinâmico com uma capacidade inicial de 40 elementos e inicializa `n_elementos_vetor` com 0.
2. **Carregamento (`carregar`):** A função `carregar` é chamada para popular o vetor. Ela abre o arquivo `frutastropicais.csv` e pula a linha de cabeçalho. Em seguida, ela lê o arquivo *linha por linha* usando `getline`. Para garantir a portabilidade entre Windows (que usa `\r\n`) e Linux (que usa `\n`), cada linha lida é processada por um `stringstream`. Este `stringstream` é usado para extrair cada campo, usando a vírgula como delimitador. Os campos de texto (ID, Calorias) são convertidos para inteiros usando `stoi`.
3. **Redimensionamento (`adicionar_elementos`):** Durante o carregamento (ou inserção), se `n_elementos_vetor` atingir a `capacidade_total`, esta função é chamada. Ela aloca um novo vetor com 10 posições a mais, copia todos os elementos do vetor antigo para o novo, libera a memória (`delete[]`) do vetor antigo e faz o ponteiro original apontar para o novo vetor.
4. **Menu Principal:** Um loop `do-while` na função `main` exibe o menu de opções. Um `switch-case` direciona a escolha do usuário para a função correspondente.
5. **Inserção (`inserirElemento`):** O usuário informa os dados da nova fruta. O programa verifica se há espaço e, se necessário, chama `adicionar_elementos`. O novo registro é então adicionado ao final do vetor, na posição `n_elementos_vetor`, e este contador é incrementado. Ao inserir, a flag de ordenação é resetada para "nenhum".
6. **Remoção (`removerElemento`):** O usuário informa um ID. O sistema usa uma busca linear (`buscarPorId_Linear`) para encontrar o índice do item. Caso encontrado, o programa realiza uma **remoção física**: ele "puxa" todos os elementos subsequentes uma posição para a esquerda (sobreescrevendo o item a ser removido) e, ao final, decrementa `n_elementos_vetor`.

7. **Ordenação (`ordenarDados`):** O usuário escolhe ordenar por ID ou por Nome. O programa então chama a função correspondente (`quickSortPorId` ou `quickSortPorNome`), que implementa o algoritmo **QuickSort** para reordenar o vetor em memória. A flag `ordenadoPor` é atualizada.
8. **Busca (`buscarElemento`):** O usuário escolhe buscar por ID ou por Nome. O sistema primeiro verifica a flag `ordenadoPor`. Se o vetor não estiver ordenado pelo critério correto, ele avisa o usuário e cancela a operação. Se estiver ordenado, ele chama a função de **Busca Binária** (`buscaBinariaPorId` ou `buscaBinariaPorNome`), que retorna o índice do elemento encontrado (ou -1).
9. **Listagem (`mostrarDados`):** Permite ao usuário listar todos os elementos (de 0 a `n_elementos_vetor`) ou um trecho específico (de `início` ao `fim`), imprimindo-os na ordem em que estão armazenados no vetor.
10. **Gravação (`gravarArquivo`):** A função abre o `frutastropicais.csv` em modo de escrita (sobrescrevendo o original). Ela primeiro escreve a linha de cabeçalho e, em seguida, percorre o vetor de 0 até `n_elementos_vetor`, salvando cada registro no formato CSV.

3. Ordem dos Dados no Arquivo

Os dados são armazenados no arquivo `frutastropicais.csv`, que acompanha o projeto. A ordem e o tipo dos campos em cada linha são os seguintes:

1. **#ID:** Inteiro. Identificador único de cada fruta.
2. **Nome da Fruta:** String. Nome comum da fruta.
3. **Continente de Origem:** String. Região de origem, podendo conter espaços.
4. **Calorias (aprox. por 100g):** Inteiro. Valor numérico.
5. **Coloracao:** String. Cor predominante da fruta.

A primeira linha do arquivo é um cabeçalho que descreve estes campos. Cada linha subsequente representa um registro, com os campos separados por vírgula.

4. Acertos e Erros Durante o Desenvolvimento

Durante a implementação do projeto, a equipe pôde solidificar conceitos teóricos, mas também enfrentou desafios práticos significativos.

Acertos:

- **Escolha dos Algoritmos:** A implementação do QuickSort atendeu ao requisito de usar um método eficiente que não fosse o BubbleSort. Da

mesma forma, a Busca Binária garantiu uma funcionalidade de pesquisa rápida, demonstrando o domínio de busca eficiente.
- **Modularização:** A separação do código em funções claras (ex: `carregar`, `gravar`, `ordenar`, `buscar`) tornou o `main` limpo e facilitou a depuração.
- **Gestão de Memória:** O sistema de alocação e redimensionamento dinâmico foi implementado com sucesso, permitindo que o programa se adapte a entradas de dados de tamanho variável.

Erros e Desafios:

- **O "Bug de 0 Frutas" (Portabilidade):** O maior desafio foi, sem dúvida, a leitura do arquivo CSV. O código inicial, que usava `getline` para cada campo separadamente, funcionava em alguns ambientes, mas falhava silenciosamente no Windows. O programa reportava "Carga finalizada. 0 frutas carregadas."
    - **Causa:** O Windows utiliza `\r\n` (carriage return e line feed) para quebras de linha. O `getline` final lia até o `\n`, mas deixava o `\r` no *buffer*. Na iteração seguinte, o `getline` lia o `\r` junto com o ID (ex: `"\r2"`), o que causava a falha da função `stoi`.
    - **Solução:** O problema foi resolvido refatorando a função `carregar` para ler a *linha inteira* primeiro (consumindo o `\r\n`) e, em seguida, usar um `stringstream` para processar essa linha em memória. Isso tornou o código robusto e portável.
- **Erro de Caminho (Path):** Um erro simples, mas que consumiu tempo, foi o de "Arquivo não encontrado". Isso ocorreu porque o ambiente de desenvolvimento (IDE) estava executando o programa (`.exe`) de dentro de uma subpasta (`/output`), enquanto o arquivo (`.csv`) estava na pasta raiz do projeto. A solução foi garantir que ambos estivessem no mesmo diretório de execução.

5. Conclusão

Este projeto prático permitiu aplicar e consolidar os conhecimentos adquiridos em sala de aula sobre estruturas de dados fundamentais. A equipe obteve sucesso em desenvolver um sistema de cadastro 100% funcional, atendendo a todos os requisitos obrigatórios da especificação.

Os objetivos de aprendizado relacionados ao gerenciamento de memória, especificamente a **alocação dinâmica**, o **redimensionamento de vetores** e a manipulação de ponteiros, foram alcançados. Além disso, a implementação manual de algoritmos clássicos como **QuickSort** e **Busca Binária** foi crucial para compreender não apenas *o que* eles fazem, mas *como* eles operam internamente.

Os desafios encontrados, especialmente na manipulação de arquivos entre diferentes sistemas operacionais (Windows e Linux), serviram como uma lição valiosa sobre a importância de escrever código portável e robusto. Ao final, o programa se mostrou estável, eficiente e capaz de gerenciar a base de dados de Frutas Tropicais conforme o esperado.