Universidade Federal de Lavras
Departamento de Ciência da Computação
Introdução aos Algoritmos

Gabriele Macedo Gimenez
Maria Lina da Silva

Músicas

Projeto Prático - Turma 22W

Agosto
2024

Descrição em alto nível

A função **`redimensionarVetor`** é usada para ajustar o tamanho de um vetor dinâmico que armazena as músicas. Quando o vetor atinge sua capacidade máxima, a função cria um novo vetor com uma capacidade maior, copia os elementos do vetor original para o novo vetor, libera a memória ocupada pelo vetor antigo e atualiza a capacidade do vetor. A função retorna um ponteiro para o novo vetor redimensionado.

A função **`carregarCSV`** é responsável por ler e carregar dados de um arquivo CSV para um vetor de músicas. Ela abre o arquivo CSV e, enquanto lê cada linha, preenche a estrutura de dados da música com os valores extraídos. Se o vetor de músicas estiver cheio, a função redimensiona o vetor para acomodar mais elementos. Após a leitura de todos os dados, a função fecha o arquivo e retorna um status indicando se a operação foi bem-sucedida.

A função **`exportarParaCSV`** é responsável por salvar os dados das músicas em um arquivo CSV. Ela abre um arquivo CSV para escrita e, para cada música válida no vetor, escreve os detalhes da música no arquivo no formato especificado. Após a escrita dos dados, a função fecha o arquivo e exibe uma mensagem de confirmação.

A função **`inserirMusica`** adiciona uma nova música ao final do vetor de músicas. Se o vetor atingir sua capacidade, a função redimensiona o vetor para garantir espaço suficiente. A nova música é então adicionada ao vetor, e a quantidade total de músicas é atualizada.

A função **`removerMusica`** realiza a remoção lógica de uma música do vetor. Ela percorre o vetor procurando a música com o nome fornecido. Quando encontra a música, marca-a como não válida (removida logicamente) e exibe uma mensagem de confirmação. Se a música não for encontrada, a função exibe uma mensagem informando isso.

A função **`buscarMusicaBinaria`** realiza uma busca binária para encontrar uma música no vetor com base no nome fornecido. A função assume que o vetor está ordenado. Ela busca o nome da música, e se encontrada é válida, retorna o índice da música e um status indicando sucesso. Caso contrário, retorna um status indicando falha na busca.

A função **`quickSort`** ordena o vetor de músicas utilizando o algoritmo Quick Sort. Ela seleciona um elemento pivô e particiona o vetor em torno desse pivô, garantindo que todos os elementos menores que o pivô venham antes dele e todos os

elementos maiores venham depois. A função é chamada recursivamente para ordenar as subpartições resultantes.

A função **`mostrarTrechoMusicas`** exibe um intervalo específico de músicas do vetor, definido pelos parâmetros de início e fim. A função verifica se o intervalo é válido e, em caso afirmativo, imprime as informações de cada música dentro desse intervalo, incluindo nome, duração, data de lançamento, artista e gênero. Se o intervalo não for válido, exibe uma mensagem de erro.

A função `main` é o ponto de entrada do programa e apresenta um menu interativo para o usuário. O menu permite ao usuário escolher entre várias opções, como mostrar todas as músicas, buscar uma música, inserir ou remover uma música, ordenar as músicas, e exportar os dados para um arquivo CSV. A função controla a execução das opções selecionadas pelo usuário e gerencia a memória do vetor de músicas.

Ordem dos dados

A **struct** chamada **Músicas** armazena informações sobre as músicas. Ela possui os seguintes campos:

- Char nome - Armazena o nome da música;
- Int duracao - Guarda a duração da música;
- Char dataLancamento - Contém a data de lançamento da música;
- Char artista - Armazena o nome do artista;
- Char gênero - Indica o gênero musical da música.

Acertos e erros

Pensamos em estruturar o código de forma a fazer a leitura do arquivo e o processamento dos dados usando um bloco único de funções. Nossa ideia era concentrar todas as atividades em uma única função, iria realizar de leitura completa, ordenação e gravação dos dados. No entanto, essa abordagem acabou tornando a lógica mais complexa e difícil de depurar.

Após rever essa estrutura, resolvemos dividir as tarefas em funções menores e mais específicas, o que nos permitiu ter um melhor controle sobre o fluxo do programa. Ao lidar com o arquivo, enfrentamos o desafio de eliminar caracteres indesejados, como quebras de linha e espaços em branco, que estavam presentes no início e no final de cada bloco de texto. Isso exigiu a criação de funções específicas para sanitizar os dados antes de processá-los. O professor Joaquim sugeriu a utilização de um buffer intermediário para manipular os dados durante a leitura, o que tornou a implementação mais simples.

O gerenciamento da memória foi outro desafio, especialmente ao lidar com grandes quantidades de informações. Adotamos uma estratégia de alocação dinâmica, na qual realocamos a memória conforme necessário para evitar

desperdícios e assegurar que o programa pudesse lidar com os dados de maneira eficiente. Este processo envolveu criar novos vetores, copiar informações já existentes e liberar a memória antiga. Decidimos manter a função principal o mais organizada possível, focando apenas em chamadas de tarefas e transferindo as tarefas específicas para tarefas auxiliares. O que tornou mais fácil identificar e corrigir problemas.

De início tínhamos feito a inserção de números de faixa na estrutura de dados, professor joaquim deu a sugestão da exclusão já que o objetivo era busca de músicas e não de álbuns

Conclusão

Apesar dos desafios que enfrentamos, o sistema possui funcionalidade para busca e alteração de dados, utiliza arquivo binário para o armazenamento das informações, gerenciamento de memória através de alocação dinâmica, conversão de dados para o arquivo binário e ordenação utilizando o quicksort.