RELATÓRIO

Nome:

- Gustavo Gerônimo Ribeiro Turma: 10 A
- Lívia Maria Almeida Silva 10 A
- Maurício Vicente Sandim 10 A

Introdução

O objetivo deste trabalho é construir um código que receba um grande arquivo binário (property-transfer-statistics-march-2022-quarter-csv.csv), cujo tamanho impede sua ordenação na memória primária. Esse arquivo deve ser ordenado de forma crescente utilizando o método de intercalação polifásica.

Além disso, o sistema deve contar com funções básicas para visualizar parte do arquivo, visualizar todo o conteúdo, trocar posições entre registros, inserir novos dados e alterar registros existentes.

Tema

Tipo de ordenação: Intercalação polifásica crescente

Arquivo CSV: property-transfer-statistics-march-2022-quarter-csv.csv

Organização do Código

O código foi organizado em duas pastas: a pasta src/, que contém todos os arquivos .cpp, e a pasta include/, que contém todos os arquivos .h. O arquivo CSV utilizado na execução do código deve estar localizado fora dessas pastas.
Há também um Makefile que facilita a compilação do código.

Struct: dado

A struct usada para armazenar os dados do arquivo contém 14 campos:

- Series_reference (vetor de char)
- Period (float)
- Data_value (float)
- Status (char)
- Units (vetor de char)
- Magnitude (int)

- Subject (vetor de char)
- Periodicity (vetor de char)
- Group (vetor de char)
- Series_title_1 (vetor de char)
- Series_title_2 (vetor de char)
- Series_title_3 (vetor de char)
- Series_title_4 (vetor de char)
- Series_title_5 (vetor de char)

Arquivo: main.cpp

O arquivo main.cpp é responsável pela leitura do arquivo .csv e sua conversão para o formato .bin. Há tratamento de exceção caso não consiga ler o arquivo.

Durante a leitura do CSV, foi feito um tratamento especial no campo Data_value, pois em alguns registros ele estava vazio. Nestes casos, o valor foi substituído por -1.

Também houve tratamento no campo Group, pois ele continha informações com vírgulas em seu conteúdo.

A função ordenar() é chamada logo no início da execução, antes que qualquer outra ação seja feita, ou seja, o programa começa com o arquivo sendo ordenado.

Além disso, há a função menu(), que apresenta todas as opções disponíveis ao usuário e, de acordo com a escolha, chama as funções correspondentes:

- [1] Adicionar um elemento no arquivo em uma posição específica. → chama inserir()
- [2] Visualizar os registros entre duas posições. → chama mostra_x_ate_y()
- [3] Alterar os dados de um registro. → chama alterar()
- [4] Trocar dois registros de posição no arquivo. → chama trocar()
- [5] Visualizar todos os registros do arquivo armazenados. → chama visualizarTudo()
- [6] Exportar arquivo para CSV. → chama exportaArquivo()
- [0] Encerrar o programa. → encerra o laço de execução

Arquivo: ORDENAR.cpp

Este arquivo contém as funções responsáveis pela implementação da intercalação polifásica crescente, utilizando o campo Data_value como critério de ordenação.

A função principal, ordenar(), inicia com o comando system("clear") para limpar o terminal e, em seguida, chama criaBlocos().

Função criaBlocos()

Cria um heap de 10 mil posições, inicialmente com todos os registros com a variável marcado igual a false.

Dentro do laço while, é construído o arquivo fita1, que recebe a raiz do heap. Esta raiz é removida e substituída por um novo registro vindo do arquivo original. Se esse novo registro for menor que a raiz retirada, seu campo marcado é definido como true.

Quando o heap estiver composto apenas por registros com marcado == true, todas as variáveis marcado são resetadas para false. Esse processo se repete até que o arquivo original esteja vazio. Assim, fita1 conterá blocos ordenados de tamanhos variados. O restante dos dados presentes no heap é escrito em fita2.

Função intercala()

Após a construção das fitas, a função intercala() é chamada. Enquanto a fita1 não estiver vazia, ocorre a intercalação entre os arquivos, ora entre fita1 e fita2, ora entre fita1 e fita3.

Como a ordenação é crescente, são comparados os menores valores entre os dois arquivos, e o menor é escrito no terceiro arquivo. Esse processo se repete até que uma das fitas ou o bloco em questão da fita1 acabe.

Após o conteúdo da fita1 acabar ela é apagada, e o mesmo ocorre com fita que foi lida por última. A fita que ficou com o conteúdo ordenado recebe o nome o.bin.

Arquivo: INSERE.cpp

Contém a função inserir(), que insere um novo registro em uma posição específica informada pelo usuário. A posição deve estar entre 0 e o total de registros existentes.

Após o usuário informar a posição desejada, ele deve preencher todos os campos do novo registro. Para garantir a integridade dos dados, cada entrada passa por uma função de validação.

Com os dados validados, todos os registros a partir da posição escolhida são deslocados uma posição à frente, abrindo espaço para o novo registro.

Arquivo: MOSTRARPARTE.cpp

Contém a função mostra_x_ate_y(), responsável por exibir uma parte específica do arquivo.

O usuário informa a posição inicial (maior que 0 e menor que o número total de registros) e a posição final (maior ou igual à inicial e também dentro dos limites do arquivo).

Usando a função seekg(), o ponteiro de leitura é movido até a posição inicial, e, por meio de um laço for, os registros entre as posições selecionadas são lidos e exibidos no terminal.

Arquivo: ALTERA.cpp

Contém a função alterar(), que permite modificar os campos de um registro específico.

Após selecionar a posição, o usuário vê um menu com todos os campos disponíveis para edição e a opção "[0] Finalizar alterações " que sai do loop do menu. Ao selecionar um campo, é solicitado um novo valor, que também passa por validação.

Quando as alterações forem concluídas, o registro modificado é exibido.

Arquivo: TROCA.cpp

A função trocar() permite ao usuário informar duas posições de registros a serem trocadas.

As informações desses registros são armazenadas em variáveis temporárias do tipo dado, e em seguida são gravadas de volta no arquivo, mas com suas posições invertidas.

Arquivo: MOSTRARTUDO.cpp

A função visualizarTudo() começa exibindo os 100 primeiros registros do arquivo.

Após isso, o usuário pode optar por visualizar mais 100 registros ou o arquivo completo. O processo se repete até que o usuário escolha a opção para mostrar o arquivo inteiro ou o arquivo chegue ao fim.

Arquivo: EXPORTAARQUIVO.cpp

A função exportaArquivo() converte o arquivo binário que estamos usando em um arquivo .csv e o exporta.

Primeiro, o cabeçalho é adicionado. Em seguida, o conteúdo do arquivo binário é lido e escrito no novo CSV, cujo nome é definido pelo usuário.

Arquivo: VALIDACAO.cpp

Este arquivo contém quatro funções de validação:

- validacao() – Confere se a string recebida ultrapassa o tamanho máximo do vetor de char.
- validacaoInt() – Verifica se a string contém apenas números.
- validacaoData() – Valida se a string está no formato de data 0000.00.
- validacaoFloat() – Verifica se a string representa um número do tipo float, ou seja, contém apenas dígitos e no máximo um ponto (.).

Makefile

O Makefile facilita o processo de compilação, permitindo que o programa seja compilado com o simples comando ‘make’.

Para executá-lo, basta digitar ‘./projeto’ no terminal.

Pasta include/

Dentro da pasta include/ estão localizados todos os arquivos de cabeçalho (.h) correspondentes aos arquivos .cpp.

Conclusão

Com a conclusão deste projeto, aprendemos, de forma prática, a lidar com grandes arquivos CSV e a ordená-los por meio da intercalação polifásica, uma das

técnicas utilizadas quando a quantidade de dados é grande demais para ser totalmente carregada na memória RAM. Além disso, pudemos perceber como o processo de ordenação em memória secundária é mais lento e complexo quando comparado aos algoritmos de ordenação em memória primária, estudados na disciplina de Introdução aos Algoritmos.

Por meio deste projeto, também colocamos em prática conhecimentos adquiridos anteriormente, tanto na disciplina de Estrutura de Dados quanto na de Introdução aos Algoritmos, como a manipulação de strings, a manipulação de arquivos tipados e o uso da estrutura de dados heap, empregada na técnica de seleção por substituição.

Além disso, aprendemos técnicas ainda não abordadas em sala de aula que contribuem para a melhor organização de projetos, como a separação de funcionalidades em arquivos distintos e a criação de um Makefile, que facilita o processo de compilação.

Por fim, também exploramos mais a fundo o processo de documentação de um projeto, compreendendo sua importância tanto para os usuários quanto para nós, desenvolvedores, que trabalhamos diretamente com o código.