UNIVERSIDADE FEDERAL DE LAVRAS
DEPARTAMENTO DE COMPUTAÇÃO APLICADA

Nome: Gabriel Henrique De Paula Santos **Matrícula:** 202320596
Nome: Marina Barbosa Valim **Matrícula:** 202410197
Nome: Vitor Gabriel Firmino **Matrícula:** 202320923

1. Introdução

Este relatório descreve o desenvolvimento de um sistema para gerenciamento e ordenação de dados provenientes de arquivos CSV, com foco na manipulação de grandes volumes de informações que excedem a capacidade da memória principal. O projeto implementa funcionalidades de conversão de CSV para formato binário, visualização, inserção, alteração e troca de registros, culminando em um robusto algoritmo de ordenação externa baseado no método *Multi-Way Merge Sort* otimizado por uma estrutura *Max-Heap*. O objetivo principal é demonstrar a aplicação de estruturas de dados e algoritmos eficientes para o tratamento de dados em memória secundária, garantindo a integridade e a ordenação dos registros sem sobrecarregar os recursos do sistema.

2. Estruturas de Dados Utilizadas

O sistema faz uso de três estruturas de dados principais para cumprir seus objetivos:

2.1. EntradaDadosCsv

A classe EntradaDadosCsv é fundamental para a representação de cada registro de dados no sistema. Ela encapsula os diversos campos extraídos do arquivo CSV, como referência de série, período, valor de dado, status, unidades, magnitude, assunto, periodicidade, grupo e múltiplos títulos de série. A escolha de tipos de dados apropriados (e.g., char arrays para strings, double para valores numéricos e int para inteiros) garante a compatibilidade com o formato binário. Além de armazenar os dados, esta classe contém a lógica de comparação para a ordenação, permitindo que os registros sejam ordenados com base em critérios específicos, como o período e a referência da série, de forma decrescente.

2.2. NohHeap

A estrutura NohHeap é um auxiliar para a implementação da *Max-Heap*. Cada nó da *heap* armazena uma instância de EntradaDadosCsv (o dado real a ser ordenado) e um índice (long indiceArq) que refere-se ao arquivo temporário de

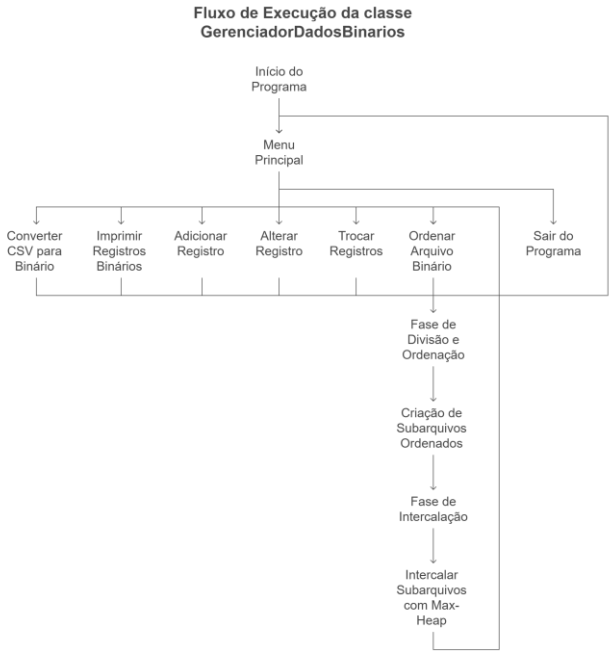
onde aquele dado foi lido. Essa associação é crucial durante a fase de intercalação do algoritmo de ordenação externa, permitindo que o sistema saiba de qual subarquivo o próximo registro deve ser lido após a extração do maior elemento da *heap*.

2.3. MaxHeap

A classe MaxHeap implementa uma estrutura de dados *Max-Heap*, que é um tipo de árvore binária completa onde o valor de cada nó é maior ou igual ao valor de seus filhos. No contexto deste projeto, a *Max-Heap* é utilizada para otimizar a fase de intercalação do *Multi-Way Merge Sort*. Ao invés de comparar todos os primeiros elementos de cada subarquivo, a *Max-Heap* mantém o maior elemento entre os disponíveis na raiz, permitindo uma seleção eficiente do próximo registro a ser gravado no arquivo de saída ordenado. As operações básicas da *Max-Heap* incluem inserção (insereNoh) e extração do maior elemento (extraiRaiz), ambas mantendo a propriedade da *heap* através de operações de "corrigir subindo" (corrigeSubindo) e "corrigir descendo" (corrigeDescendo).

3. Lógica do Programa

A lógica do programa é orquestrada pela classe GerenciadorDadosBinarios, que atua como a interface principal para as operações do sistema. O fluxo de execução pode ser visualizado no fluxograma a seguir:



3.1. Conversão de CSV para Binário

A funcionalidade de conversão (converterCsvParaBinario) lê o arquivo CSV linha por linha, ignorando o cabeçalho. Para cada linha, os campos são extraídos utilizando a função extrairCampoCsv, que lida com delimitadores e aspas em strings. Valores numéricos são processados por analisarCampoNumerico e analisarCampoInteiro, que tratam casos de valores vazios ou "Not Provided". Cada conjunto de dados extraído é então empacotado em uma instância de EntradaDadosCsv e gravado diretamente no arquivo binário.

3.2. Impressão de Registros

A função imprimirArquivoBinario permite a visualização do conteúdo do arquivo binário. O usuário pode especificar um intervalo de posições para impressão ou optar por exibir todos os registros. O programa calcula o número total de registros e, a partir da posição inicial desejada, lê sequencialmente cada EntradaDadosCsv do arquivo binário e exibe seus campos na saída padrão.

3.3. Manipulação de Registros (Adição, Alteração, Troca)

- **Adicionar Registro (adicionarRegistroEmPosicao)**: Permite ao usuário inserir um novo registro em uma posição específica do arquivo

binário. Para isso, um arquivo temporário é criado. Os registros do arquivo original são copiados para o temporário até a posição de inserção, o novo registro é gravado, e então os registros restantes do arquivo original são copiados. Finalmente, o arquivo original é removido e o temporário é renomeado para o nome do arquivo original.

- **Alterar Registro (alterarRegistroEmPosicao)**: Permite modificar um registro existente em uma dada posição. O programa navega até a posição desejada no arquivo binário, lê o registro existente, solicita novos dados ao usuário e, em seguida, sobrescreve o registro na mesma posição com as novas informações.
- **Trocar Registros (trocarRegistros)**: Implementa a troca de posição entre dois registros no arquivo binário. O sistema lê os dois registros, armazena-os temporariamente, e então os grava de volta em suas posições trocadas.

3.4. Ordenação Externa (Multi-Way Merge Sort)

A funcionalidade de ordenação (ordenarArquivoBinario) é a parte central do projeto para lidar com grandes volumes de dados. Ela é implementada em duas fases principais, seguindo o conceito do *Multi-Way Merge Sort*:

3.4.1. Fase de Divisão e Ordenação (Criação de "Runs")

A função dividirEOrdenarSubArquivos divide o arquivo binário original em múltiplos subarquivos menores (chamados "runs"). Cada um desses blocos é carregado na memória principal, ordenado internamente utilizando o algoritmo *Merge Sort* (ordenarIntercalacao e intercalar), e então gravado em um arquivo binário temporário distinto. Este processo é repetido até que todo o arquivo original tenha sido dividido e seus blocos iniciais ordenados e salvos.

3.4.2. Fase de Intercalação (Merge)

A função intercalarSubArquivosOrdenados é responsável por combinar os subarquivos ordenados em um único arquivo de saída final, também ordenado. Esta fase é otimizada pelo uso da estrutura MaxHeap.
Um NohHeap é criado para o primeiro registro de cada subarquivo e inserido na MaxHeap. Em cada iteração, o maior registro é extraído da raiz da MaxHeap e gravado no arquivo de saída. Em seguida, o próximo registro do subarquivo de onde o registro extraído veio é lido e inserido na MaxHeap. Este processo continua até que a MaxHeap esteja vazia, garantindo que o arquivo de saída final esteja completamente

ordenado em ordem decrescente. Após a conclusão, os subarquivos temporários são removidos.

4. Compilação e Execução

O projeto foi desenvolvido para ser compilado e executado em um ambiente Linux. Para compilar o código-fonte C++, é necessário ter um compilador C++ (como o g++) instalado. As instruções detalhadas para compilação e execução, que geralmente envolvem o uso de um Makefile, podem ser encontradas no arquivo README.txt fornecido com o projeto.

5. Conclusão

O sistema desenvolvido demonstra uma solução eficaz para o gerenciamento e ordenação de grandes conjuntos de dados que não se encaixam completamente na memória principal. A utilização do algoritmo *Multi-Way Merge Sort* em conjunto com a estrutura *Max-Heap* para a ordenação externa se mostrou uma abordagem robusta e eficiente, minimizando o uso de recursos de memória e otimizando as operações de I/O em disco. As funcionalidades de conversão, impressão e manipulação de registros complementam o sistema, oferecendo uma ferramenta completa para o trabalho com dados binários. Este projeto serve como uma excelente demonstração prática dos conceitos de estruturas de dados e algoritmos aplicados a cenários de dados em larga escala