UNIVERSIDADE FEDERAL DE LAVRAS – UFLA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

DAVI HERMÓGENES SIQUEIRA
GUSTAVO SOARES SILVA
RAFAEL FURTADO MORAIS

RELATÓRIO DO PROJETO PRÁTICO

ETAPA 2

ESTRUTURA DE DADOS

MINISTRADA PELOS DOCENTES
RENATO RAMOS DA SILVA E
JOAQUIM QUINTEIRO UCHÔA

Lavras – MG

2022

Relatório:

A segunda parte do projeto consiste basicamente em ordenar uma base de dados sorteada, ficando o método de ordenação a cargo do grupo. O seguinte relatório tem como objetivo discutir o processo de desenvolvimento do projeto, bem como os métodos e estruturas utilizadas e a lógica geral do programa.

Processo de desenvolvimento:

Inicialmente, o desenvolvimento do programa girava em torno de uma ideia básica para ordenação, isso é, um ponto de partida para implementações iniciais. Algumas ideias surgiram ao longo desse processo, mas nenhuma era sólida o suficiente a ponto de ser implementada.

Após esse processo inicial, foram buscados métodos já testados para ordenação secundária. A maioria se baseia na seguinte ideia: Os dados são divididos e ordenados em blocos menores e enviados para arquivos temporários. Após isso, se inicia o processo de ordenação entre os arquivos. A ideia, juntamente com a implementação, são bem eficazes para ordenação secundária, visto que o objetivo da ordenação secundária é fazer uso de artifícios para ordenar dados que, normalmente, seriam grandes demais para ordenação completa em memória primária.

Tivemos várias referências ao longo do desenvolvimento, mas a maior referência, e que forneceu a base do código, foi encontrada em um site popular entre programadores, chamado “geeksforgeeks”. Nele, é fornecida uma explicação da lógica empregada e um código base, que para o exemplo deles, ordena números, que em seguida teve de ser alterada, visto que a base de dados dada para ordenação é composta de registros.

Posteriormente foram aplicadas mais algumas alterações no código base para que a ordenação funcionasse corretamente para a base sorteada. Além das funções para ordenação, o código contém funções auxiliares, algumas das quais já foram descritas no relatório da primeira parte; as que não foram descritas se encontram devidamente comentadas.

O projeto consiste em dois arquivos, um para o cabeçalho, com o registro e algumas funções (já detalhadas na primeira parte do projeto), e outro para a ordenação, conversão e manipulação do arquivo.

Todas as funções e partes do código não explicadas aqui foram recicladas da primeira parte do projeto prático.

Descrição Geral:

A função que realiza a chamada inicial é a “ExternalSort”, que é responsável por chamar as funções “criarArquivosIniciais” e “mergeArquivos”.

“criarArquivosIniciais” cria os arquivos temporários, abrindo todos para escrita. Em seguida lê os registros do arquivo principal, de mil em mil, os armazena em um vetor e ordena pelo algoritmo de MergeSort. Após essa ordenação, esse vetor com mil registros é gravado em um dos arquivos temporários.

Em seguida a função “mergeArquivos” é chamada para juntar os arquivos temporários no arquivo principal, de forma que se mantenha ordenado no final. Para isso é utilizado a estrutura de dados MaxHeap. Para uma ideia geral do funcionamento do código, é lido o primeiro registro de cada arquivo temporário e armazenado num vetor. Em seguida é feito um MaxHeap nesse vetor, sendo que o maior desses registros estará na raiz do Heap. Essa raiz é armazenada no arquivo principal e então é lido o segundo registro de onde essa raiz foi retirada e aplicado um heapify neste registro.

É feito isso até que todos os arquivos temporários tenham sido lidos. Ao final, os arquivos temporários são deletados. No código base foi utilizado um MinHeap, mas como a ordenação solicitada ao grupo era em ordem decrescente isso também teve de ser alterado.

Esclarecimentos e conclusão:

Ao longo do percurso foram ficando algumas dúvidas em relação à performance do código, mas nada que interferiu no funcionamento, apenas em performance. Como exemplo podemos citar a quantidade de arquivos temporários criados e a quantidade de registros escritos em cada um, sendo divididos em 271 arquivos preenchidos com 1000 registros. A proporção foi decidida de forma arbitrária, e se mostrou não muito diferente de outras proporções (em termos de performance), onde foram criados menos arquivos e inseridos mais registros (em cada arquivo).

Foram surgindo algumas dúvidas em relação ao modo como foi decidido o método de ordenação entre arquivos pego no código base, onde, como citado anteriormente, utilizava a ideia do MinHeap. Para ordenar os arquivos entre si, outros métodos foram sugeridos pelos professores, mas como deixaram isso a cargo do grupo, preferimos manter o método pego de referência, alterando apenas de Min para MaxHeap.

O desenvolvimento do projeto consiste, em sua grande maioria, no que foi apresentado acima, tanto o funcionamento quanto os métodos. Como foi informado pelo professor de que não seria necessária uma descrição técnica do código, e sim um processo de desenvolvimento, optamos por manter o pouco que foi descrito o mais breve possível.

Referências:

Aditya Goel, External Sorting, GeeksforGeeks, última atualização: 2 de set.de 2022, Dísponível em: https://www.geeksforgeeks.org/external-sorting/, Acesso em: 14 de agos.de2022.

Nunes, Graça. Ordenação Externa. Icmc Usp, São Carlos, out.2012. Disponível em: http://wiki.icmc.usp.br/images/1/1e/SCC0203-1o-2012-15.OrdenacaoExterna.pdf . Acesso em: 15 de agos.de 2022

External Sort-Merge Algorithm. Java T Point. Disponível em: https://www.javatpoint.com/external-sort-merge-algorithm . Acesso em: 15 de agos. De 2022.