Relatório de análise e construção do código de implementação do algoritmo de indexação de Lefkowitz na linguagem C

Davi Cardoso de Oliveira[1], Maxwell William Severiano Chagas[2], Rafael Souza de Almeida[3]

[1]Instituto Multidisciplinar – Universidade Federal Rural do Rio de Janeiro (UFRRJ)

[2]Departamento de Ciência da Computação

Abstract. *This report aims to present the process of constructing the Dynamic Hash Table code in C language and its analysis to verify its execution, with the objective of inserting the records using the hash table in partitions so that a direct access to a record can be made, that the table expands dynamically in view of its hashing function and the load factor. In addition to showing the results obtained in the execution of the code.*

Resumo. *Este relatório tem como objetivo apresentar o processo de construção do código de Lefkowitz em linguagem C e a análise do mesmo para verificar a sua execução, tendo como objetivo criar partições de modo que possa ser feito um acesso direto à um registro pela chave secundária dele e não mais pela chave primária. Além de mostrar os resultados obtidos na execução do código.*

1. Introdução

Para não precisar fazer uma busca sequencial pelas chaves secundárias e ler todos os registros anteriores na busca de um. É necessário acessar os arquivos de forma direta pela chave secundária que eu estou buscando, mas como? Uma solução é usar o algoritmo de Lefkowitz, que cria partições no seu passo a passo até ter a final com as chaves primárias ordenadas e um índice para cada chave secundária.

Esse trabalho visa implementar esse algoritmo de Lefkowitz na linguagem C com suas informações e criações de partições executadas, guardadas e mostradas no resultado final.

2. Registros

Os registros serão uma struct denominados de “CLIENTE” onde nela haverá o código do cliente como um inteiro, sua idade e o nome do mesmo como uma string.

2.1 Resumo dos códigos dos registros

Cliente.h: Biblioteca dos registros, aqui está inicializada a struct e os métodos nominados.

Cliente.c: Implementações dos métodos nominados da biblioteca “Cliente.h” como: salvar cliente, ler registro de cliente, inicializar cliente inserindo os elementos e retornar o tamanho do registro.

addclientes.h: Biblioteca para ter os métodos de inserção de clientes e leitura nominados

addclientes.c: Implementação dos métodos nominados da addclientes.h tendo as instâncias dos atributos dos clientes feitas aqui e métodos para inseri-las no registro da struct “Cliente”.

3. “Commits”

Partes do código feitas até termos o código final pronto.

3.1 First Commit

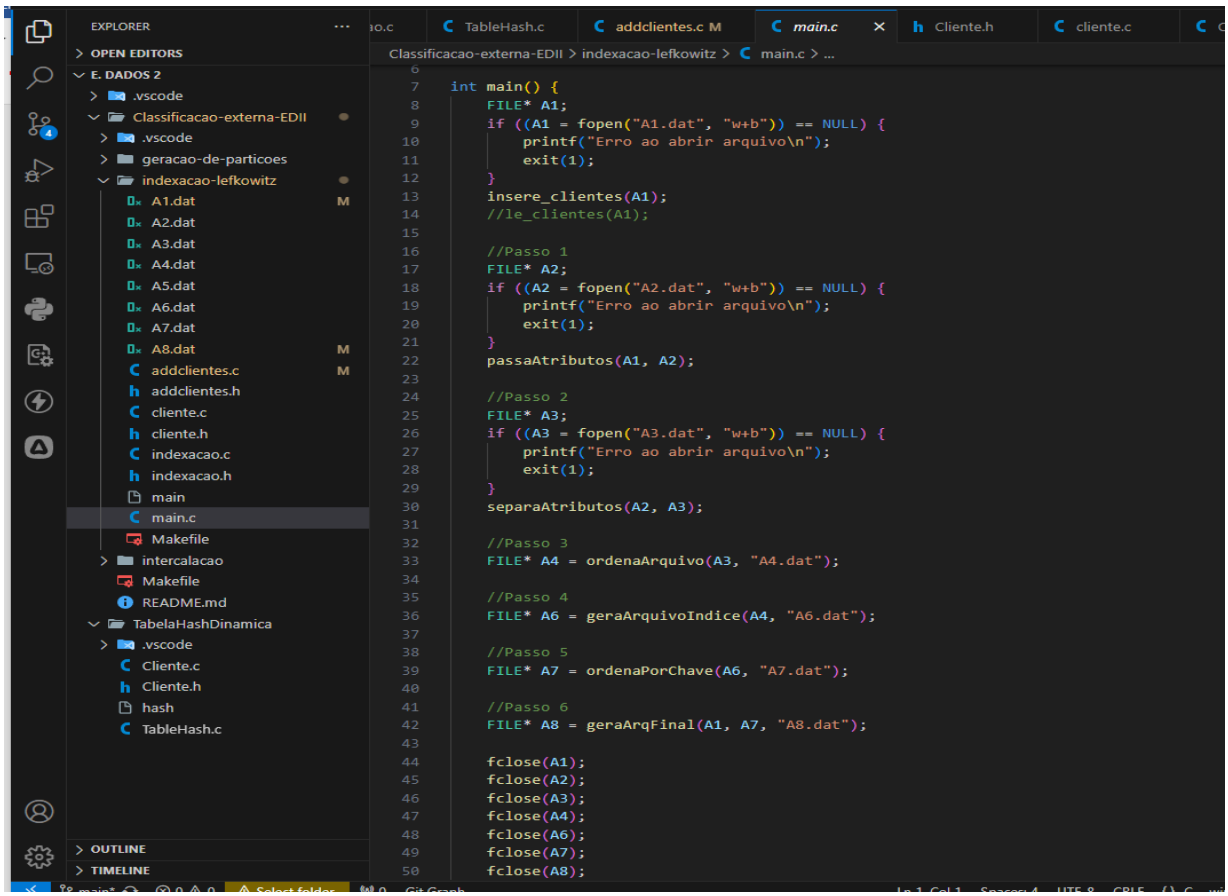
Foi feito o início do nosso código de implementação. Aqui consistimos em fazer o início dos arquivos dos registros (Cliente.h, Cliente.c e addclientes.h), e, o arquivo “indexacao.h” ao qual é a biblioteca com os métodos do algoritmo de Lefkowitz nominados e a struct “ClienteIndex” que será nossa struct auxiliar para “Cliente”.

3.2 Segundo Commit

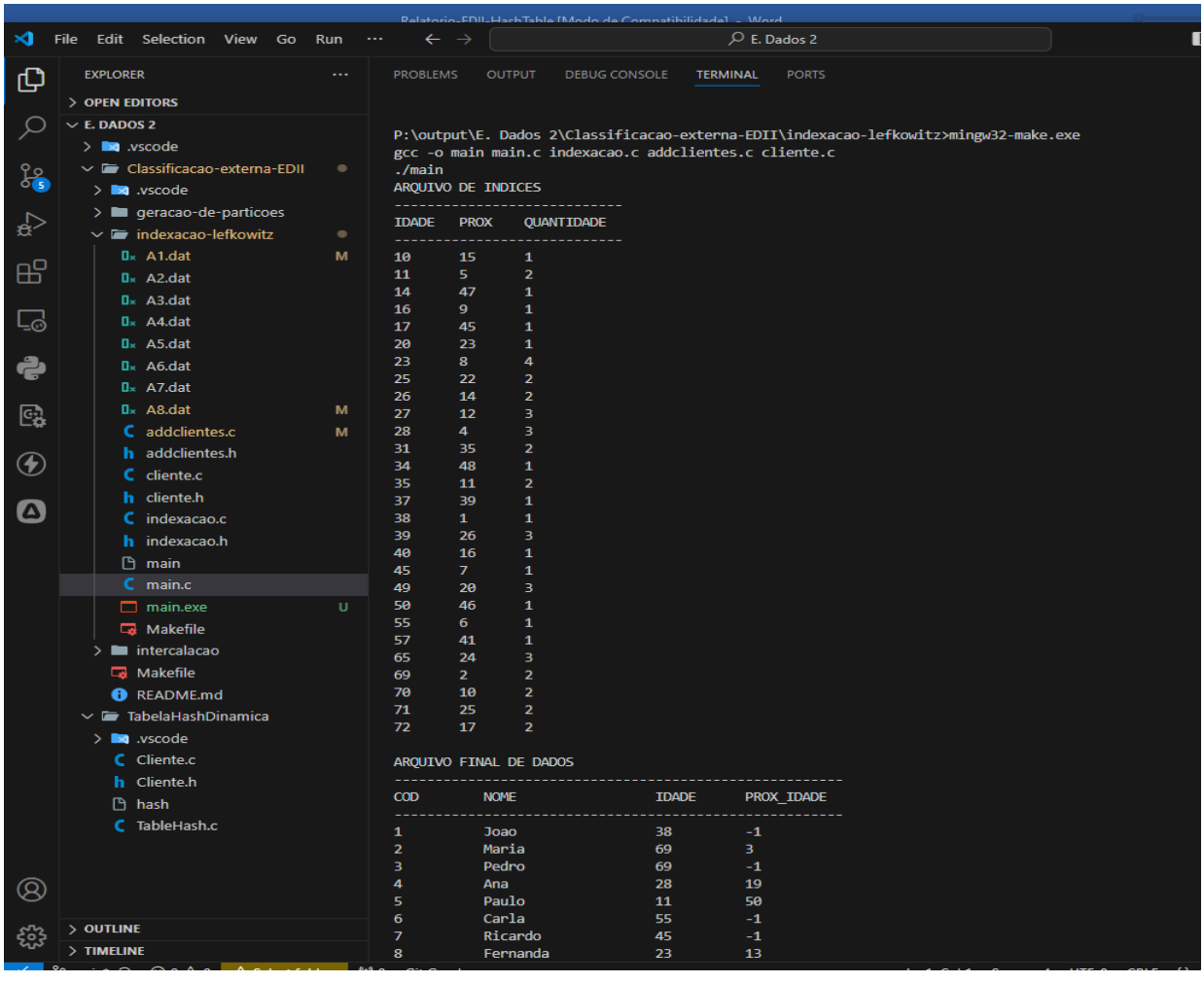
Após termos a todos os arquivos de registros e a struct “ClienteIndex” com os métodos do algoritmo de Lefkowitz nominados, agora neste commit foram realizadas as primeiras implementações desses métodos com o arquivo “indexacao.c”, entre eles foi feito: “passaAtributos”(Escreve na partição os atributos da partição inicial), “separaAtributos” (Escreve na struct Cliente 1 atributo da struct ClienteIndex para todos os registros das mesmas), “ordenaArquivo”(Ordenando os registros de clientes no arquivo), “ordenaPorChave” (Aqui estamos ordenando por chave primária os registros), “geraArquivoIndice”(Escrevendo o arquivo índice dos registros dos clientes da struct ClienteIndex, antes de ordenar por chave primária e termos o resultado final), “geraArquivoFinal(Após todo o passo a passo do algoritmo de Lefkowitz, geramos aqui nesse método o arquivo final contend as chaves primárias ordenadas, a quantidade de atributos iguais e o ponteiro para o próximo elemento.

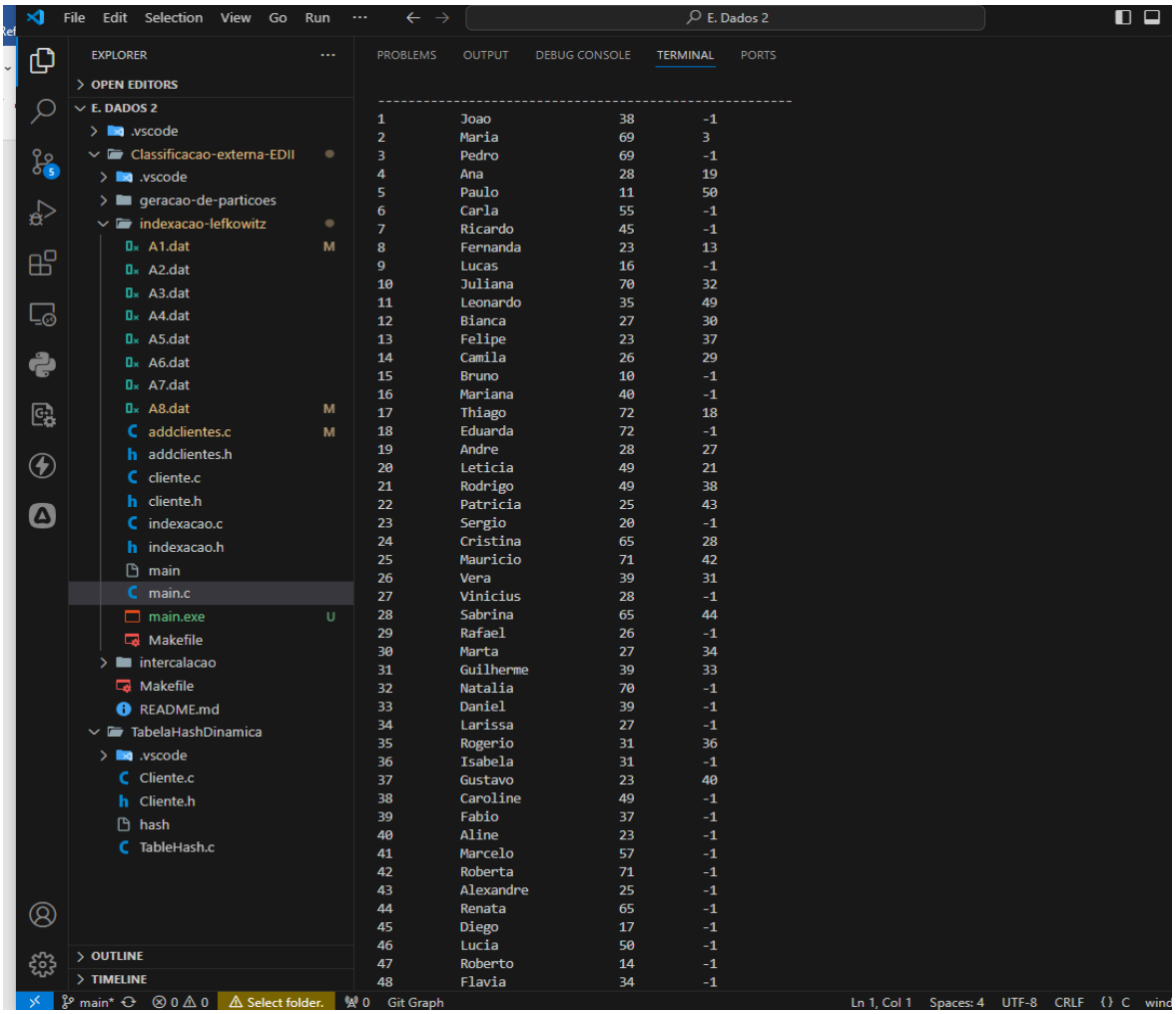
3.3 Commit final

No último passo foi feita a “main.c” onde basicamente colocamos os nomes das partições (A1, A2 , … , A8) e chamamos os métodos correspondentes a cada passo do algoritmo de Lefkowitz, mostraremos a seguir um print dessa main para mostrar o passo a passo do algoritmo e as chamadas das funções a cada partição:



4 Resultado final





5 Conclusão

Neste trabalho vimos a manipulação de arquivos, a criação de registros e a ordenação de chaves primárias e organização das chaves secundárias dos registros, de forma que pode ser feito um acesso direto a um registro buscando pela chave secundária sem precisar realizar uma busca sequencial.

Na linguagem C os desafios com ponteiros e manipulação de arquivos é sempre desafiador, mas neste trabalho em específico conseguimos fazer as criações de partições, ler e escrever nelas de forma mais fácil, seja pela dificuldade do algoritmo de implementação na linguagem ser menos elevada que os demais da disciplina, ou, pelo fato de lidarmos com muitas implementações usando ponteiros e partições binárias na disciplina, assim, não tivemos grandes desafios com a linguagem e conseguimos implementar de uma maneira bem eficiente o algoritmo de Lefkowitz e ver sua funcionalidade em prática.