Trabalho Prático 0
Operações com matrizes alocadas dinamicamente

Luís Felipe Ramos Ferreira
2019022553

Universidade Federal de Minas Gerais
(UFMG) Belo Horizonte – MG – Brasil

lframos_ferreira@outlook.com

1. Introdução

A proposta do trabalho prático 0, da disciplina de Estruturas de Dados, foi implementar um programa para realização de operações entre matrizes alocadas dinamicamente. As operações em questão são as de somar duas matrizes, multiplicar duas matrizes e transpor uma matriz.

2. Implementação

O programa foi desenvolvido nas linguagens C e C++, compiladas pelo compilador G++ da GNU Compiler Collection.

2.1. Estrutura de Dados

O programa foi estruturado em torno de uma implementação de matriz representada por um vetor de vetores do tipo 'double'. Como a alocação das matrizes foi dinâmica, utilizou-se de ponteiros para definir estes vetores. Dessa maneira, a matriz foi definida como um ponteiro para ponteiro do tipo 'double'('double **').

A estrutura de dados foi definida em uma classe Matriz, que possui como atributos o vetor principal do tipo 'double **', representante da matriz, e seu número de linhas e colunas. Utilizar destas ferramentas facilitou o desenvolvimento do programa, uma vez que permitiu uma maior organização sobre a construção de instâncias de matrizes, assim como sobre a implementação das operações que serão realizadas com ela.

2.2. Classes

A implementação deste programa foi realizada com a criação de uma classe, nomeada de Matriz, a qual representa as matrizes com as quais serão realizadas as operações desejadas.

A classe Matriz possui três atributos privados, sendo eles dois inteiros que representam o número de linhas e o número de colunas da matriz. O outro, é um ponteiro para ponteiro do tipo ‘double’, o qual representa a matriz alocada dinamicamente desejada para o programa.

Além destes, a classe Matriz também possui alguns métodos úteis, os quais permitem acessar e manipular os atributos da estrutura de dados. Vale ressaltar, principalmente, o construtor e o destrutor da classe Matriz. O construtor recebe como parâmetro os dois inteiros representantes do número de linhas e colunas e, com isso, aloca dinamicamente a matriz com os tamanhos especificados. O destrutor, por sua vez, desaloca os espaços de memória criados para a matriz.

2.3. Funções de operações sobre matrizes

Para a realização das operações sobre as matrizes, foram implementadas três funções, uma para cada operação, que possuem como retorno um ponteiro para o objeto Matriz criado após a operação.

A primeira delas, nomeada ‘somaMatriz’, recebe como parâmetro a referência para os dois objetos do tipo Matriz que devem ser somados. A segunda, nomeada 'multiplicacaoMatriz', recebe, assim como a função de soma, a referência para dois objetos do tipo Matriz como argumento, os quais são multiplicados. A terceira, por sua vez, é a função utilizada para transpor uma matriz, denominada ‘transposicaoMatriz’, a qual recebe como argumento apenas a referência para a matriz que deve ser transposta.

2.4. Funções de inicialização, escrita e validação da matriz

Na implementação do programa, foram desenvolvidas também três funções para inicializar, escrever e validar uma matriz.

A função de inicialização ‘inicializaMatriz’ recebe como parâmetro uma string com o nome do arquivo de texto que possui os valores da matriz. Ela percorre o arquivo e armazena em um objeto do tipo Matriz os valores encontrados. Por fim, um ponteiro para esse objeto é retornado.

Para criar o arquivo resultante onde a matriz será escrita, a função ‘escreveMatriz’ recebe como parâmetro a referência para o objeto do tipo Matriz que deve ser escrito, assim como o nome do arquivo onde isso será feito. Na primeira linha são escritas as dimensões da matriz, e abaixo delas são escritos os valores da matriz segundo sua disposição lógica.

Além disso, foi desenvolvida a função 'valida Entrada', utilizada para checar se o arquivo de entrada que representa a matriz apresenta inconsistências. Ela é chamada apenas dentro da função ‘inicializaMatriz’ e, caso o arquivo apresente inconsistências, o programa é terminado. Ela recebe como argumento uma referência para o arquivo ‘ifstream’ criado pela abertura do arquivo.

3. Análise de Complexidade

3.1. Tempo

- **função ‘somaMatriz’:** essa função realiza algumas operações constantes de complexidade O(1). Além disso, ela possui dois laços aninhados. O laço externo percorre até o valor do número de linhas ‘i’ das matrizes sendo somadas, enquanto o laço interno percorre até o número de colunas ‘j’ das matrizes. Dessa forma, sua complexidade de tempo é O(ij).

- **função 'multiplicacaoMatriz':** essa função realiza algumas operações constantes de complexidade O(1). Além disso, ela possui três laços aninhados. O laço externo percorre até o valor do número de linhas ‘i’ da matriz resultante. O laço do meio percorre até o valor do número de colunas ‘j’ da matriz resultante. Por fim, o laço interno percorre até o valor do número de colunas da matriz 1 (ou número de linhas da matriz 2) ‘k’. Dessa forma, sua complexidade de tempo é O(ijk).

- **função ‘transposicaoMatriz’:** essa função realiza algumas operações constantes de complexidade O(1). Além disso, ela possui dois laços aninhados. O laço externo percorre até o valor do número de linhas ‘i’ da matriz que deve ser transposta. O laço externo percorre até o valor do número de colunas ‘j’ da matriz que deve ser transposta. Dessa forma, sua complexidade de tempo é O(ij).

- **função ‘validaEntrada’:** essa função realiza algumas operações constantes de complexidade O(1). Além disso, ela possui dois laços aninhados. O laço externo percorre até o valor do número de linhas ‘i’ da matriz no arquivo. O laço externo percorre até o valor do número de colunas ‘j’ da matriz no arquivo. Dessa forma, sua complexidade de tempo é O(ij).

- **função ‘inicializaMatriz’:** essa função realiza algumas operações constantes de complexidade O(1). Também, ela chama a função ‘validaEntrada’ dentro dela, que possui complexidade de tempo de O(ij). Além disso, ela possui dois laços aninhados. O laço externo percorre até o valor do número de linhas ‘i’ da matriz no arquivo. O laço externo percorre até o valor do número de colunas ‘j’ da matriz no arquivo. Dessa maneira, esses dois laços aninhados possuem uma complexidade de O(ij). Portanto, como O(ij) + O(ij) = O(ij), temos que a complexidade de tempo dessa função é de O(ij).

- **função ‘escreveMatriz’:** essa função realiza algumas operações constantes de complexidade O(1). Além disso, ela possui dois laços aninhados. O laço externo percorre até o valor do número de linhas ‘i’ da matriz que será escrita no arquivo. O laço externo percorre até o valor do número de

colunas ‘j’ da matriz que será escrita no arquivo. Dessa forma, sua complexidade de tempo é O(ij).

3.2. Espaço

- **função ‘somaMatriz’:** essa operação cria e retorna uma nova instância da classe Matriz, após realizar a soma entre das duas matrizes passadas como parâmetro, que é formada por estruturas de complexidade O(1), assim como por uma matriz alocada dinamicamente, formado por um vetor de ‘i’ (número de linhas da matriz) elementos que são vetores de ‘j’ (número de colunas da matriz) elementos. Assim, a complexidade de espaço da função é O(ij).

- **função 'multiplicacaoMatriz':** essa operação cria e retorna uma nova instância da classe Matriz, após realizar a multiplicação entre das duas matrizes passadas como parâmetro, que é formada por estruturas de complexidade O(1), assim como por uma matriz alocada dinamicamente, formado por um vetor de ‘i’ (número de linhas da matriz) elementos que são vetores de ‘j’ (número de colunas da matriz) elementos. Assim, a complexidade de espaço da função é O(ij).

- **função ‘transposicaoMatriz’:** essa operação cria e retorna uma nova instância da classe Matriz, após realizar a transposição da matriz passa como parâmetro, que é formada por estruturas de complexidade O(1), assim como por uma matriz alocada dinamicamente, formado por um vetor de ‘i’ (número de linhas da matriz) elementos que são vetores de ‘j’ (número de colunas da matriz) elementos. Assim, a complexidade de espaço da função é O(ij).

- **função ‘validaEntrada’:** essa função realiza todas as operações considerando estruturas auxiliares unitárias O(1). Logo, sua complexidade de espaço é O(1).

- **função ‘inicializaMatriz’:** essa operação cria e retorna uma nova instância da classe Matriz, após ler seus valores no arquivo de entrada, que é formada por estruturas de complexidade O(1), assim como por uma matriz alocada dinamicamente, formado por um vetor de ‘i’ (número de linhas da matriz) elementos que são vetores de ‘j’ (número de colunas da matriz) elementos. Assim, a complexidade de espaço da função é O(ij).

- **função ‘escreveMatriz’:** essa função realiza todas as operações considerando estruturas auxiliares unitárias O(1). Logo, sua complexidade de espaço é O(1).

4. Estratégias de robustez

Para tratar falhas e tornar o código mais robusto, utilizou-se da biblioteca disponibilizada ‘msgassert.h’. Essa biblioteca define duas ‘macros’ para tratar os possíveis erros encontrados no uso das funções do programa.

A biblioteca foi utilizada em basicamente todas as funções e métodos implementados no código, uma vez que ela permitiu tratar, com facilidade, erros e inconsistências nos parâmetros passados para elas.

5. Conclusão

Este trabalho teve como propósito resolver o problema de realização de operações sobre matrizes alocadas dinamicamente. Uma abordagem orientada a objetos foi adotada e, por isso, a linguagem escolhida para o desenvolvimento da matriz e das operações sobre ela foi C++.

A matriz foi desenvolvida como um vetor de vetores do tipo ‘double’, e essa estrutura foi posta como um atributo de uma classe Matriz, que possuía também o número de linhas e o número de colunas dessa matriz. Além disso, a classe também possui métodos que facilitam a manipulação da matriz.

Também foram criadas funções para realizar as operações de soma, multiplicação e transposição sobre as matrizes, assim como funções para ler a matriz do arquivo de entrada, validar esse arquivo, bem como para escrever a matriz resultante de alguma operação em um arquivo final.

Pode-se verificar que a solução adotada foi importante para que o trabalho funcionasse bem. Uma abordagem orientada a objetos permite manipular e controlar melhor como as matrizes são inicializadas e destruídas, da mesma maneira que permite uma melhor visualização de como a estrutura de dados funciona.

Por meio da implementação do trabalho, foi possível trabalhar e enraizar conceitos importantes de programação orientada a objetos, como a abstração. Destaca-se também a revisão e utilização de expressões regulares, as quais são importantíssimas para validação de entradas. Por fim, outro conteúdo praticado foi o de alocação dinâmica de memória, forma como as matrizes foram desenvolvidas.

Os principais desafios encontrados durante a implementação do trabalho foram em relação à liberação de memória após a utilização das matrizes. Também é importante ressaltar que a validação da entrada dos arquivos que continham a matriz foi um trabalho que exigiu muita refatoração, uma vez que a disposição dos valores no arquivo, seguindo o modelo de uma matriz, exigiam uma atenção muito detalhada.

6. Referências bibliográficas

CPLUSPLUS.ifstream.cplusplus.Disponível em: <https://www.cplusplus.com/reference/fstream/ifstream/>. Acesso em: 08/08/2021.

SOFTWARETESTINGHELP.C++ Errors: Undefined Reference, Unresolved External Symbol Etc. SoftwareTestingHelp. Disponível em: <https://www.softwaretestinghelp.com/cpp-errors/>. Acesso em: 10/08/2021.

GREATLEARNINGTEAM. Operator Overloading in C++ | What is Operator Overloading.mygreatlearning. Disponível em: <https://www.mygreatlearning.com/blog/operator-overloading-in-cpp/>. Acesso em: 11/08/2021.

GEEKSFORGEEKS. Core Dump (Segmentation fault) in C/C++.geeksforgeeks. Disponível em: <https://www.geeksforgeeks.org/core-dump-segmentation-fault-c-cpp/>. Acesso em: 08/08/2021.

DAEMONIO. Usando As Funções getopt() e getopt_long() Em C. daemoniolabs. Disponível em: <https://daemoniolabs.wordpress.com/2011/10/07/usando-com-as-funcoes-getopt-e-getopt_long-em-c/>. Acesso em: 05/08/2021.

MIZRAHI, Victorine Viviane. Treinamento em linguagem C. 2ª Edição. Editora Pearson.

DROZDEK, Adam. Estrutura de dados e algoritmos em c++. 4ªEdição. Editora Cengage Learning.

7. Compilação e execução

- Acesse o diretório TP
- No terminal, utilize comando make para compilar os arquivos e gerar o 'target' do programa.
- Para realizar as operações, digite no terminal ‘bin/programa’ seguido das opções de linha de comando. As opções são as seguintes:
  -s (somar matrizes)
  -m (multiplicar matrizes)
  -t (transpor matriz)
  -p <arquivo> (arquivo de registro de acesso)
  -l (registrar acessos à memória)
  -1 <arquivo> (arquivo da primeira matriz)
  -2 <arquivo> (arquivo da segunda matriz)
  -o <arquivo> (arquivo para escrita da matriz resultante)
- Para apagar os arquivos ‘object’ e o ‘target’, utilize no terminal o comando make clean.