Trabalho Prático 0

Operações com Matrizes Dinamicamente Alocadas

Ítalo Dell'Areti

Universidade Federal de Minas Gerais (UFMG)

Belo Horizonte – MG – Brasil

italodellareti@ufmg.br

1. Introdução

O problema proposto é implementação das operações soma, multiplicação e transposição em matrizes dinamicamente alocadas, com base em uma implementação já realizada de forma estática fornecida pelos professores. Portanto, utilizando dinamicamente podemos usufruir da quantidade necessária de recursos, reduzindo a quantidade de recursos que outras estruturas de dados gastariam desnecessariamente. Tendo em vista isso, o este documente possui como objetivo explicitar o funcionamento desta implementação.

2. Implementação

O programa foi desenvolvido na linguagem C, compilada pelo compilador G++ da GNU Compiler Collection. Ademais, o programa foi produzido e testado em um ambiente Linux distribuição Ubuntu 20.04 LTS.

2.1 - Estrutura de Dados

A implementação do programa possui como base o tipo abstrato de dados de uma matriz. Esta matriz é alocada dinamicamente, pois desta forma podemos atribuir valores de linhas e colunas variáveis, a tornando de tamanho manipulável e sem precisar de se estimar um valor fixo ou predeterminado.

Neste tipo abstrato de dados foi se utilizado uma estrutura chamada mat, a qual possui a propriedade de agrupar variáveis sob um mesmo tipo de dado. Como já preestabelecido pela implementação anterior, estamos utilizando as variáveis "tamx" e "tamy" na definição das dimensões da matriz e a variável "id" para ser utilizada pela biblioteca "Memlog" para testes de performasse e localidade. Como estes valores vão ser armazenados em memória, conseguimos eliminar algumas operações de custo linear para a obtenção do tamanho da matriz.

2.2 - Organização

A implementação do programa foi realizada utilizando apenas as três classes fornecidas previamente. A classe principal é a **mat.c**, na qual possui as funções de criação das matrizes, suas operações, escrita e leitura dos arquivos texto onde são armazenadas as matrizes. Além disso, as classes **matop.c** e **memlog.c** são responsáveis respetivamente pelo controle de fluxo do programa e criação de um histórico de escrita e leitura na memória.

2.3 - Operações

As operações realizadas no programa são a **soma, multiplicação e transposição.** Os valores para a realização destas funções são obtidos dos arquivos texto m1 e m2, que estão na pasta raiz do programa. A partir dos dados fornecidos pelas matrizes, é realizado a operação desejada e o resultado é salvo em outro arquivo.

Na soma precisamos dos dois arquivos texto, m1 e m2, e as matrizes fornecidas precisam possuir as mesmas dimensões para realizar a tarefa. A operação Multiplicação também precisamos dos dois arquivos texto, m1 e m2, e as matrizes fornecidas precisão possuir a seguinte característica: a primeira matriz deve possuir número de linhas iguais ao número de colunas da segunda matriz. E em sua última operação, a transposição, é necessário apenas uma matriz de qualquer tamanho.

3. Análise Complexidade

A seguir vamos analisar algumas das principais funções do programa, tendo em vista que a letra **m** significa o número de linhas da matriz e a letra **n** o número de colunas da matriz.

3.1 - Soma

A função **opcaoDeSoma**, localizada no arquivo **matop.c**, é a responsável pela operação de soma entre as duas matrizes fornecidas pelos arquivos texto m1 e m2. Sendo assim, para a realização desta operação ela deve alocar três matrizes, sendo elas as matrizes que vão ser somadas e a matriz solução. Esta tarefa é realizada pela função **cria_matriz** que é chamada nesta função e equivalente assintótica de **Θ(m)**.

Além disto, a função chama outra para a realização da soma das matrizes, a **soma_matrizes**, que é a real responsável pelo somatório e possui equivalência assintótica de **Θ(mxn)**. Ademais, outras funções que são executadas juntamente como **criar_arquivo_matriz** e **destroi_matrizes** as quais possuem como equivalentes assintóticos, **Θ(mxn)** e **Θ(m)** respectivamente. Ao final, encontramos o equivalente assintótico para a função **opcaoDeSoma** de **Θ (mxn)**.

3.2 - Transposição

A função **opcaoDeTransposicao**, localizada no arquivo **matop.c**, é a responsável pela operação de transposição de qualquer matriz fornecida. Sendo assim, para a realização desta operação ela deve alocar duas matrizes, através das funções **criar_matriz_de_arquivo** e **cria_matriz** as quais possuem como equivalente assintótico **Θ(mxn**). A função que realmente realiza a transposição é a função **transpoe_matriz** a qual possui como equivalente assintótico **Θ(mxn)**.

Outras funções que também são executadas juntamente como **criar_matriz_arquivo** e **destroi_matriz**, as quais possuem como equivalentes assintóticos **Θ (mxn)** e **Θ (m)**, respectivamente. Ao final, encontramos o equivalente assintótico para a função **opcaoDeTransposicao** de **Θ (mxn)**.

3.3 - Multiplicação

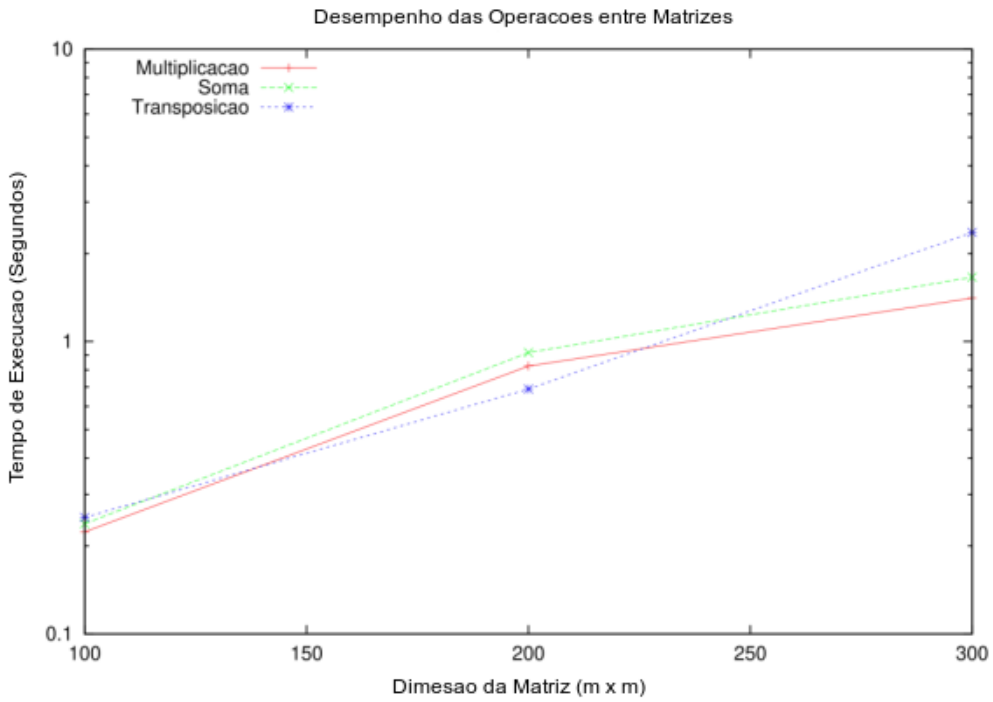
A função **opcaoDeTransposicao**, localizada no arquivo **matop.c**, é a é a responsável pela operação de multiplicação das matrizes. Dessa forma, para a realização desta operação ela deve alocar três matrizes, através das funções **criar_matriz_de_arquivo** e **cria_matriz** as quais possuem como equivalente assintótico **Θ(mxn**). Quem de fato realiza a multiplicação entre as matrizes é a função **multiplica_matrizes**, a qual possui como equivalente assintótico de **Θ (m^2 x n).**

Além destas funções, outras que fazem parte são **criar_matriz_arquivo** e **destroi_matriz** as quais possuem equivalentes assintóticos de **Θ (mxn)** e **Θ (m)**. Logo ao final podemos concluir que esta função possui equivalente assintótico de **Θ (m^2 x n).**

4. Desempenho

4.1 - Tempo

Conforme o previsto anteriormente, o tempo de execução depende diretamente das dimensões das matrizes. Evidentemente que dependendo da operação realizada, o seu gráfico varia com a sua complexidade.



4.2 - Espaço

As matrizes são criadas a partir da função criar_matriz_de_arquivo que aloca dinamicamicamente o tamanho e elementos da matriz que foi fornecida. O que está matriz utiliza é a outra função cria_matriz (como equivalente assintótico de espaço Θ (mxn)) e utiliza dois laços que são realizados na função principal para a leitura e verificação do conteúdo do arquivo da matriz. Essas iterações possuem como equivalente assintótico de espaço Θ (mxn). Sendo assim, ao calcular o equivalente assintótico ele possui complexidade de Θ (mxn).

5. Robustez

Algumas tomadas de decisão foram necessárias para garantir o bom funcionamento e a consistência dos dados. Portanto, em todas as operações entre as matrizes e suas leituras, são necessários atender certos requisitos matemáticos para a realização do cálculo.

As matrizes que são passadas para realizar as operações não podem estar vazias e um alerta é emitido ao usuário. Em relação as operações somam, multiplicação e transposição elas devem respeitar as condições pré-requeridas. Nas somas as ordens das matrizes devem ser iguais e na multiplicação é necessário que o número de colunas da primeira matriz seja igual ao número de linhas da segunda matriz. Além disso, ao final das operações, desalocamos o espaço utilizado e destruímos as matrizes.

6. Conclusão

No final deste trabalho, podemos inferir que este possui o propósito de abordar e relembrar conceitos das matérias anteriores PDS1 e PDS2 e trazer uma nova visão para nós que desenvolvemos estes programas, um foco maior no desempenho do programa. Nas matérias anteriores, o foco nos recursos de acesso e memórias não eram tão evidentes quanto podemos experimentar nesta matéria e, em certa parte neste trabalho.

Podemos levar como aprendizado deste problema como o impacto desordenado da memória e a má alocação dos dados impactam severamente o desempenho da aplicação, gerando grandes problemas em seu custo.

Bibliografia

BACKES, A. R. (2018). *Linguagem C - Completa e Descomplicada*

Ziviani, N., Projeto de Algoritmos com Implementações em Pascal e C

Instruções para compilação e execução

Para se executar o programa basta dar um comando **make** para executar tudo. Mas se quiser pode estar executando partes exclusivas. São algumas regras

make all - compila tudo

make mem - gera arquivos de registro de acesso

make prof - faz medidas de tempo de desempenho

make gprof - ilustra uso do gprof para depuração

make clean - remove objetos e executável

Ao se utilizar os comandos make prof e make gprof você **terá que descomentar no Makefile** estas operações, pois foram utilizadas apenas para medir o desempenho e localidade do programa. Estes comandos, não são importantes para a realização das operações

Ademais, as operações são representadas por flags e elas são a soma(-s), multiplicação(-m), transposição(-t). Além disso, todos os arquivos de matriz que são geradas e que são obtidas através dos **arquivos de texto devem estar no diretório raiz da pasta do programa.**

Os resultados das operações vão aparecer na pasta raiz do projeto através dos arquivos texto res-soma (resultado da soma de duas matrizes), res-mult (resultado da multiplicação de duas matrizes) e res-transp (resultado da transposição de uma matriz).