

Algoritmos e Estrutura de Dados III

Trabalho Prático 0 - Matrizes 3D

Lucas Pereira Monteiro¹

¹Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)

lucasmonteiro@dcc.ufmg.br

Resumo. O objetivo deste trabalho é a manipulação de Matrizes Tridimensionais. Tais manipulações podem ser de soma $C = A + B$, o que implica em somar o elemento correspondente nas duas matrizes A e B tal que $c_{ijk} = a_{ijk} + b_{ijk}$, ou de subtração $c_{ijk} = a_{ijk} - b_{ijk}$. O programa será executado para múltiplas instâncias $N, \forall N \geq 0$ e deverá salvar o resultado das operações sobre as matrizes em um arquivo de saída previamente especificado.

1. Introdução

A manipulação de matrizes e ponteiros é um dos aspectos fundamentais em Ciência da Computação. Para se fazer um melhor uso das linguagens estruturadas¹, é importante utilizar os Tipos Abstratos de Dados, os quais encapsulam os detalhes de implementação.

O objetivo desse trabalho é implementar uma Matriz Tridimensional, a qual é representada por $M_{ijk, \forall (i,j,k) \geq 1}$. A variável i é igual ao número de planos, j é igual ao número de linhas e, por fim, k é igual ao número de colunas. Como saberemos as dimensões (i, j, k) da matriz M somente em tempo de execução, a matriz será alocada dinamicamente.

Ao final do trabalho prático, espera-se praticar os conceitos básicos e necessários de programação utilizados no mesmo. São eles: manipulação de arquivos, manipulação de vetores, alocação dinâmica e encapsulamento de dados. Além da implementação e manipulação de TAD's e ponteiros, os quais são extremamente necessários para um bom desenvolvimento em praticamente qualquer trabalho utilizando-se a linguagem C.

1.1. Descrição superficial do problema e da sua resolução

A empresa *Matrix3D* precisa realizar operações aritméticas sobre matrizes tridimensionais. A operação de soma de matrizes $C = A + B$ implica em somar o elemento correspondente nas duas matrizes A e B , ou seja, temos $c_{ijk} = a_{ijk} + b_{ijk}$. As matrizes devem ser alocadas e desalocadas dinamicamente com as funções *malloc()* e *free()*. O problema consiste em realizar sequências de operações de soma e subtração em matrizes tridimensionais, sendo que o número de matrizes, as suas 3 dimensões e as operações são definidas na entrada.

A resolução superficial para o problema é: Dado o número de instâncias N que serão executadas, o programa será executado N vezes. Para cada vez que o programa executar ele vai ler as dimensões (i, j, k) da matriz M e alocar uma matriz com tais dimensões. Posteriormente, será feita a leitura dos dados e os mesmos serão armazenados

¹http://en.wikipedia.org/wiki/Programming_language

na matriz. Então será lido o operador que pode ser $+$ ou $-$ ou $=$. Se for $+$ (soma) o programa somará os elementos da matriz atual M com a matriz que está no arquivo. Se o operador for $-$ será realizado uma subtração. Por fim, se o operador for $=$, o programa termina a execução para uma dada matriz M , escreve seus dados em um arquivo passado de saída passado na entrada, desaloca a matriz e verifica se há uma nova instância para executar. Se sim, repete o processo acima, se não, termina a execução.

2. Implementação

2.1. Estruturas e tipos utilizados

O número de instâncias do programa é armazenado em um tipo inteiro *NumInstancias*.

A matriz M é do tipo ****double*, o qual foi encapsulado em um tipo *Matriz3D* para uma maior legibilidade do código.

A estrutura utilizada para armazenar os dados da matriz é chamada *DadosMatriz*, a qual contém as informações sobre suas dimensões (i, j, k) .

```
1 typedef int NumeroInstancias ;
2
3 typedef double*** Matriz3D ;
4
5 typedef struct DadosMatriz {
6     int X; // Numero de planos
7     int Y; // Numero de linhas
8     int Z; // Numero de columnas
9 } DadosMatriz ;
```

Estruturas e Tipos

2.2. Decisões e percurso de implementação

A matriz M é do tipo *Matriz3D*, o que indica que ela é uma matriz tridimensional (como estava da especificação do trabalho). Portanto, para a indexação de qualquer elemento da matriz, é necessário utilizar $M[i][j][k]$. Logo abaixo, há um pseudo-código de como o programa principal funciona.

Algorithm 1 Manipulação de matrizes tridimensionais

Require: Arquivo de entrada *arqEntrada*, Arquivo de saída *arqSaida*

- 1: **while** Não terminar de ler o número de instâncias **do**
- 2: Lê as dimensões (i, j, k) da matriz (M).
- 3: Aloca a matriz (M) dinamicamente
- 4: Lê o operador (O)
- 5: **while** (O) for diferente '=' **do**
- 6: Realiza as operações de acordo com o operador lido
- 7: Desaloca a matriz M .

end

2.3. Funções principais e análise de complexidade

- `Matriz3D CriaMatriz3D (int numPlanos, int numLinhas, int numColunas)`. Dado que a matriz criada será em função do número de planos x , número de linhas y , e número de colunas z a sua complexidade de tempo é $O(x*y*z)$ e o espaço utilizado para armazenar seus elementos é da forma $x*y*z$.
- `void ImprimeMatriz (FILE *arqSaida, Matriz3D matriz, int numPlanos, int numLinhas, int numColunas)`. A função que faz a impressão da matriz M no arquivo de saída tem complexidade $O(x*y*z)$, pois é necessário percorrer cada elemento da matriz para imprimó-lo no arquivo.
- `void DesalocaMatriz (Matriz3D matriz, int numPlanos, int numLinhas, int numColunas)`. Para percorrer a matriz e desalocar seus elementos temos a complexidade de $O(x*y*z)$, pois é necessário acessar cada elemento para então desalocá-lo.

3. Organização do código, detalhes técnicos e execução

3.1. Organização

O código enviado foi dividido em vários arquivos, para que fosse realizada a modularização do mesmo. São eles:

- *io.c/h*: Cuida da parte de manipulação dos arquivos.
- *matriz.c/h*: Contém as operações para manipulação de matrizes.
- *main.c*: Arquivo principal, que faz o programa executar.

3.2. Detalhes técnicos

- IDE de desenvolvimento: Codeblocks 12.11;
- Processador: Core i5;
- Memória RAM: 4GB;
- Sistema Operacional: Linux Mint 15.

3.3. Compilação e execução

O código fonte deve ser compilado e o executável gerado através do compilador GCC através do *Makefile*.

make

Após o executável ter sido gerado, o mesmo já está apto a executar o programa. O executável necessita de dois parâmetros, são eles:

- 1 Arquivo de entrada *input.txt* com as instâncias do problema.
- 2 Arquivo de saída *output.txt* que conterà o resultado gerado pelo programa.

Para rodar o executável, é necessário digitar o seguinte comando no terminal.

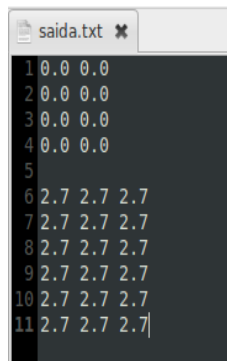
./tp0 input.txt output.txt

4. Testes

Foram realizados 2 testes para conferir a saída do programa. O primeiro teste foi realizado com o exemplo que estava na documentação e produziu uma saída igual a que estava na especificação do trabalho. O segundo teste foi realizado usando 0 instâncias, e produziu um arquivo vazio. Um terceiro teste foi realizado para verificar o tempo de execução para um número de instâncias $N = (18, 36, 72, \dots, 147456)$.

4.1. Arquivo de entrada da documentação

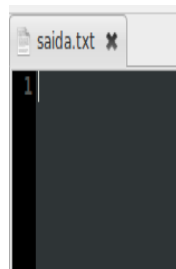
O arquivo de entrada contém duas matrizes, sendo uma matriz $M_{1,2,2}$ e uma matriz $M_{2,3,3}$. Como especificado na documentação a saída foi de acordo com o esperado.



```
1 0.0 0.0
2 0.0 0.0
3 0.0 0.0
4 0.0 0.0
5
6 2.7 2.7 2.7
7 2.7 2.7 2.7
8 2.7 2.7 2.7
9 2.7 2.7 2.7
10 2.7 2.7 2.7
11 2.7 2.7 2.7
```

4.2. Arquivo de entrada com zero instâncias

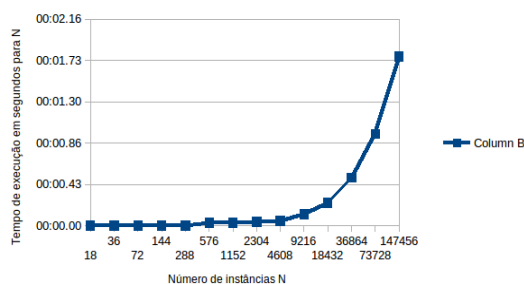
O arquivo de saída foi gerado corretamente, resultando em um arquivo vazio.



```
1
```

4.3. Tempo de execução

Foi realizado um teste em relação ao número de instâncias a serem executadas. Notamos que para a partir de 9000 instâncias o custo fica linear com uma constante multiplicativa 2.



5. Conclusão

O entendimento do problema proposto ocorreu sem maiores problemas, o que levou a um começo quase imediato da implementação da solução proposta. O problema que gastou mais tempo para ser compreendido e implementado foi o *Makefile*, o qual não tinha muitos conhecimentos.

Após tentar fazer algumas alocações dinâmicas de uma matriz tridimensional (****double*) consegui chegar ao resultado esperado. No começo tive alguns problemas com a manipulação de ponteiros e a alocação dos mesmos. A parte de lógica (como percorrer e preencher uma matriz tridimensional sendo lida do arquivo) também foi um pouco difícil e demorada. Como precisava de percorrer a matriz na ordem $((0, 0, 0), (0, 0, 1), \dots, (1, 0, 0), \dots)$, aprendi a usar o *dubbuger* do Codeblocks, o qual me foi bem útil para mostrar quais as posições que eu estava acessando.

Após a finalização do trabalho, fiquei satisfeito com meu desempenho, visto que pude praticar e me aprofundar nos conceitos de alocação dinâmica, manipulação de arquivos, manipulação de matrizes, encapsulamento e *Makefile*. Portanto, imagino que os resultados ficaram dentro do esperado, uma vez que pude praticar bastante os conceitos da linguagem C.

Por fim, acredito que o trabalho proposto cumpriu seu objetivo, de proporcionar ao aluno maior prática e aprendizagem da linguagem C.

6. Referências

- 1 Ziviani, N. (2004). Projeto de Algoritmos com Implementações em C e Pascal. Editora Thomson.