

**Trabalho prático 1**

**Matriz Esparsa**

# UFV – Campus Florestal

|  |  |
| --- | --- |
| **Professor:** | Thais Regina de Moura Braga Silva |
| **Disciplina:** | Algoritmos e Estrutura de Dados I – CCF 211 |
| **Aluno:** | Lucas Takeshi Moreira Chang - 2665 |
| **Aluno:** | Marcos Túlio Rodrigues Almeida - 3504 |
| **Aluno:** | Victor Hugo Rezende dos Santos - 3510 |

**Florestal, 26 de setembro de 2019**

### ÍNDICE

1. [**INTRODUÇÃO**](https://docs.google.com/document/d/1Wd9ejZjaOe27rexoqnbd_E9pf7tmQi5GVN-Ivoq_bCc/edit#heading%3Dh.gjdgxs)
2. [**DESENVOLVIMENTO**](https://docs.google.com/document/d/1Wd9ejZjaOe27rexoqnbd_E9pf7tmQi5GVN-Ivoq_bCc/edit#heading%3Dh.30j0zll)
3. [**CONSIDERAÇÕES**](https://docs.google.com/document/d/1Wd9ejZjaOe27rexoqnbd_E9pf7tmQi5GVN-Ivoq_bCc/edit#heading%3Dh.1fob9te) **FINAIS**
4. [**BIBLIOGRAFIA**](https://docs.google.com/document/d/1Wd9ejZjaOe27rexoqnbd_E9pf7tmQi5GVN-Ivoq_bCc/edit#heading%3Dh.3znysh7)
5. **INTRODUÇÃO**

Após, em sala de aula e laboratório, aprendermos os conceitos de TAD, Listas, alocação dinâmica de memória e listas encadeadas, foi passado o primeiro trabalho prático da disciplina Algoritmos e Estruturas de dados 1, com o objetivo de concretizar esses conhecimentos implementando uma Matriz Esparsa encadeada circular seguindo os critérios cobrados no roteiro do trabalho pelo docente.

Nesse sentido, utilizamos a definição de Matriz Esparsa que é uma matriz na qual a maioria da posições são preenchidas por zero. Para essas matrizes, podemos economizar um espaço significativo de memória se apenas os termos diferentes de zero forem armazenados, poupando gasto de memória e até mesmo de processamento, caso queiramos percorrê-la.

1. **DESENVOLVIMENTO**

* **Estrutura do TAD**

Para o desenvolvimento de tal matriz, inicialmente foram criados três arquivos, sendo dois deles responsáveis pelo **TAD\_Matriz\_Esparsa** (TAD\_Matriz\_Esparsa.h e TAD\_Matriz\_Esparsa.h) e o main.c, responsável pelos testes e execuções do TAD.

No TAD\_Matriz\_Esparsa.h estão as declarações da Estrutura do TAD e também as funções que este executa, a estrutura da matriz se dá da seguinte forma:

typedef struct{

    int i, j;

    TLista linha, coluna;

}TMatriz;

**Figura 1: Estrutura da matriz**

Criamos um struct ***TMatriz*** com dois campos inteiros: *i* (quantidade de linhas) e *j* (quantidade de colunas) e também dois ponteiros para **TLista**: *linha* e *coluna*. Estes campos serão exatamente nossas duas listas referentes a linha e coluna, respetivamente, ou seja, cada elemento da minha matriz pertencerá, ao mesmo tempo a duas listas. ***TLista*** é da seguinte forma:

typedef struct{

    Apontador pPrimeiro;

}TLista;

**Figura 2: Estrutura da lista**

\**Apontador* é do tipo ponteiro para struct Celula e sua estrutura foi dada na descrição do trabalho.

typedef struct Celula\* Apontador;

typedef struct Celula{

    Apontador direita, abaixo;

    int linha, coluna;

    double valor;

}TCelula;

**Figura 3: Estrutura da Célula**

Após definida a estrutura da nossa matriz, é hora de implementarmos as funcionalidades por meio de subprogramas (chamados também de funções) que conseguem efetuar operações com a matriz. Na descrição do trabalho é especificado a criação das seguintes funções: ***void ImprimeMatriz(), void leArquivo(), void InsereMatriz().*** A função *InsereMatriz* é a função responsável por criar células das listas que formam a matriz, a célula criada recebe valores da função *LeMatriz,* quelê de algum arquivo de entrada a quantidade de linhas e de colunas (primeira linha do arquivo) e a tripla (i(valor da linha), j(valor da coluna), valor(referente às linhas e colunas passadas)) nas demais linhas do arquivo.

A função *leArquivo* é da seguinte forma:

void leArquivo(TMatriz\* pMatriz)

{

    FILE \*arq;

    int tamLinha, tamColuna;

    int lin, col;

    double val;

    arq = fopen("teste.txt", "r");

    //le a primeira linha com o tamanho da matriz

    fscanf(arq, "%d, %d\n", &tamLinha, &tamColuna);

    printf("tamlinha e tamcoluna: %d %d", tamLinha, tamColuna);

    InicializaMatriz(pMatriz,tamLinha+1,tamColuna+1);

    while (fscanf(arq, "%d, %d, %lf", &lin, &col, &val) != EOF)

    {

        InsereMatriz(pMatriz,lin, col, val);

    }

}

**Figura 4: Estrutura da função *leArquivo***

Esta função abre um arquivo e retira de lá valores, por meio da função fscanf(), que serão atribuídos a nossa matriz. Como dito anteriormente, a primeira linha contém a quantidade de linhas e colunas e as demais contém a tripla (i, j, valor). A medida que o arquivo é lido, os valores são atribuídos à função *InsereMatriz*, vale lembrar que nesta função está também a função *InicializaMatriz*, pois é a partir da leitura arquivo que a matriz será criada.

Porém, antes de implementar tais funções, devemos inicializar a matriz, ou seja, criar as células cabeça referente a lista linha e também as células cabeça da lista coluna. A função  *InicializaMatriz* é da matriz é da seguinte forma:

void InicializaMatriz(TMatriz \*pMatriz, int QuantidadeLinhas, int QuantidadeColunas){

    pMatriz->coluna.pPrimeiro = (Apontador) malloc(sizeof(TCelula));

    pMatriz->linha.pPrimeiro = pMatriz->coluna.pPrimeiro;

    pMatriz->coluna.pPrimeiro->coluna = pMatriz->linha.pPrimeiro->linha = -1;

    pMatriz->coluna.pPrimeiro->direita = pMatriz->linha.pPrimeiro->abaixo = pMatriz->coluna.pPrimeiro;

    pMatriz->i = QuantidadeLinhas;

    pMatriz->j = QuantidadeColunas;

    InicializaLinha(&pMatriz->linha, QuantidadeLinhas);

    InicializaColuna(&pMatriz->coluna, QuantidadeColunas);

**Figura 5: Estrutura da função *InicializaMatriz***

Esta função, em primeiro instate, aloca um espaço no heap para a célula cabeça da lista *linha* e também o mesmo para a lista *coluna*. Esta alocação será chamada de Célula Cabeça Mestra, pois é a única célula cabeça pertencente às duas listas e também para facilitar o entendimento da explicação do código. Após isto, é atribuido para esta, nos campos *linha e coluna* o valor -1 e para os campos *abaixo e direta*, que são apontadores para ***Tcelula,*** é atribuido ela mesma, para torná-la cíclica. Após isto, o campo *i e j* da matriz são preenchidos com a *QuantidadeLinhas e QuantidadeColunas.* As funções inseridas *InicializaLinha e InicializaColuna* são funções que criam uma lista linha e uma lista coluna a partir da Célula Mestra.

Após inicializada a matriz, é hora de inserir células na matriz, tal inserção se dá por meio da função insere, que é da seguinte forma:

void InsereMatriz(TMatriz \*pMatriz, int i, int j, double valor)

{

    TCelula \*CeCabecaL, \*CeCabecaC, \*pAUX;

    CeCabecaC = PercorreColuna(&pMatriz->coluna, j);

    CeCabecaL = PercorreLinha(&pMatriz->linha, i);

    pAUX = (TCelula \*)malloc(sizeof(TCelula));

    pAUX->linha = i;

    pAUX->coluna = j;

    pAUX->valor = valor;

    inserirListaColuna(CeCabecaC, pAUX);

    inserirListaLinha(CeCabecaL, pAUX);

}

**Figura 6: Estrutura da função *InsereMatriz***

Esta função, inicialmente percorre a lista *coluna*, por meio da função *PercorreColuna*, que retorna a célula cabeça referente à posição da linha que queremos inserir, que será passado como parâmetro para tal função. O mesmo ocorre com a lista *linha* por meio da função *PercorreLinha.* As estruturas de cada são diferentes, já que uma trata a linha e a outra a coluna, mas suas lógicas são idênticas, mudando apenas os apontadores. Após verificada a posição referente à lista *linha* e também a posição referente à lista *coluna*, já sabemos onde inserir nosso item em cada uma das listas e tais inserções serão dadas pelas funções *inserirListaLinha e inserirListaColuna.*

Neste estágio, a matriz já está inicializada e criada, por fim, nos resta imprimí-la. A função *ImprimeMatriz* é da seguinte forma:

void ImprimeMatriz (TMatriz \*pMatriz){

    TCelula \*pAUX, \*Cabeca;

    double matriz[pMatriz->i][pMatriz->j];

    int i, j;

    Cabeca = pMatriz->linha.pPrimeiro->abaixo;

    for(i=1;i<(pMatriz->i);i++){

        for(j=1;j<(pMatriz->j);j++){

            matriz[i][j] = 0;

        }

    }

    while (Cabeca->abaixo != pMatriz->linha.pPrimeiro){

        pAUX = Cabeca->direita;

        while(pAUX != Cabeca){

            matriz[pAUX->linha][pAUX->coluna] = pAUX->valor;

            pAUX = pAUX->direita;

        }

        Cabeca = Cabeca->abaixo;

    }

    for(i=1;i<(pMatriz->i);i++){

        for(j=1;j<(pMatriz->j);j++){

            printf("%.0lf ",matriz[i][j]);

        }

        printf("\n");

    }

}

**Figura 7: Estrutura da função *ImprimeMatriz***

* Main.c

Por fim, no main.c, há a chamada das funções. Foi criado uma estrutura switch(case) para simular um menu interativo que possui as opções: 1- Ler Arquivo, 2- Imprimir a matriz e 0- Sair. Após o usuário digitar uma entrada válida, o programa chama as respectivas funções e executa o comando dado pelo usuário.

* TAD\_PRODUTOS

A última parte do desenvolvimento deste trabalho envolve aplicar o TAD\_Matriz\_Esparsa para o cenário em que temos i clientes x j produtos, ou seja, cada linha i da matriz refere a um cliente, assim como cada coluna refere a um produto. A entrada de valores para a função se dá também por meio de um arquivo .txt e se dá da seguinte forma: i j(primeira linha) e data(dd/mm/aa) quantidadeProdutos. Após a obtenção da entrada do arquivo e da inserção destes intenso na matriz, pode-se efetuar operações do tipo: imprime, quantidadeProdutosCliente e quantidadeClientesProduto. Infelizmente após tentarmos implementar a segunda parte, devido algumas dificuldades de implementação não conseguimos concluir essa parte.

1. **CONSIDERAÇÕES FINAIS**

Esse trabalho foi muito importante para nós, integrantes do grupo, colocarmos toda a teoria apreendida em sala de aula em prática. Foram demandadas muitas horas para a implementação de todas as funções e estruturas de dados necessários para concluir este trabalho prático.

Sobre tudo, foi uma experiência bastante produtiva e positiva para o desenvolvimento do grupo na disciplina, tendo em vista que a prática ajuda no aperfeiçoamento do conhecimento, como afirma Lênin, “A teoria sem a prática de nada vale, a prática sem a teoria é cega.”

1. **BIBLIOGRAFIA**

ZIVIANI, N. Projeto de Algoritmos com implementação em PASCAL e C. 3ª ed. CENGAGE, 2017.

Prática Administração, 2011. Disponível em: <http://praticaadministrativa.blogspot.com/2011/03/departamentalizacao-da-teoria.html>. Acesso em 31 de agosto de 2019.