

**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. [**CONCLUSÃO**](https://docs.google.com/document/d/1Wd9ejZjaOe27rexoqnbd_E9pf7tmQi5GVN-Ivoq_bCc/edit#heading%3Dh.1fob9te)
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 dado o objetivo de concretizar os mesmos por meio da implementação de uma Matriz Esparsa.

Uma matriz Esparsa 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.

Foi dado também, por meio de um arquivo disponibilizado na plataforma PVANet a descrição do trabalho, que diz que cada coluna da matriz será representada por uma lista linear circular com uma célula cabeça e o mesmo para as linhas, e é a partir de tal ideia, que a matriz será implementada.

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;

    TItem Item;

}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 LeMatriz(), 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. 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 responsável pela inicialização da matriz é da seguinte forma:

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

    pMatriz->coluna->pPrimeiro = pMatriz->linha->pPrimeiro = (TCelula\*)malloc(sizeof(TCelula));

    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 4: 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, TItem valor){

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

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

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

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

    inserirListaColuna(pMatriz->coluna, CeCabecaC);

    inserirListaLinha(pMatriz->linha, CeCabecaL);

}

**Figura 5: 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.*

Função Le

\*\*

Função Imprime

\*\*

* Main.c

1. **CONCLUSÃO**
2. **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.