Trabalho T1

Entregar os exercícios definidos ao final deste documento (cada exercício deve ser um arquivo em separado). Para a realização de tais exercícios, faça uma leitura atenciosa dos fundamentos teóricos necessários, apresentados a seguir. Procure materiais complementares em sites da internet e em livros da biblioteca. Implemente na linguagem C. Cada funcionalidade deve ser implementada como uma função diferente. Use alocação dinâmica sempre que possível (alocação de vetores, de listas, de matrizes, etc).

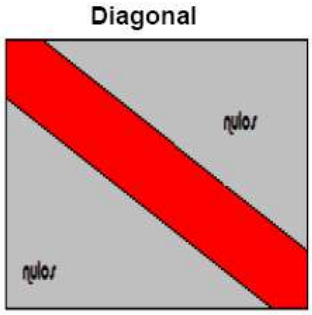
Trabalho em duplas
Entrega: por e-mail para deise@inf.ufsm.br
Data: 02 de outubro

O professor confirmará o recebimento do e-mail em até 24h. Se não receber a confirmação, entre em contato imediatamente. Trabalhos atrasados não serão considerados. Trabalhos copiados de outros grupos terão a nota zerada.

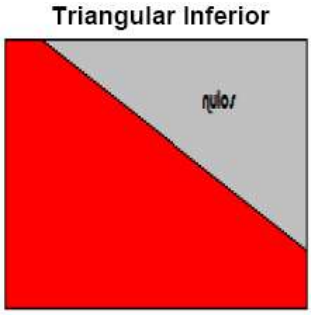
Fundamentos teóricos

Matrizes Especiais

São aquelas em que seus elementos se concentram de forma especial, de modo a se poder lançar mão de esquemas alternativos de armazenamento, visando a otimização em termos de espaço ou de velocidade de acesso. Exemplos:



Se i != j então M_{ij} = nulo



Se i < j então M_{ij} = nulo

Na medida em que se pode inferir um valor em função de uma propriedade geral da matriz, este valor não necessita ser armazenado. Por exemplo, na matriz diagonal, somente os valores da diagonal principal precisam ser armazenados, uma vez que os outros são nulos.

Algumas matrizes, por suas características, recebem denominações especiais:

- **Matriz diagonal**: matriz quadrada em que todos os elementos que não estão na diagonal principal são nulos. Por exemplo:

$$a)\ A_{2x2} = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \qquad b)\ B_{3x3} = \begin{bmatrix} 4 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 7 \end{bmatrix}$$

- **Matriz identidade**: matriz quadrada em que todos os elementos da diagonal principal são iguais a 1 e os demais são nulos; é representada por $\mathbf{I_n}$, sendo **n** a ordem da matriz. Por exemplo:

$$a)\ I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad b)\ I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Assim, para uma matriz identidade:

$$I_n = [a_{ij}], a_{ij} = \begin{cases} 1, sei = j \\ 0, sei \neq j \end{cases}$$

A única matriz identidade que não contém zero é a matriz identidade de ordem 1: $I_1 = [1]$

- **Matrizes Esparsas**

Uma matriz é dita esparsa quando possui uma grande quantidade de elementos que valem zero (ou não presentes, ou não necessários). Por exemplo:

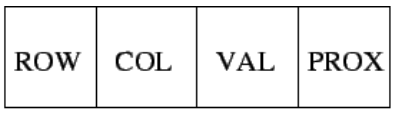
$$\begin{bmatrix} 0 & 0 & 7 & 0 & 0 & 11 & 0 \\ 8 & 0 & 0 & 0 & 0 & 0 & 3 \\ 0 & -1 & 0 & 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -3 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

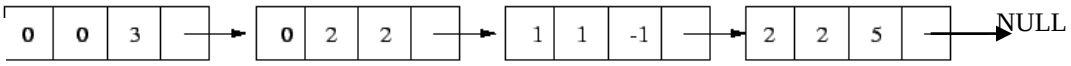
Podemos ver no exemplo acima que muito espaço em memória seria economizado se apenas os elementos não nulos fossem armazenados.
Matrizes esparsas têm aplicações em problemas de engenharia, física (por exemplo, o método das malhas para resolução de circuitos elétricos ou sistemas de equações lineares). Também têm aplicação em computação: armazenamento de dados (e.g., planilhas eletrônicas).

Representação:

Uma forma de representarmos uma matriz esparsa é utilizando uma lista simplesmente encadeada contendo todos os elementos não nulos da matriz. Os elementos podem ser armazenados por linha ou coluna. A figura abaixo apresenta uma estrutura de nó e a representação (por linhas) gerada a partir da matriz abaixo:

$$A_{3\times3} = \begin{bmatrix} 3 & 0 & 2 \\ 0 & -1 & 0 \\ 0 & 0 & 5 \end{bmatrix}$$





Onde: ROW: linha onde o elemento não nulo se encontra na matriz
COL: coluna onde o elemento não nulo se encontra na matriz
VAL: valor não nulo propriamente dito, armazenado na matriz
PROX: ponteiro para o próximo valor não nulo da matriz

Com a representação acima, para acessarmos o elemento que está na *i*-ésima linha temos que percorrer todos os elementos que estão nas linhas anteriores e para encontrarmos todos os elementos de uma coluna temos que percorrer todo o vetor. Outras formas de representação de matrizes esparsas podem ser encontradas no livro "C Completo e Total", de Herbert Schildt.

Observação para os exercícios 1, 2 e 3: observe que não há a alocação nem o preenchimento da matriz usando a abordagem de vetor simples ou de vetor de ponteiros. Existe apenas a alocação das structs apresentadas.

Exercício 1) Matriz diagonal

Faça um programa que trate uma matriz diagonal de inteiros. O programa deve armazenar os valores da diagonal principal (que são os não nulos) em um vetor *v*, e os demais valores (que são nulos) não devem ser lidos nem armazenados em lugar algum. O programa deve contemplar as operações de:
- criação da matriz;
- preenchimento da matriz;
- impressão da matriz;
- consulta de um determinado elemento da matriz (para consultar um elemento, o usuário informa a linha e a coluna onde o elemento se encontra);

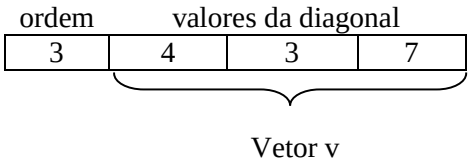
Use a seguinte definição para a *struct*:

```
struct diagonal
{
        int ordem; //ordem da matriz
        int* v; //o tamanho do vetor é igual a ordem da matriz. Os elementos da diagonal principal são armazenados neste vetor
};
typedef struct diagonal Diagonal;
```

Exemplo:
Para a matriz diagonal a seguir:

$$b)\ B_{3x3} = \begin{bmatrix} 4 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 7 \end{bmatrix}$$

A representação seria:



Exercício 2) Matriz esparsa

Faça um programa que trate uma matriz esparsa de inteiros. O programa deve armazenar os valores não nulos da matriz esparsa em uma lista simplesmente encadeada (ver *struct* Lista), e os demais valores (que são nulos) não devem ser lidos nem armazenados em lugar algum. O programa deve contemplar as operações de:
- criação da matriz;
- preenchimento da matriz (ler apenas os valores não nulos);
- impressão da matriz;
- consulta de um determinado elemento da matriz (para consultar um elemento, informe a linha e a coluna onde ele se encontra);
- impressão de somatório de uma linha i (informada pelo usuário);
- percentual de elementos não nulos na matriz lida.

Use as seguintes definições para as *structs*:

```
struct lista //lista de valores não nulos da matriz esparsa. Os elementos não nulos da matriz são armazenados nesta lista. Os valores restantes, q são nulos, não são armazenados em nenhum lugar.
{
        int linha; // linha onde se encontra o elemento não nulo
```

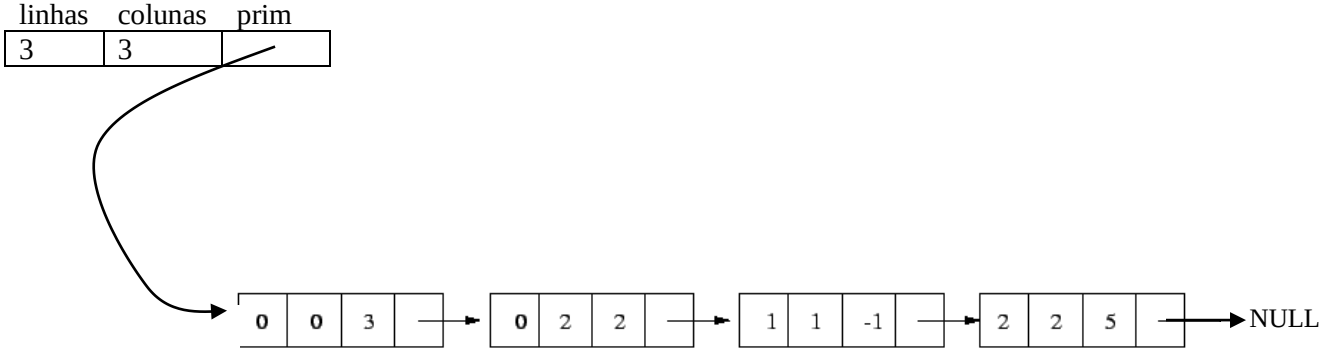
```
        int coluna; // coluna onde se encontra o elemento não nulo
        int info; // valor do  elemento não nulo
        struct lista* prox; // apontador para o proximo elemento não nulo da matriz
};
typedef struct lista Lista;

struct esparsa
{
        int linhas; //numero de linhas da matriz
        int colunas; //numero de colunas da matriz
        struct lista* prim; //apontador para o primeiro noh não nulo da matriz
};
typedef struct esparsa Esparsa;
```

Exemplo:
Para a matriz esparsa a seguir:

$$A_{3\times3} = \begin{bmatrix} 3 & 0 & 2 \\ 0 & -1 & 0 \\ 0 & 0 & 5 \end{bmatrix}$$

A representação seria:



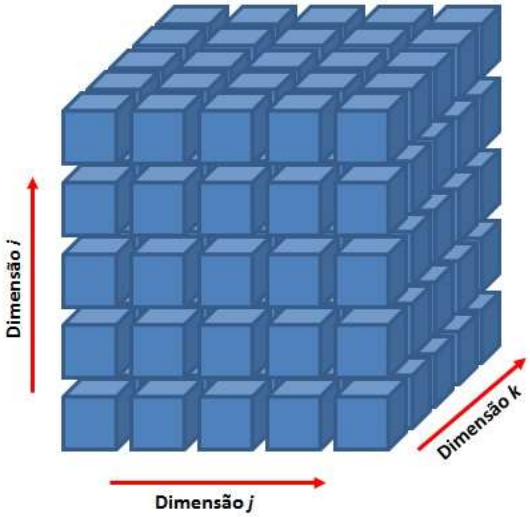
Exercício 3) Matriz identidade

Faça um programa que trate uma matriz quadrada de ordem N. O programa deve armazenar os valores da matriz e verificar se é uma matriz identidade. Se não for, mostrar quais elementos e suas respectivas posições (linha e coluna) que violaram a propriedade de matriz identidade.
O programa deve contemplar as operações de:
- criação da matriz;
- preenchimento da matriz;
- impressão da matriz;
- mostra de elementos (com sua respectiva linha e coluna) que violam a propriedade da identidade.

Proponha uma definição para a *struct* que representa a matriz identidade, usando listas, de uma forma otimizada em termos de armazenamento. Não aloque espaços desnecessários.

Exercício 4) Matriz de *n* dimensões

Faça um programa que trate uma matriz de inteiros de 3 dimensões (linhas, colunas e profundidade), conforme figura abaixo:



O programa deve contemplar as operações de:
- criação da matriz;
- preenchimento da matriz;
- impressão da matriz.

Os seguintes protótipos de funções devem ser usados:

int*** aloca_matriz (int m, int n, int z);
void preenche_matriz (int m, int n, int z, int ***mat);
void imprime_matriz (int m, int n, int z, int ***mat);