Matrizes Esparsas

Definição

- São matrizes de grande dimensão (centenas ou milhares de linhas e colunas) e que contêm muitos elementos com valor zero.

- Este tipo de matriz surge em diversas aplicações principalmente na física, na matemática e na economia.
  - ✓ **Também tem aplicação em computação: armazenamento de dados.**

- Se tentarmos representar uma matriz deste tamanho pelo modo convencional (*array*), o mais provável é que o compilador não permita a sua declaração, e mesmo que o faça, estaremos desperdiçando muita memória.

Definição

$$ME = \begin{bmatrix} 6 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 3 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

- Podemos otimizar o uso de memória armazenando somente os elementos não nulos desta matriz num vetor. No entanto, ao armazenarmos somente o valor perdemos a posição do elemento na matriz. Portanto, devemos armazenar, junto ao valor, seu respectivo índice.

vetme = [(1, 1, 6) (1, 3, 3) (2, 4, 2) (2, 6, 3) (3, 2, 1)]

Definição

vetme = [(1, 1, 6) (1, 3, 3) (2, 4, 2) (2, 6, 3) (3, 2, 1)]

- Note que cada elemento deste vetor é constituído por 3 valores:
  - ✓ a linha do elemento,
  - ✓ a coluna e
  - ✓ o valor armazenado na matriz.

- vetme trata-se de um vetor em que cada elemento é uma tupla e a construção desta tupla pode ser realizada utilizando o tipo *struct*.

Definição

- A consulta numa matriz esparsa se dá pela busca de um elemento que contenha o índice procurado. Caso este índice não exista significa que aquele elemento possui valor 0.

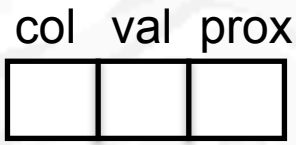
Implementação utilizando listas

- Matrizes esparsas podem ser implementadas utilizando listas encadeadas.
- Cada registro (estrutura) guarda uma entrada da matriz diferente de zero.

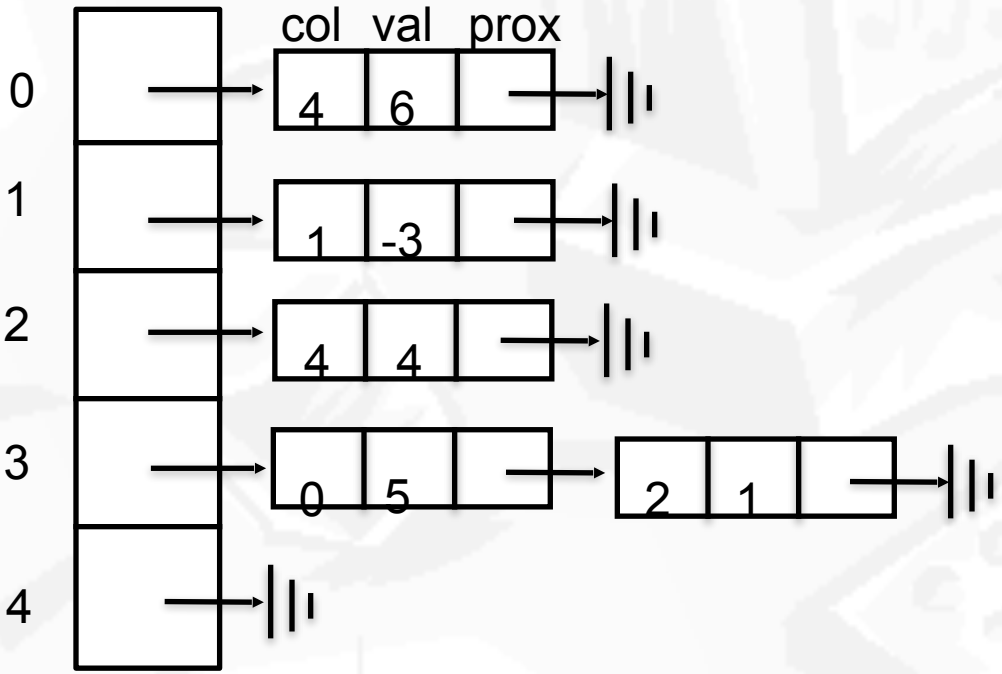
Implementação

- Uma forma de implementar matrizes esparsas utilizando listas encadeadas é guardar apenas uma lista de linhas e no registro não armazenar o número da linha, ou seja, utilizar um vetor de ponteiros.

```
#define MAX_LINHA 100
typedef struct no *pme;
struct no {
        int col, val;
        pme prox;
};
typedef pme matriz[MAX_LINHA];
```



Matriz= $\begin{pmatrix} 0 & 0 & 0 & 0 & 6 & 0 \\ 0 & -3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 \\ 5 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$



Implementação

```
int main( ) {
  matriz m;
  int lin, col, i, j, val;
  do{
       printf("Quantidade de linhas (menor que %d: ", MAX_LINHA+1);
       scanf("%d", &lin);
  }while (lin>0 && lin<= MAX_LINHA)
  printf("Quantidade de colunas: ");
  scanf("%d", &col);
  inicializa(m,lin);
  for(i=0; i<lin; i++)
     for(j=0; j<col; j++) {
          printf("m[%d][%d]= ", i, j);
          scanf("%d",&val);
         //inserir na matriz
```

Implementação

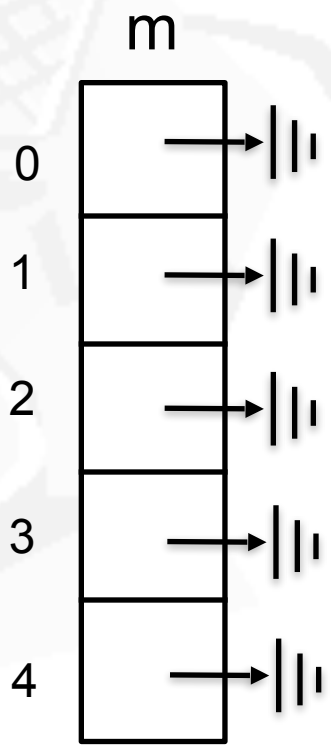
```
int main( ) {
  matriz m;
  int lin, col, i, j, val;
  do{
       printf("Quantidade de linhas (menor que %d: ", MAX_LINHA+1);
       scanf("%d", &lin);
  }while (lin>0 && lin<= MAX_LINHA)
  printf("Quantidade de colunas: ");
  scanf("%d", &col);
  inicializa(m,lin);
  for(i=0; i<lin; i++)
     for(j=0; j<col; j++) {
          printf("m[%d][%d]= ", i, j);
          scanf("%d",&val);
          if(val!=0)  insere(m,i,j,val);
     }
     imprime(m,lin,col);
     libera(m,lin);
     return(0);  }
```

Implementação

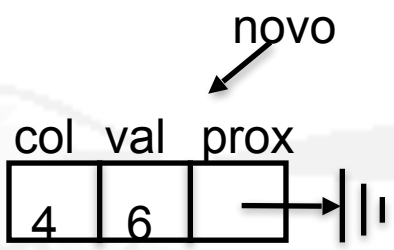
- Faça um programa para manipulação de matrizes esparsas. Inclua as funções Inicializa, Insere (considere inserção no final da lista), Imprime e Libera.

```c
typedef struct no *pme;
struct no {
            int col, val;
            pme prox;
            };
typedef pme matriz[MAX_LINHA];
```

```c
/* inicializa a matriz m */
void inicializa(matriz m, int l)
{
  int i;
  for (i=0; i<l; i++)
     m[i] = NULL;
}
```

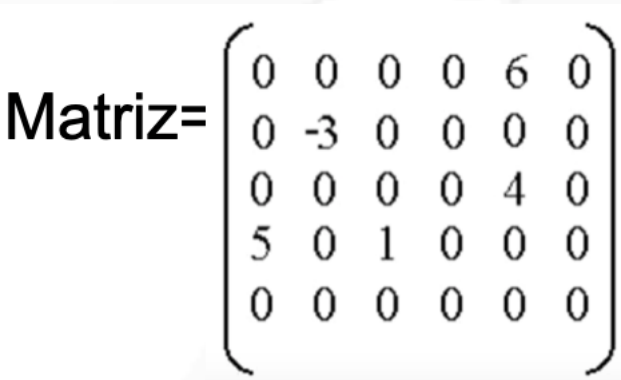


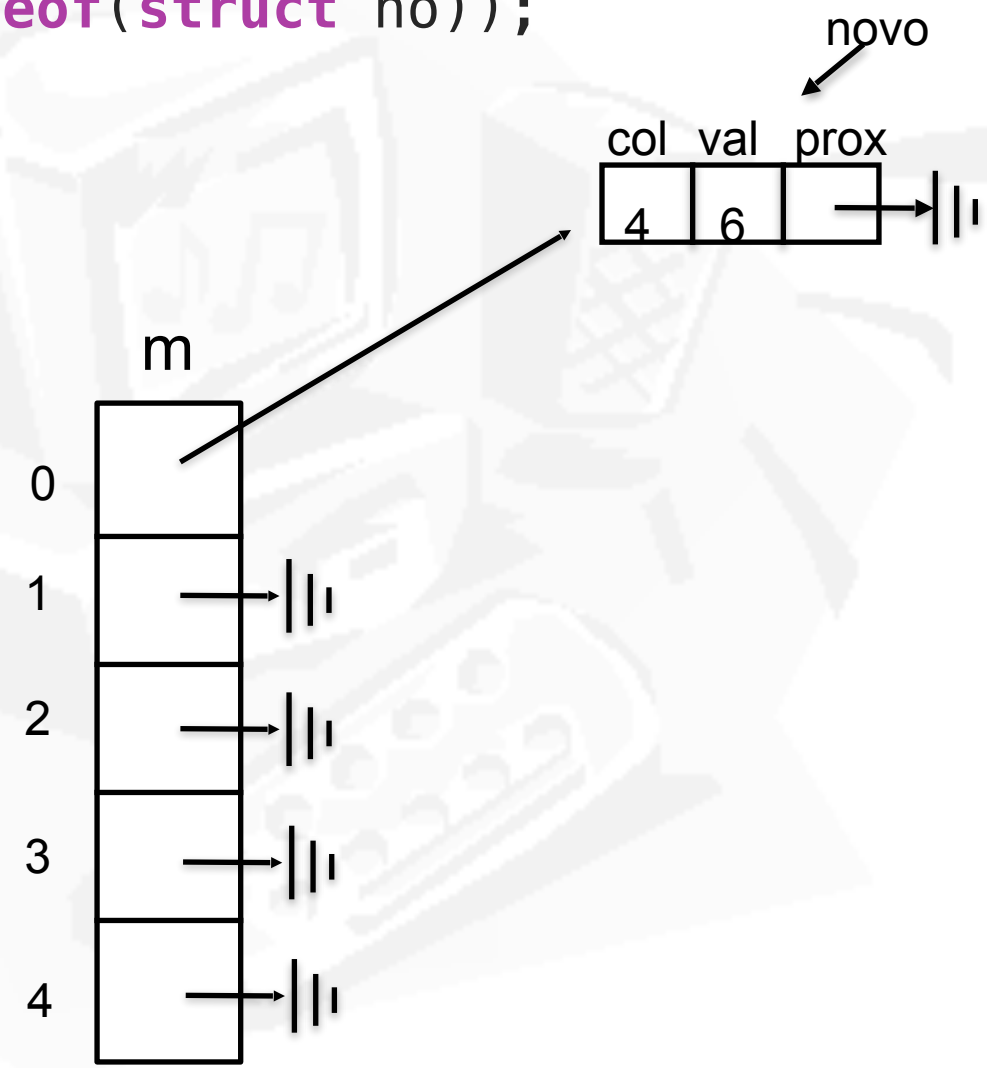
```
/* insere um elemento na matriz m */
void insere(matriz m, int i,int j, int val) {
    pme novo, p;
    novo = (pme)malloc(sizeof(struct no));
    novo->val= val;
    novo->col = j;
    novo->prox = NULL;
```



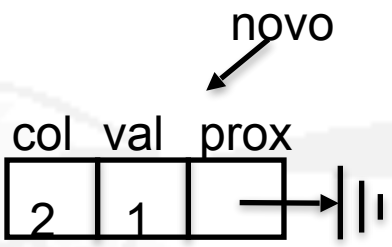
$$\text{Matriz} = \begin{pmatrix} 0 & 0 & 0 & 0 & 6 & 0 \\ 0 & -3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 \\ 5 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

```c
/* insere um elemento na matriz m */
void insere(matriz m, int i,int j, int val) {
    pme novo, p;
    novo = (pme)malloc(sizeof(struct no));
    novo->val= val;
    novo->col = j;
    novo->prox = NULL;
    if (m[i]==NULL)
       m[i] = novo;
```



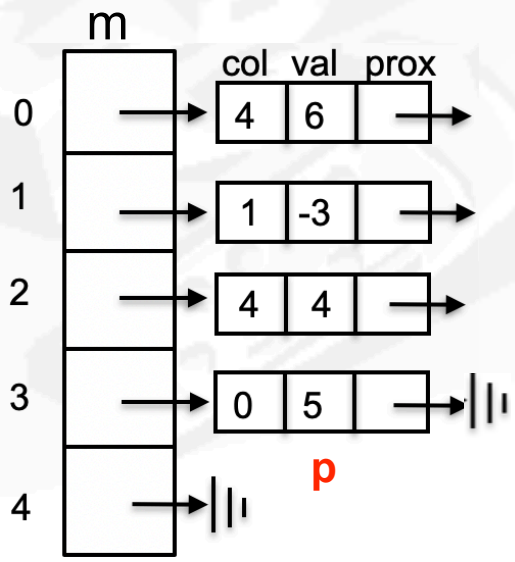


```
/* insere um elemento na matriz m */
void insere(matriz m, int i,int j, int val) {
    pme novo, p;
    novo = (pme)malloc(sizeof(struct no));
    novo->val= val;
    novo->col = j;
    novo->prox = NULL;
    if (m[i]==NULL)
       m[i] = novo;
    else   {
       for(p = m[i]; p->prox!=NULL; p=p->prox)
          ;
```

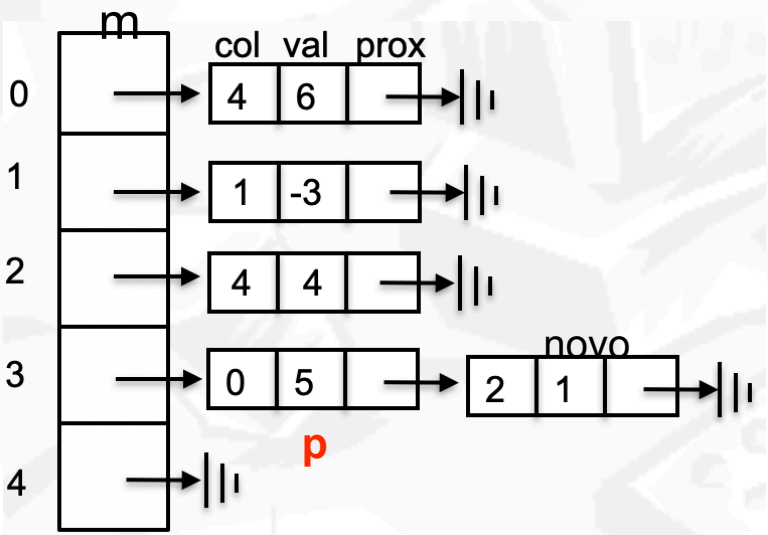


Matriz=

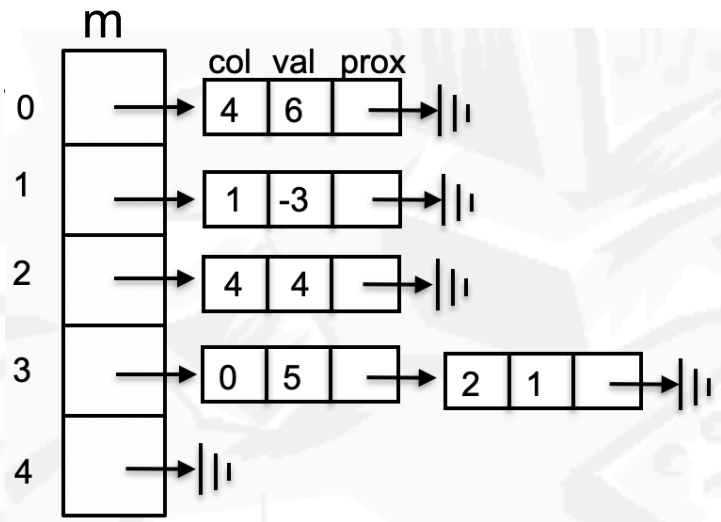
$$\begin{pmatrix} 0 & 0 & 0 & 0 & 6 & 0 \\ 0 & -3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 \\ 5 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$



```
/* insere um elemento na matriz m */
void insere(matriz m, int i,int j, int val) {
    pme novo, p;
    novo = (pme)malloc(sizeof(struct no));
    novo->val= val;
    novo->col = j;
    novo->prox = NULL;
    if (m[i]==NULL)
       m[i] = novo;
    else   {
       for(p = m[i]; p->prox!=NULL; p=p->prox)
          ;
       p->prox = novo;
    }
}
```



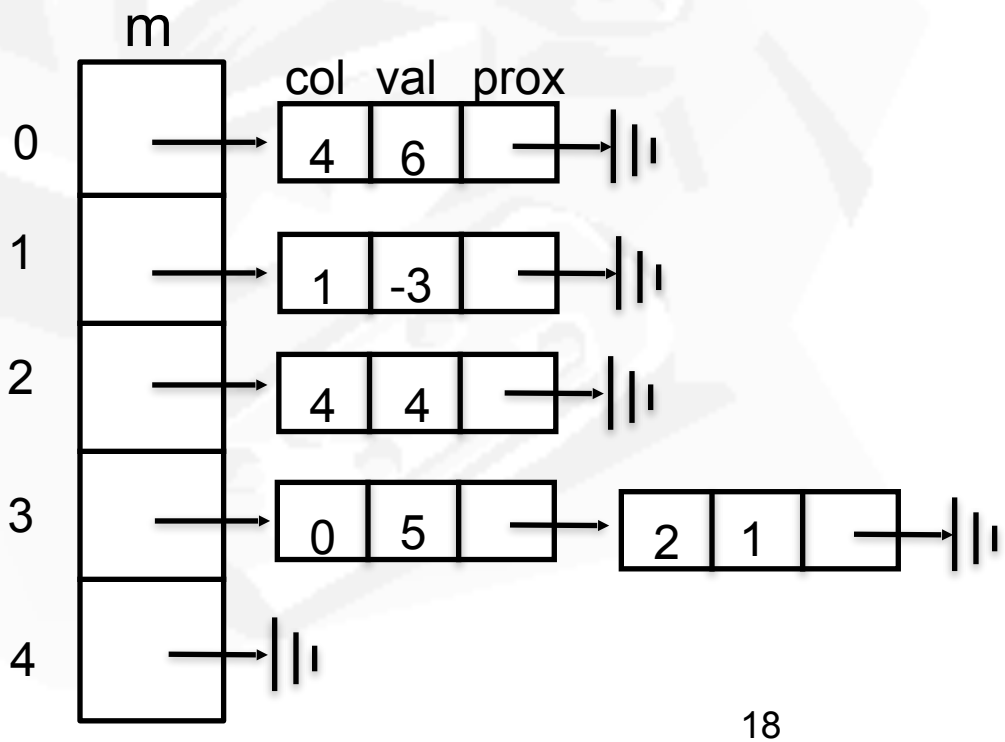
```c
/* imprime a matriz incluindo */
void imprime(matriz m, int tlin, int
{
   int i,j;
   pme p;
   for(i=0;i<tlin;i++)
   {
        p = m[i];
        for(j=0;j<tcol;j++)
            if ((p!=NULL) && (p->col==j))
            {
               printf(" %d ",p->val);
               p = p->prox;
            }
            else
               printf(" 0 ");
        printf("\n");
   }
   printf("\n\n");
}
```



Saída

0	0	0	0	6	0
0	-3	0	0	0	0
0	0	0	0	4	0
5	0	1	0	0	0
0	0	0	0	0	0

```c
/* libera os espaços de memória alocados para armazenar
as listas */
void libera(matriz m, int l)
{
    pme p, q;
    int i;
    for(i=0;i<l;i++)
    {
        p = m[i];
        while (p!=NULL)
        {
            q = p;
            p = p->prox;
            free(q);
        }
    }
}
```



Exercícios

- Escreva uma função que calcule a transposta de uma matriz esparsa.

```
/* Calcula a transposta de uma matriz esparsa */
void transposta(matriz m, int tlin, int tcol){
    int i;
    pme p;
    matriz result;
    if (tcol>MAX_LINHA) {
     printf ("Nao eh possivel criar a transposta! Estouro
de memoria!!!" );
        return();
   }
    inicializa(result,tcol);
    for (i=0;i<tlin;i++)
       for(p=m[i]; p!=NULL; p=p->prox)
           insere(result,p->col,i,p->val);
    printf("\nMatriz original:\n");
    imprime(m,tlin);
    printf("\nMatriz transposta:\n");
    imprime(result, tcol);
    libera(result, tcol);
}
```

Exercícios

- Faça uma função que verifique se uma matriz esparsa é simétrica.
- Faça uma função que verifique se uma matriz esparsa é diagonal (só tem elementos diferentes de zero na diagonal principal).
- Faça uma função que verifique se uma matriz esparsa é triangular inferior (se só tenho elementos diferentes de zero nas posições em que a linha>=coluna).

Exercícios

- Escreva funções que informem a linha e a coluna da matriz esparsa com maior valor médio.

- Tente implementar matrizes esparsas utilizando lista de lista.