

Matriz Esparsa e Lista Cruzada

Na matriz (5x6) abaixo, apenas 5 de seus 30 elementos são não nulos.

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 9 & 0 \\ 0 & -3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 \\ 5 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

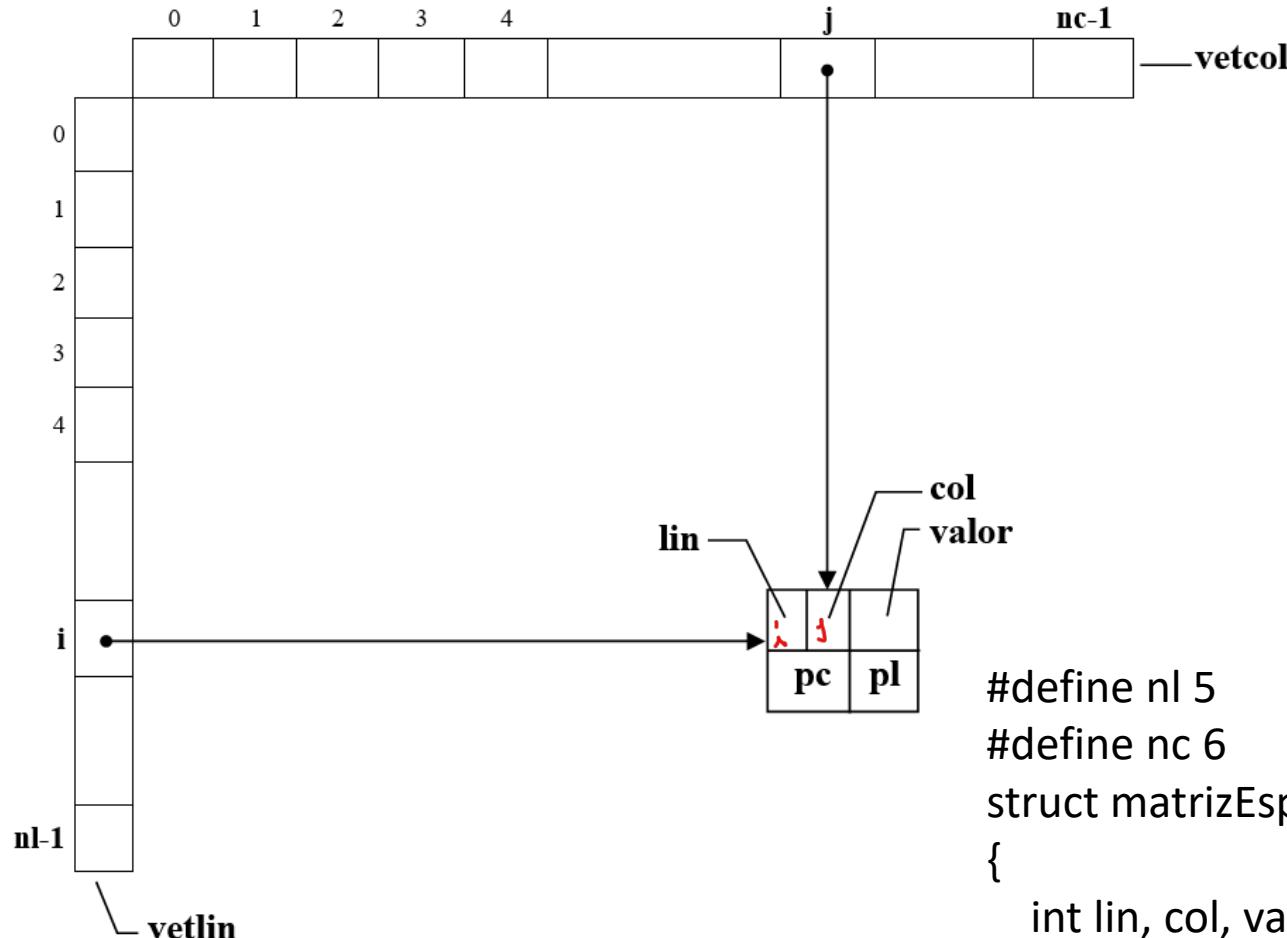
Estrutura de Dados de Listas Cruzadas propõe uma representação que evita o armazenamento de elementos nulos.

Para não representarmos os valores nulos, fazemos **listas de linhas** e **listas de colunas** tais que o elemento $m(i,j)$ diferente de 0 pertence:

- à lista dos elementos da linha i ;
- à lista dos elementos da coluna j .

Então se a matriz tem n_l linhas e n_c colunas, temos n_l listas de linhas e n_c listas de colunas.

Graficamente podemos ter algo como:

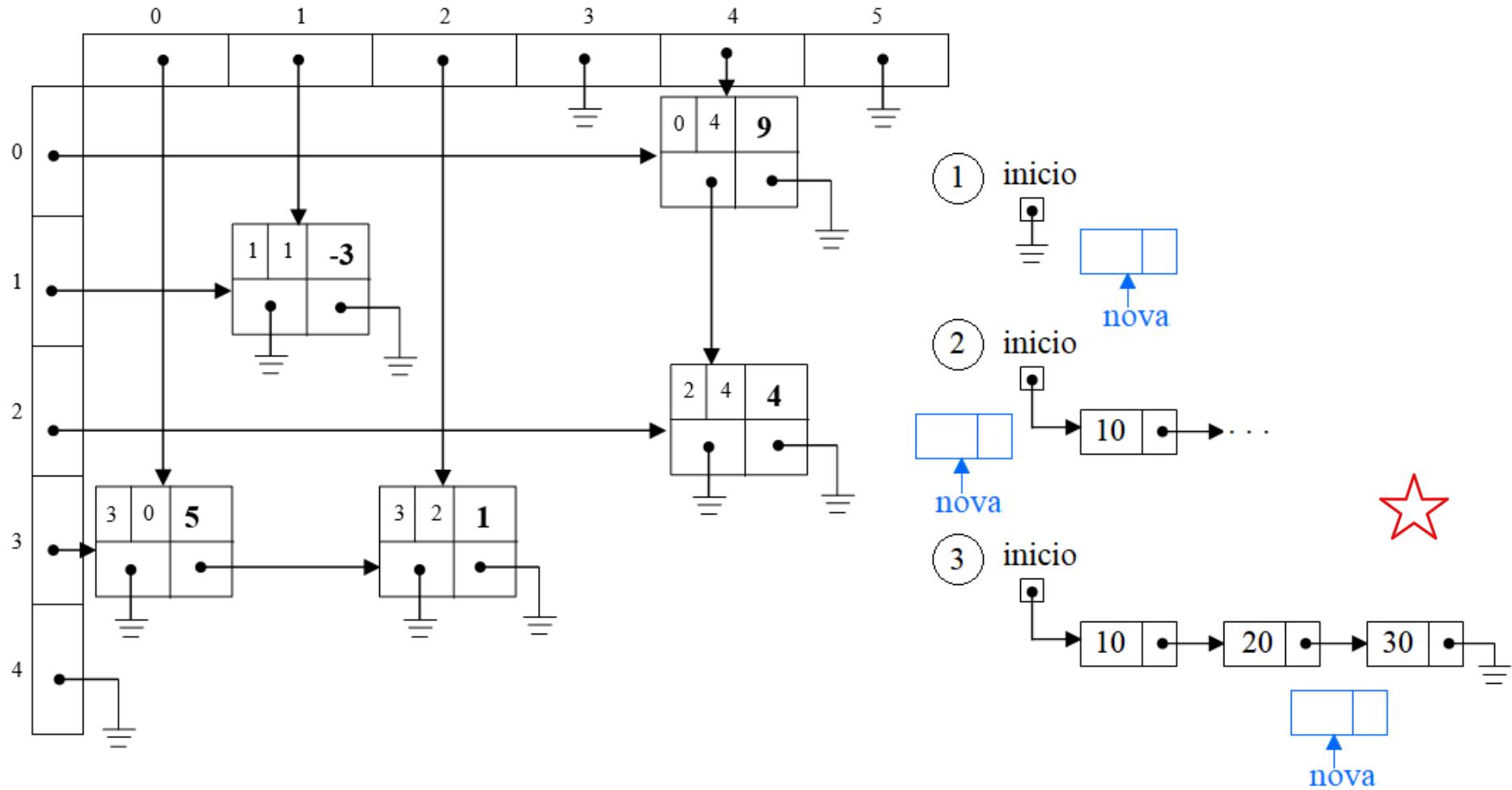


Definição da Estrutura de Dados:

```
#define nl 5
#define nc 6
struct matrizEsp
{
    int lin, col, valor;
    struct matrizEsp *pc, *pl;
};
typedef struct matrizEsp MatEsp;
```

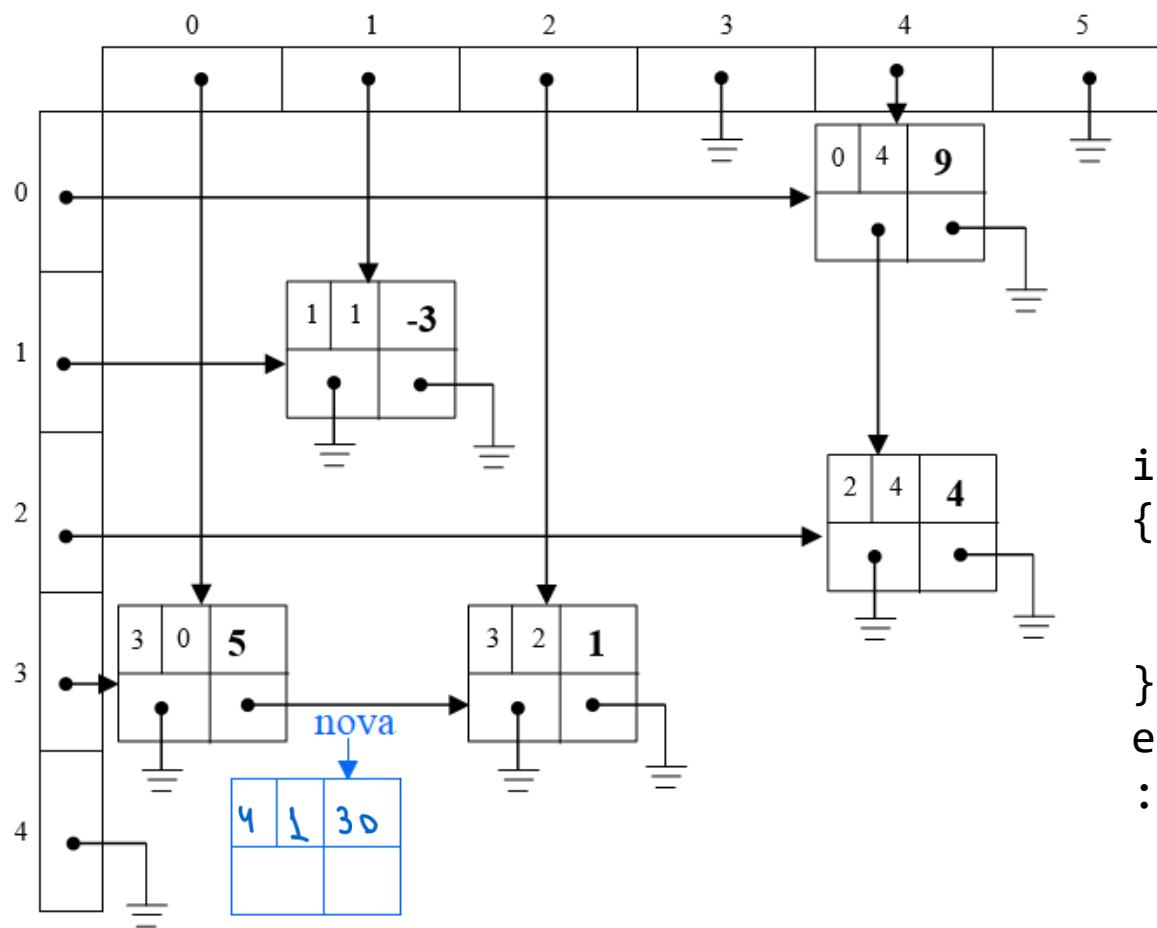
Algoritmo para inserir em um Matriz Esparsa:

3 casos a considerar: 1-aponta para NULL, 2-menor que todos, 3-no meio ou no final.



Algoritmo para inserir em um Matriz Esparsa (horizontal):

1º Caso: aponta para NULL.

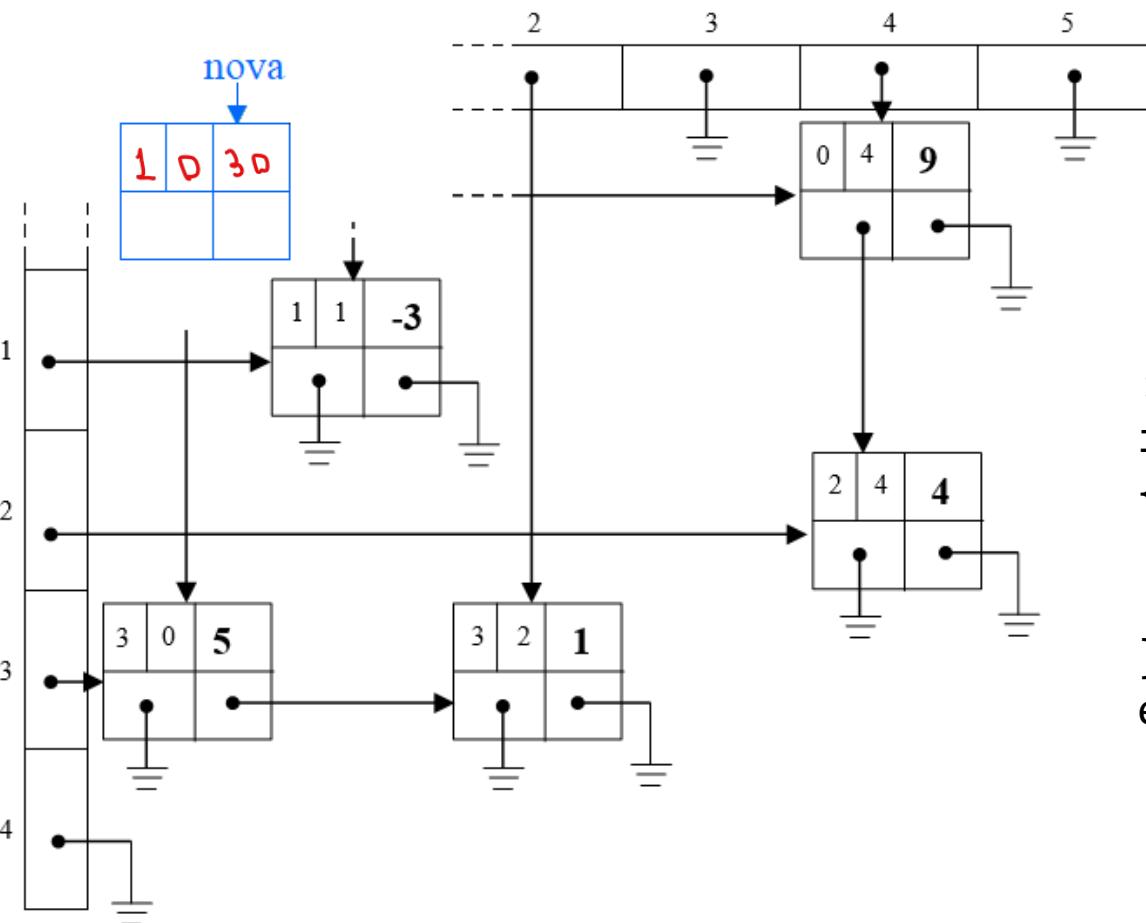


lin=4
col=1
valor=30

```
if (vetlin[lin]==NULL)
{
    vetlin[lin]=nova;
    nova->pl=NULL;
}
else
:
```

Algoritmo para inserir em um Matriz Esparsa (horizontal):

2º Caso: menor que todos.

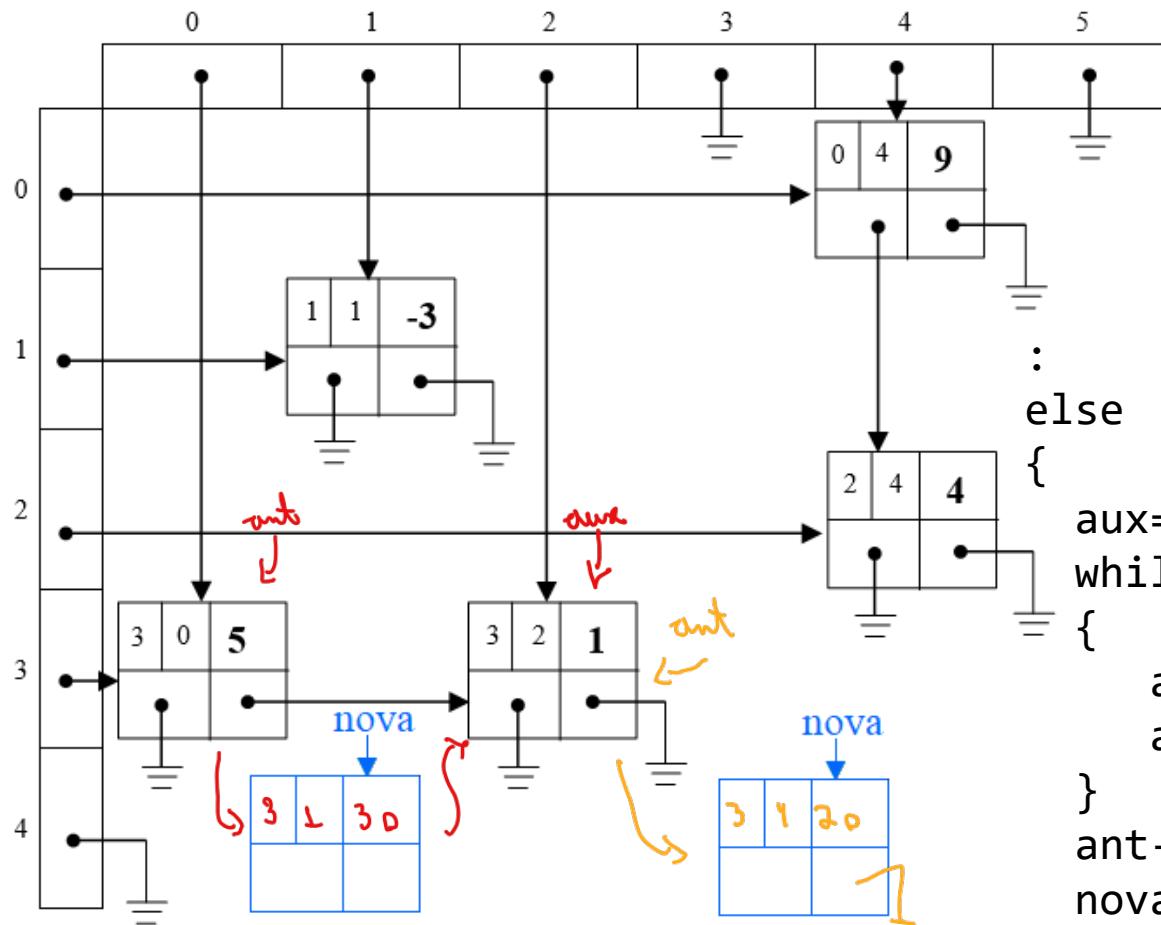


lin=1
col=0
valor=30

```
:  
if (col<vetlin[lin]->col)  
{  
    nova->pl=vetlin[lin];  
    vetlin[lin]=nova;  
}  
else  
:  
:
```

Algoritmo para inserir em um Matriz Esparsa (horizontal):

3º Caso: no meio ou no final.



lin=3
col=1
valor=30

```
lin=3
col=1
valor=30

else
{
    aux=vetlin[lin];
    while(aux!=NULL && col>aux->col)
    {
        ant=aux;
        aux=aux->pl;
    }
    ant->pl=nova;
    nova->pl=aux;
}
```

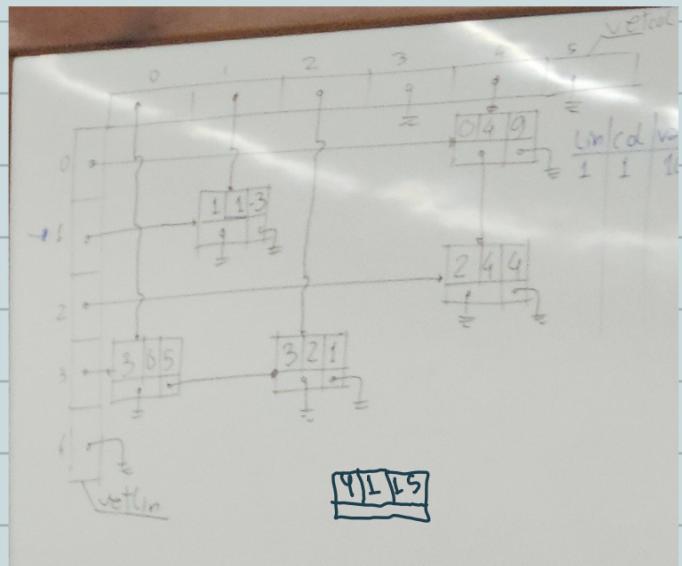
Algoritmo para inserir em um Matriz Esparsa:

```
void insere(MatEsp *vetlin[], MatEsp *vetcol[], int lin, int col,
            int valor)
{
    MatEsp *ant,*aux,*nova;
    if (lin>=0 && lin<nl && col>=0 && col<nc)
    {
        verificaOcupado(vetlin,lin,col,&aux);
        if (aux!=NULL)
            aux->valor=valor;
        else
        {
            nova=(MatEsp*)malloc(sizeof(MatEsp));
            nova->lin=lin;
            nova->col=col;
            nova->valor=valor;
            //Ligaçao Horizontal
            //Ligaçao Vertical
        }
    }
    else
        printf("As coordenadas estão fora da Matriz!");
}
```

Algoritmos a serem implementados no TAD Matriz Esparsa:

- a) inicializar uma matriz esparsa;
- b) inserir um determinado elemento na posição i, j ;
- c) excluir um elemento da posição i, j ;
- d) exibir uma matriz esparsa;
- e) somar duas matrizes esparsas e gerar uma terceira;
- f) fazer a multiplicação de duas matrizes esparsas;
- g) excluir uma matriz esparsa.

```
void insertMat( MatExp *matrix[], MatExp *matrix[],
    int dim, int col, int val);
```



MatExp *new;

```
if (dim >= 0 && dim < dim && col >= 0 && col <= nc) {
```

```
    new = newMatCreate( matrix[0], matrix[1][dim], col);
```

```
    if (new != NULL) {
```

```
        new->value = value;
```

```
}
```

```
else {
```

```
    new = (MatExp *) malloc(sizeof(MatExp));
```

```
    new->dim = dim;
```

```
    new->col = col;
```

```
    new->value = value;
```

```
// ligar no vertical
```

```
if (matrix[0][dim] == NULL) {
```

```
    matrix[0][dim] = new;
```

```
    new->pl = NULL;
```

```
}
```

```
else {
```

```
    if (col < matrix[0][dim] -> col) {
```

```
        new->pl = matrix[0][dim];
```

```
        matrix[0][dim] = new;
```

```

}
else {
    ant = retlin[lin];
    aux = auxlin[lin] -> pl;
    while (aux != NULL && col > aux->col)
        ant = aux;
        aux = aux->prox;
    }
    ant->pl = norm;
    norm->pl = aux;
}
}

```

```

// ligarrie vertical
if (retcol[col] == NULL) {
    retcol[col] = norm;
    retcol[col]->pr = NULL;
}

else {
    if (col < retcol[col]) {
        norm->pr = retcol[col];
        retcol[col] = norm;
    }
    else {
        ant = retcol[col];
        aux = ant->prox;
        while (aux != NULL && lin < aux->lin)
            ant = aux;
            aux = ant->prox;
        }
        ant->pr = norm;
        norm->pr = aux;
}

```

: }
}
}
}

```
void exile(Matrix *matrix[ ]) {  
    int i, j;  
    Matrix *arr;  
  
    for (i = 0; i < m; i++) {  
        for (j = 0; j < n; j++) {  
            arr = matrix[i][j];  
  
            if (arr != NULL) {  
                printf("%d\t", arr->val);  
            } else {  
                printf("0\t");  
            }  
        }  
        printf("\n");  
    }  
}
```

```
Matrix verifyInput(Matrix *matrix, int val) {  
    while (matrix[val] == NULL && val > matrix->val)  
        matrix = matrix->pl  
  
    if (matrix->val != val && val == matrix->val)  
        return matrix;  
    return NULL;  
}
```