Listas e matrizes esparsas

SCC122 – Estruturas de Dados

Matriz: definição

- Matriz é um arranjo (tabela) retangular de números dispostos em linhas e colunas

$$A_{3x4} = \begin{bmatrix} 1 & 0 & 4 & -3 \\ 2 & 5 & 3 & 4 \\ 9 & 8 & -2 & 1 \end{bmatrix} \qquad B_{3x3} = \begin{bmatrix} 3 & 7 & 4 \\ 1 & 0 & 6 \\ 9 & 2 & 8 \end{bmatrix}$$

nº de elementos = nº de linhas * nº de colunas

Matriz = Array Bidimensional

Matrizes especiais

$$A_{3x3} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 3 & 0 \\ 4 & 5 & 6 \end{bmatrix}$$

Triangular inferior

$$B_{4x4} = \begin{bmatrix} 1 & 2 & 0 & 0 \\ 2 & 4 & 5 & 0 \\ 0 & 6 & 7 & 8 \\ 0 & 0 & 9 & 10 \end{bmatrix}$$

Tri-diagonal

$$C_{7x9} = \begin{bmatrix} 1 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Matriz esparsa: excessivo nº de elementos nulos (0)

Matrizes Especiais

- Quadradas
  - Diagonal
  - M(i,j) = 0 para i diferente j
  - Tridiagonal
  - M(i,j) = 0 para | i-j | > 1
  - Triangular Inferior
    - M(i,j) = 0 para i < j
  - Triangular Superior
    - M(i,j) = 0 para i > j
  - Simetrica
    - M(i,j) = M(j,i) para todo i, j

Matriz esparsa: exemplo

$$C_{700x900} = \begin{bmatrix} 1 & 0 & 0 & 3 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & -1 & 4 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

700 x 900 = 630.000 elementos

Matriz esparsa com **9** elementos **não nulos**

Matrizes esparsas

- Uso da matriz tradicional
  - Vantagem
    - Ao se representar dessa forma, preserva-se o acesso direto a cada elemento da matriz
      - Algoritmos simples
  - Desvantagem
    - Muito espaço para armazenar zeros

Matrizes esparsas

- Necessidade
  - Método alternativo para representação de matrizes esparsas
- Solução
  - Estrutura de lista encadeada contendo somente os elementos não nulos

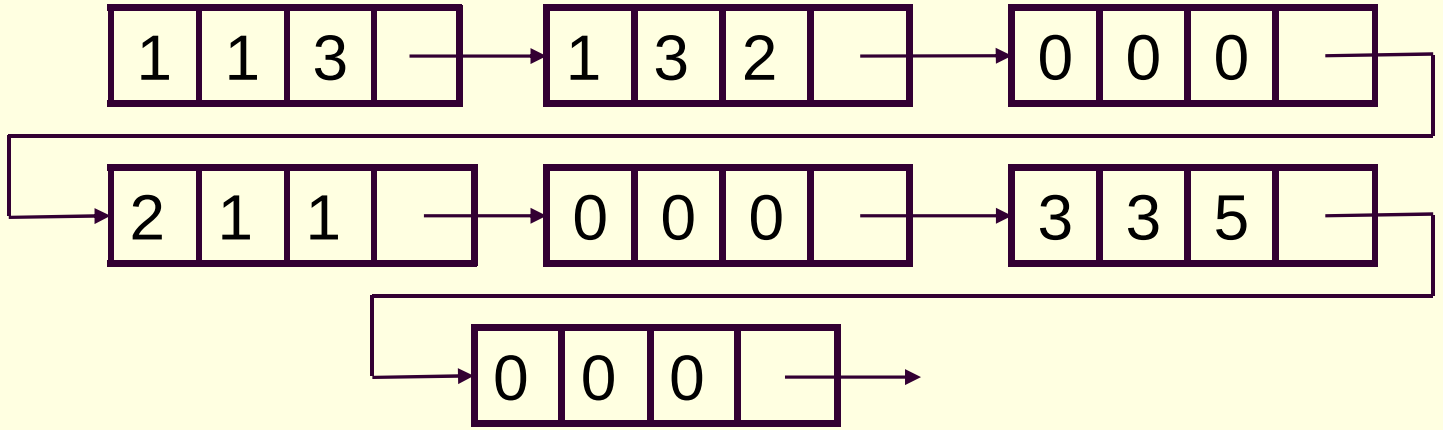
Solução 1

- Listas simples encadeadas

$$A_{3x3} = \begin{bmatrix} 3 & 0 & 2 \\ 1 & 0 & 0 \\ 0 & 0 & 5 \end{bmatrix}$$

linha	coluna	valor	next

Estrutura de um Nó:
- linha, coluna: posição
- valor: ≠ zero
- next: próx nó



Solução 1

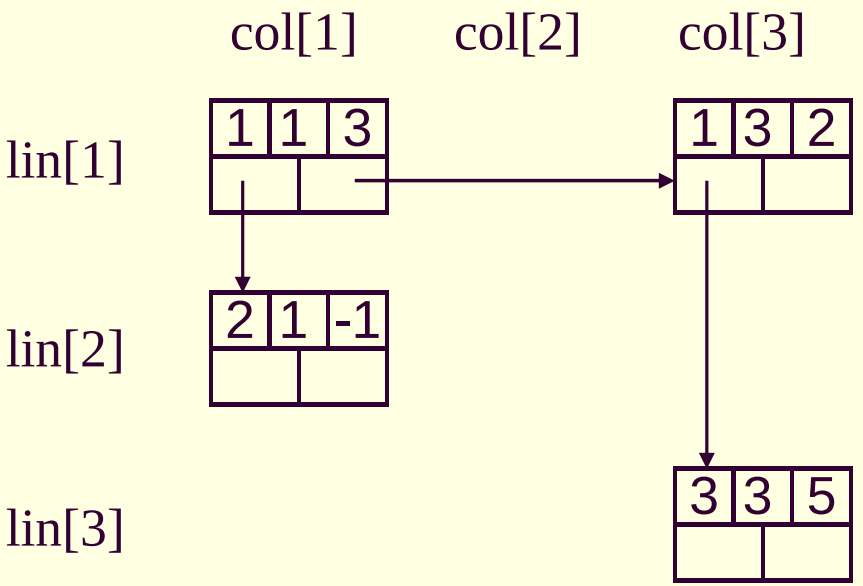
- Desvantagens
  - Perda da natureza bidimensional de matriz
  - Acesso ineficiente à linha
    - Para acessar o elemento na i-ésima linha, deve-se atravessar as i-1 linhas anteriores
  - Acesso Ineficiente à coluna
    - Para acessar os elementos na j-ésima coluna, deve-se atravessar toda lista

- Questão
  - Como organizar esta lista, preservando a natureza bidimensional de matriz?

Solução 2

- Listas cruzadas
  - Para cada matriz, usam-se dois vetores com N ponteiros para as linhas e M ponteiros para as colunas

$$A_{3x3} = \begin{bmatrix} 3 & 0 & 2 \\ -1 & 0 & 0 \\ 0 & 0 & 5 \end{bmatrix}$$



Estrutura de um Nó:

linha	coluna	valor
proxlin		proxcol

Solução 2

- Listas cruzadas
  - Cada elemento não nulo é mantido simultaneamente em duas listas
    - Uma para sua linha
    - Uma para sua coluna

Estrutura de Dados

```
typedef reg *preg;
struct reg{
        int linha;      /* 1..nl*/
        int coluna;     /* 1..nc*/
        tipo_elem valor;
        preg PL,PC;  /*ponteiro p/ próximo registro */
};
```

preg vetorCol[nl];

preg vetorLin[nc];

Matrizes esparsas

- Listas cruzadas vs. matriz tradicional
  - Em termos de espaço
    - Supor que inteiro e ponteiro para inteiro ocupam um bloco de memória
    - Listas cruzadas: tamanho do vetor de linhas (nl) + tamanho do vetor de colunas (nc) + n elementos não nulos * tamanho do nó
      - nl+nc+5n
    - Matriz tradicional bidimensional
      - nl*nc

Matrizes esparsas

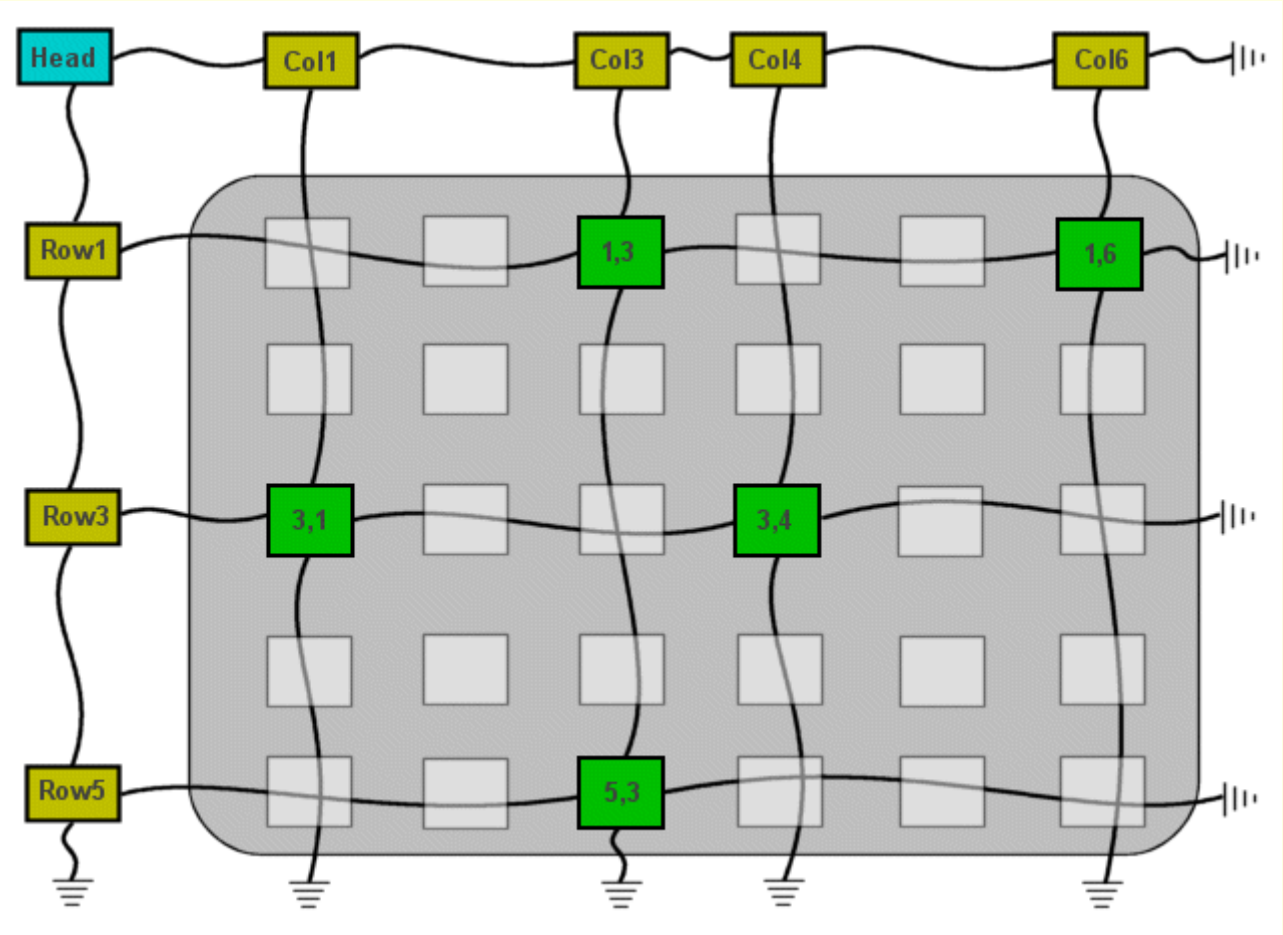
- Listas cruzadas vs. matriz tradicional
  - Em termos de tempo
    - Operações mais lentas em listas cruzadas: acesso não é direto

Matrizes esparsas

- Listas cruzadas vs. matriz tradicional

- Em termos de espaço ocupado, é vantajoso utilizar a representação de listas cruzadas quando:

  - 5n + nl + nc < nl * nc
  - ou seja, quando: n < [(nl - 1) * (nc - 1) -1] / 5
  - Como (nl - 1) * (nc - 1) é aproximadamente o tamanho da matriz, pode-se dizer, de uma maneira geral, que há ganho de espaço, quando **um número inferior a 1/5 dos elementos da matriz forem não nulos**

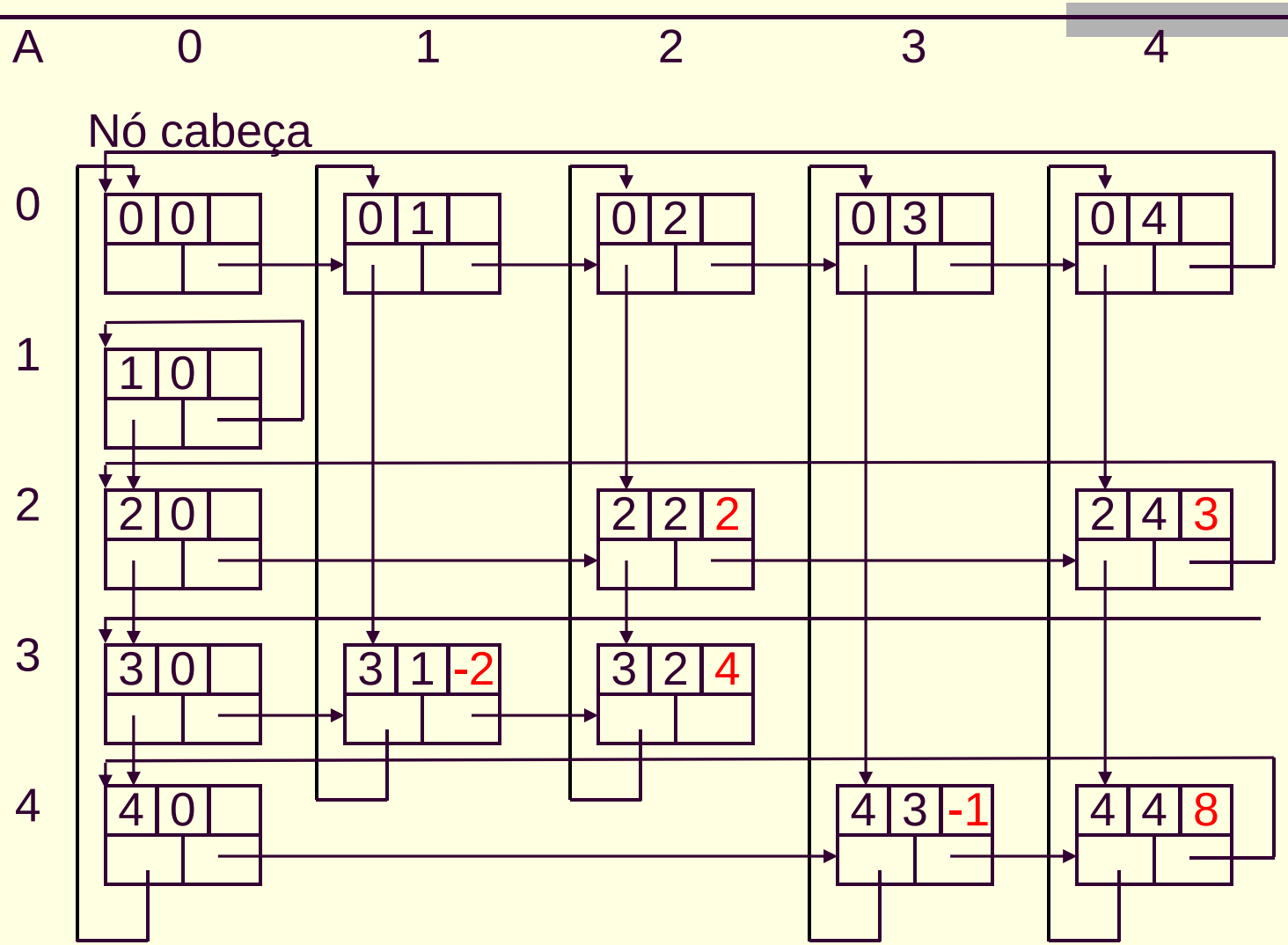

Head
Col1
Col3
Col4
Col6
Row1
1,3
1,6
Row3
3,1
3,4
Row5
5,3

Outra solução

- Listas circulares com nós cabeças
  - Ao invés de vetores de ponteiros, linhas e colunas são listas circulares com nós cabeças
    - Nós cabeças: reconhecidos por um 0 (ou -1) no campo linha ou coluna
      - 1 único ponteiro para a matriz: navegação em qualquer sentido
  - Exemplo

$$A_{4x4} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 3 \\ -2 & 4 & 0 & 0 \\ 0 & 0 & -1 & 8 \end{bmatrix}$$

$$A_{4x4} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 3 \\ -2 & 4 & 0 & 0 \\ 0 & 0 & -1 & 8 \end{bmatrix}$$



- Listas circulares com nós cabeças
  - Quais as desvantagens dessa representação?
  - Melhor ou pior do que listas cruzadas?
    - Em termos de espaço?
    - Em termos de tempo?
  - Quando usar essas listas?

Estrutura de Dados (C)

```
typedef struct RegMatriz{
    int linha, coluna;
    float valor;
    struct MatrAux *direita, *abaixo;
};

RegMatriz* Matriz;
```

- void InicializaMatriz(Matriz a, int m,n);

Este procedimento cria uma nova matriz a, com m linhas e n colunas, com todos os elementos iguais a zero.

- void LiberaMatriz(Matriz a);

Libera toda a memória dinâmica utilizada pela matriz a.

- void SomaMatrizes(Matriz a,b,c);

Soma as matrizes a e b, deixando o resultado em c. Sup˜oe que as matrizes são compatíveis quanto ao número de linhas e de colunas.

- void MultiplicaMatrizes(Matriz a,b,c);

Multiplica as matrizes a e b, deixando o resultado em c. Supõe que as matrizes são compatíveis para esta operação.

Operações sobre matrizes esparsas

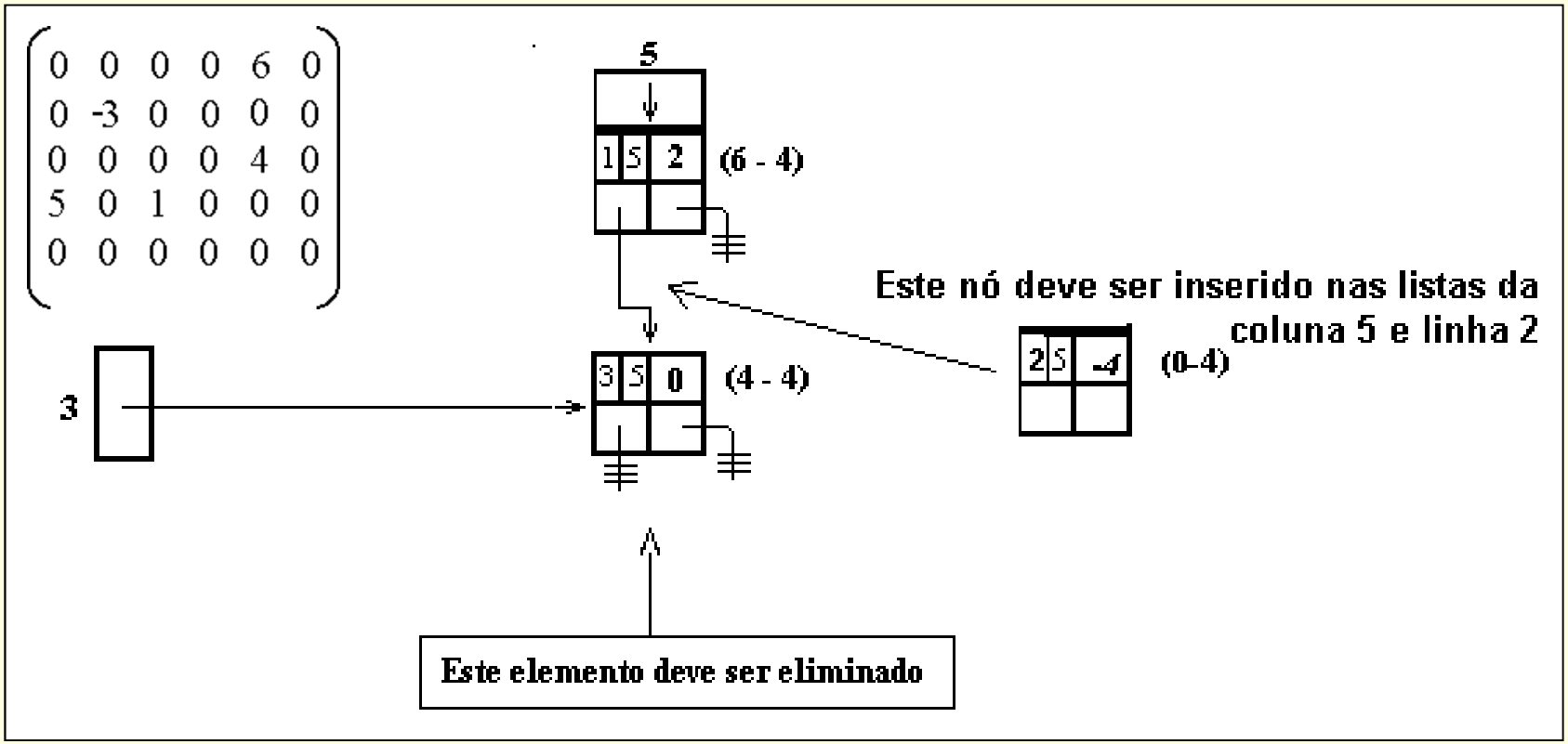
- Em geral
  - Multiplicar uma dada linha ou coluna por uma constante
  - Somar uma constante a todos os elementos de uma linha ou coluna
  - Somar duas matrizes esparsas de igual dimensão
  - Multiplicar matrizes esparsas
  - Transpor matrizes esparsas
  - Inserir, remover ou alterar elementos
  - Etc.

Operações sobre matrizes esparsas

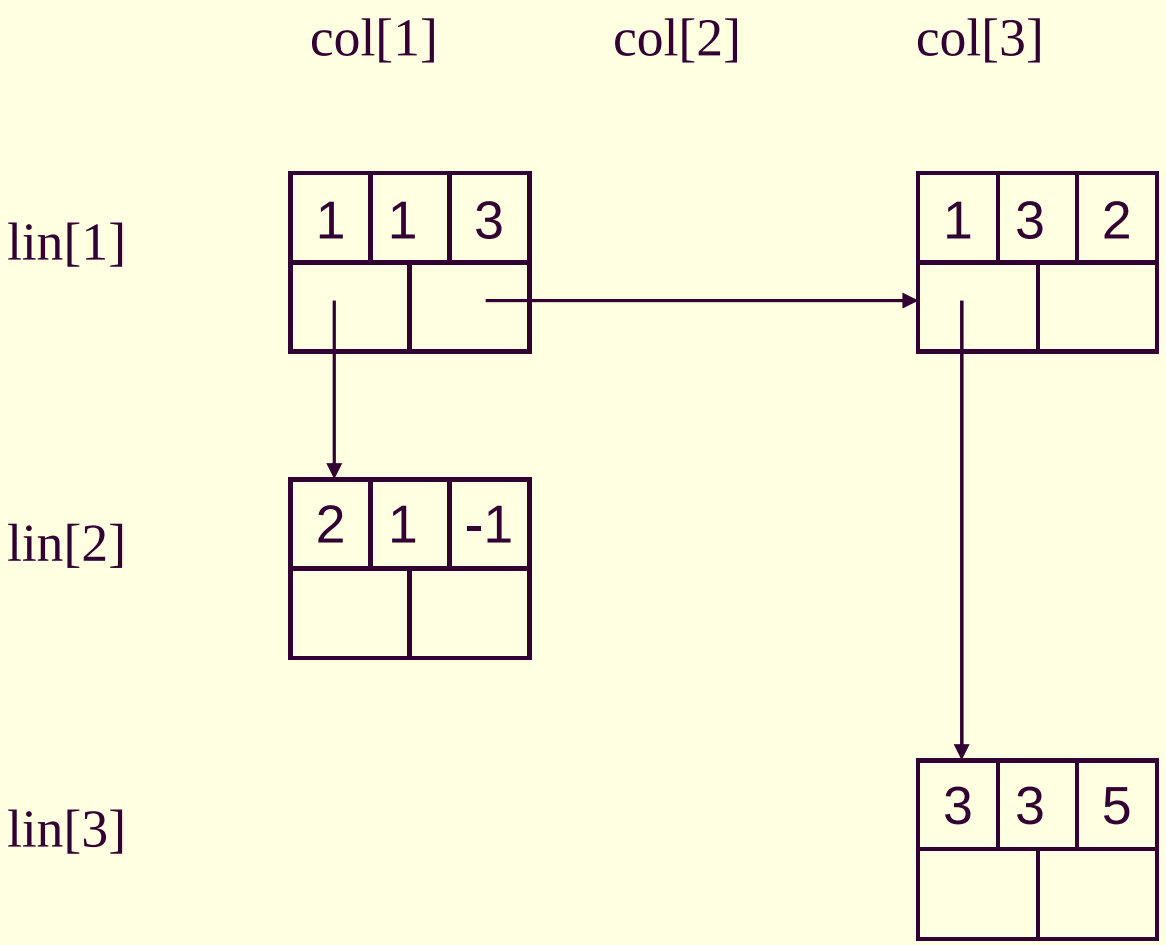
- Após a realização de alguma operação sobre a matriz
  - Quando um elemento da matriz se torna nulo
    - Remoção do elemento
  - Quando algum elemento se torna não nulo
    - Inserção do elemento

Operações sobre matrizes esparsas

- Por exemplo, ao se somar -4 a coluna 5 do exemplo



Exercício: somar -5 a coluna 3



Exercício

- Implementar uma sub-rotina para somar um número K qualquer a uma coluna da matriz
  - Usando listas cruzadas

Void soma(preg vetorLin, preg vetorCol, int nl, int nc, int j, int K){

```
typedef reg *preg;
struct reg{
        int linha;      /* 1..nl*/
        int coluna;     /* 1..nc*/
        tipo_elem valor;
        preg PL,PC;  /*ponteiro p/ próximo registro */
};
```

preg vetorCol[nl];

preg vetorLin[nc];

```
void soma(Rec *lin[], Rec *col[], int nl, int nc, int j, int k) {
    Rec *p;
    int i;
    p = col[j];
     if (p == NULL) { /* se a coluna possui apenas valores nulos */
        for (i=1; i<nl; i++)
                inserir(i, j, k, lin, col);
        return;
}

    for (i=1; i<nl; i++) {
        if (i != p->linha) /* se o valor é nulo */
            inserir(i, j, k, lin, col);
        else {
            p->valor = p->valor + k;
            if (p-> valor == 0) { /* se o valor torna-se nulo */
                p = p->proxlin;
                eliminar(i, j, lin, col);
        } else
                p = p->proxlin;
       }
   }

}
```

- Estes slides foram preparados pelos profes; Thiago Pardo e Maria da Graça Nunes e modificados pela profa. Roseli Aparecida Francelin Romero e por Mario Gazziro em 2011.