Listas e matrizes esparsas

Orthogonal List Representation
(Listas Cruzadas)

21/10/2010

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

Matrizes Estruturadas podem ser mapeadas para 1 vetor

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

Triangular inferior

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

Diagonal

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

Tri-diagonal

E esta??

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

Matriz esparsa: excessivo nº de elementos nulos (0)

Veja que não é uma definição precisa

Matrizes Densas: poucos elementos são zero

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**

Ex: Matriz de voos de linhas aéreas

- Aeroportos são numerados de 1 até n
- voo(i,j) = Lista de voos diretos do aeroporto i para aeroporto j
- Voo(i,j) = NULL se não há nenhum voo de i->j
- n = 1000 (supor)
- n x n vetor de listas => 4 milhões bytes
- Número total de voos = 20,000 (supor)
- Precisamos de até 20.000 listas => no máximo 80.000 bytes

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 ???
  - Não há uma única.

Solução 1: representação por linhas

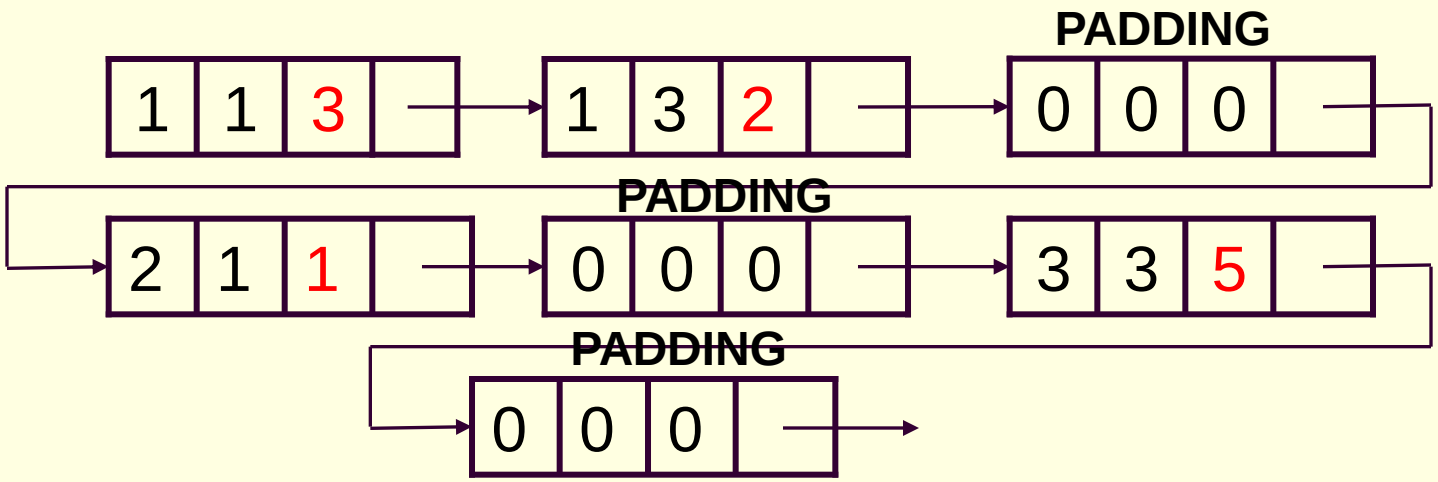
- Listas simples encadeadas: elementos não nulos

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

linha	coluna	valor	next

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

9 inteiros (18) x 4 nós de 3 inteiros cada (12) + 4 ponteiros (16), fora 3 paddings



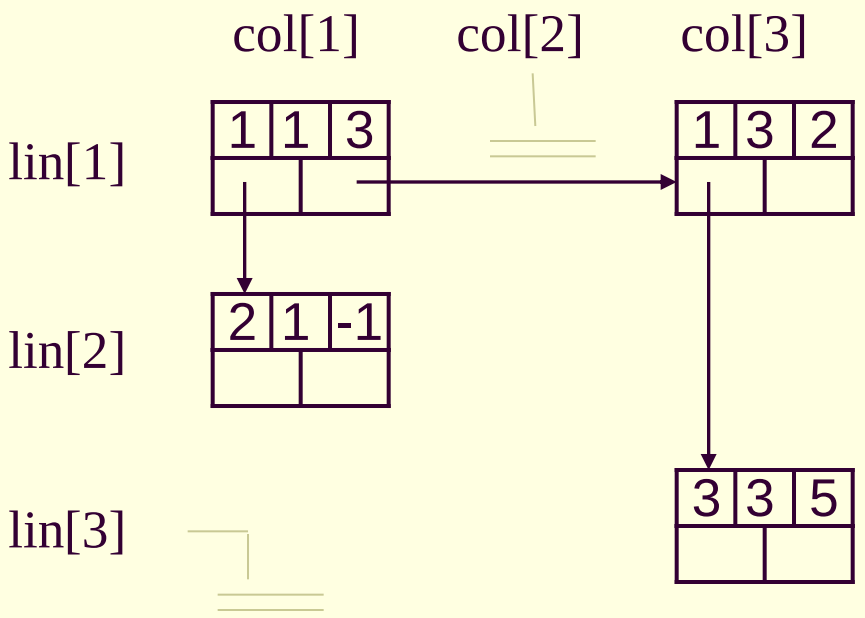
Solução 1

- 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?
- Deve haver também um balanço entre o espaço gasto pela matriz tradicional e a ED em lista (deve valer a pena) !

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

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

  - 5n + nl + nc < nl * nc
  - n < 1/5 (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**

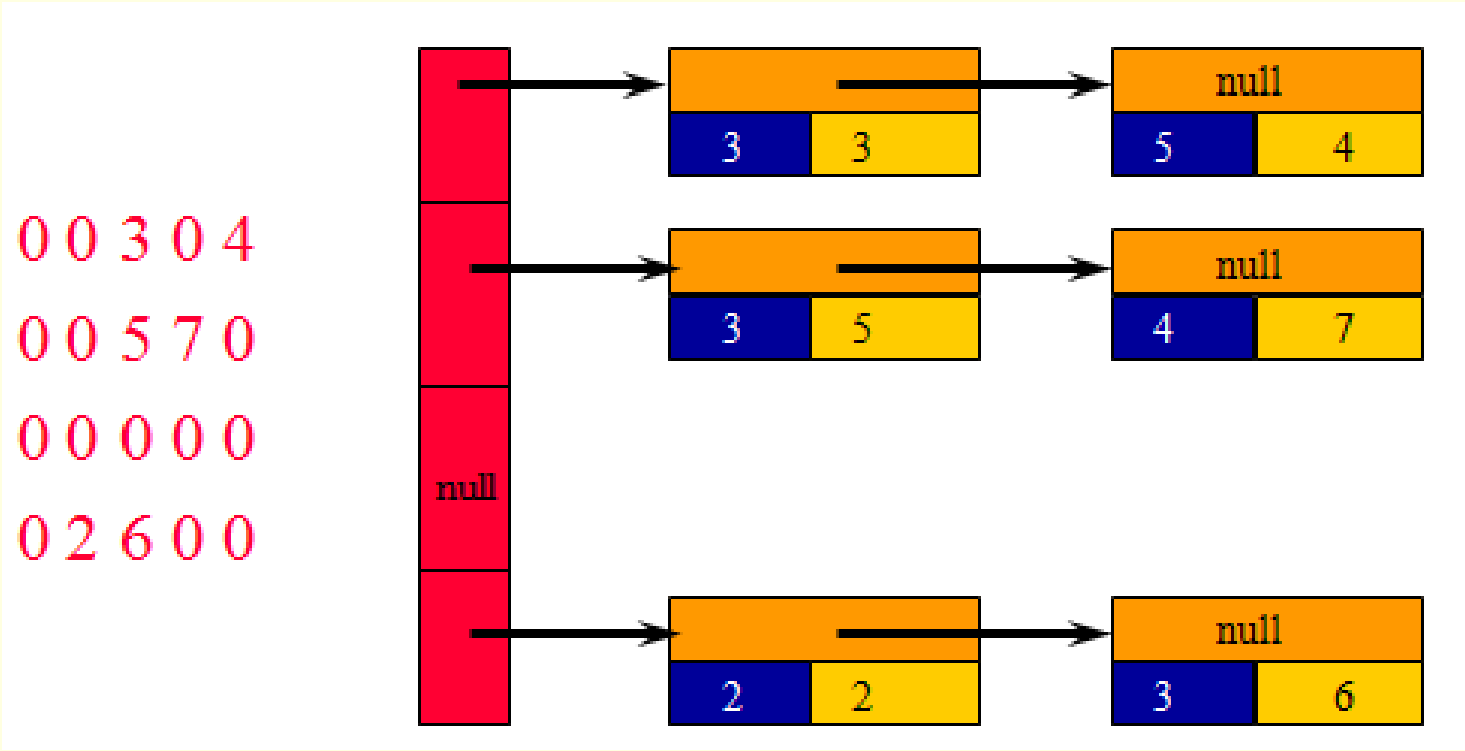
Matrizes esparsas

- Listas cruzadas vs. matriz tradicional

- Em termos de tempo
  - Operações mais lentas em listas cruzadas: acesso não é direto
  - Mas há vantagens: veja o exemplo de operações e ocupação de espaço numa das variações de listas cruzadas

Variações: vetor de ponteiros para linhas somente

Nó



Variações e vantagens

- Matriz 500 x 500 com 1994 elementos não zeros

- Transposição de matrizes
  - 2D array 210 ms
  - Matriz de ponteiros para linhas 12 ms

- Espaço
  - 2D array 500 x 500 x 4 = 1milhão de bytes
  - Lista em vetor 3 x 1994 x 4 = 23.928 bytes
  - Matriz de ponteiros para linhas 23.928 **+** 500 x 4 = 25.928

Exercício

Considere a matriz esparsa abaixo:

$$\begin{pmatrix} 0 & 0 & 2 \\ 3 & -1 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

a) Mostre, graficamente, como ela seria representada com **listas cruzadas**.

b) Declare a ED em C.

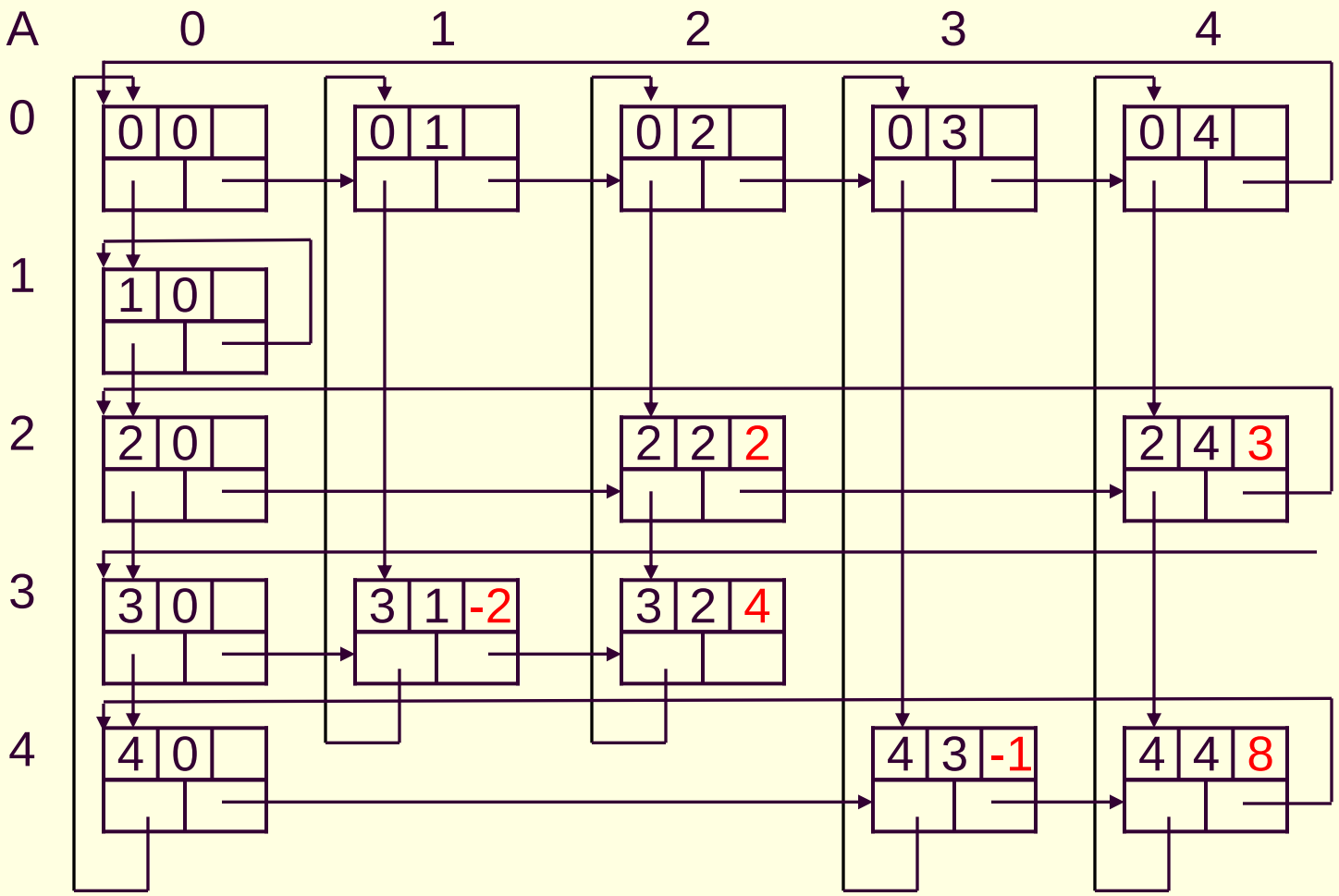
c) Valeria a pena usar a representação com listas cruzadas para a matriz anterior? Justifique.

Solução 3

- 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

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

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



Matrizes Esparsas

- Listas circulares com nós cabeças
  - Quais as vantagens/desvantagens dessa representação?
    - Manipulação mais complexa, mas a matriz pode crescer dinamicamente

Estruturas de Dados em C

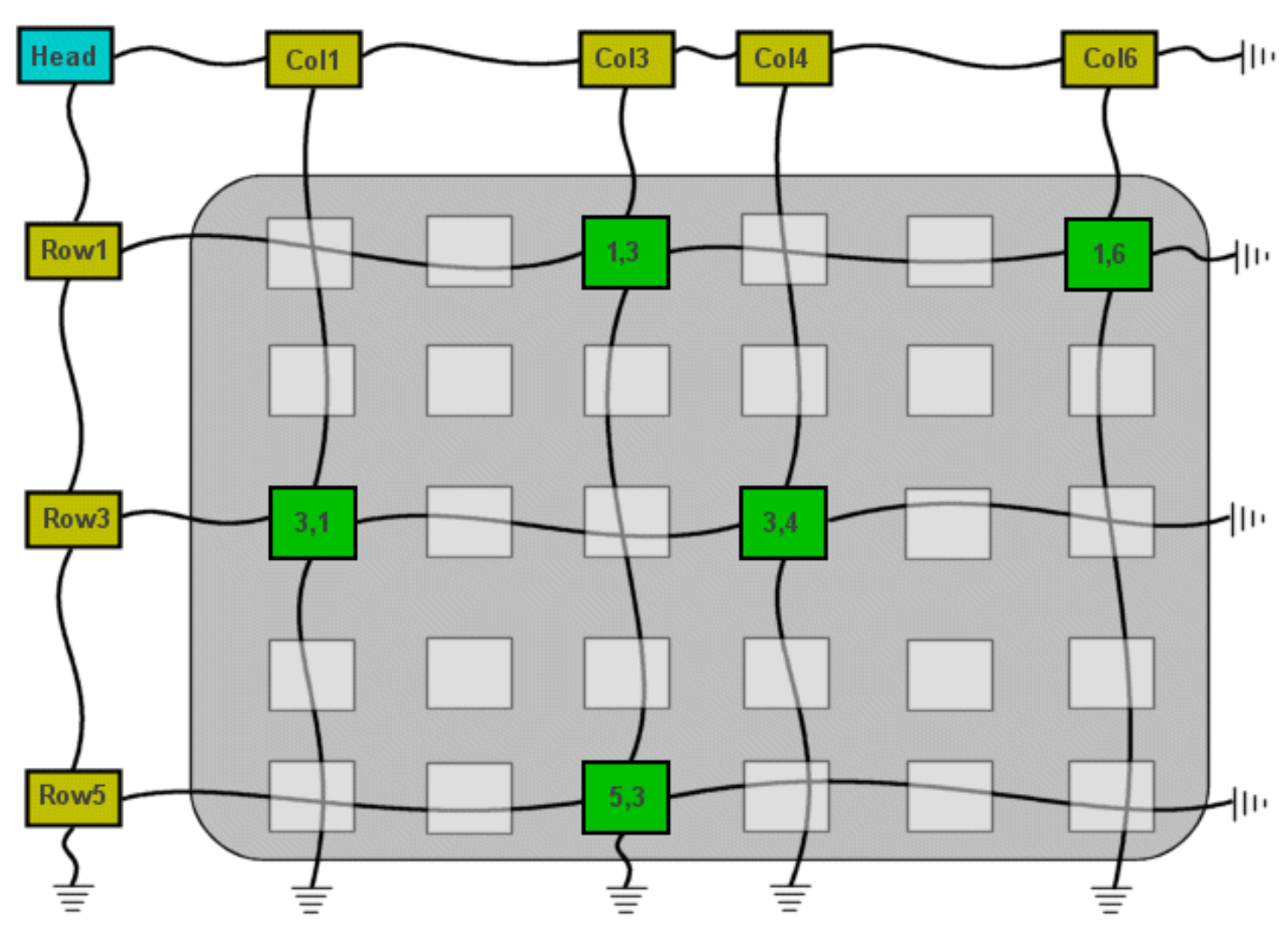
- Com arrays de ponteiros de linhas e colunas

```
struct list_rec {
        int linha,coluna,valor;
        struct list_rec  *proxlin, *proxcol;
      };
typedef  struct list_rec  Rec;
Rec *lin[m], *col[n];
```

- Como Listas Circulares com nó cabeça:

```
Rec *A
```

Variações: listas encadeadas (não circulares)



Operações sobre matrizes esparsas

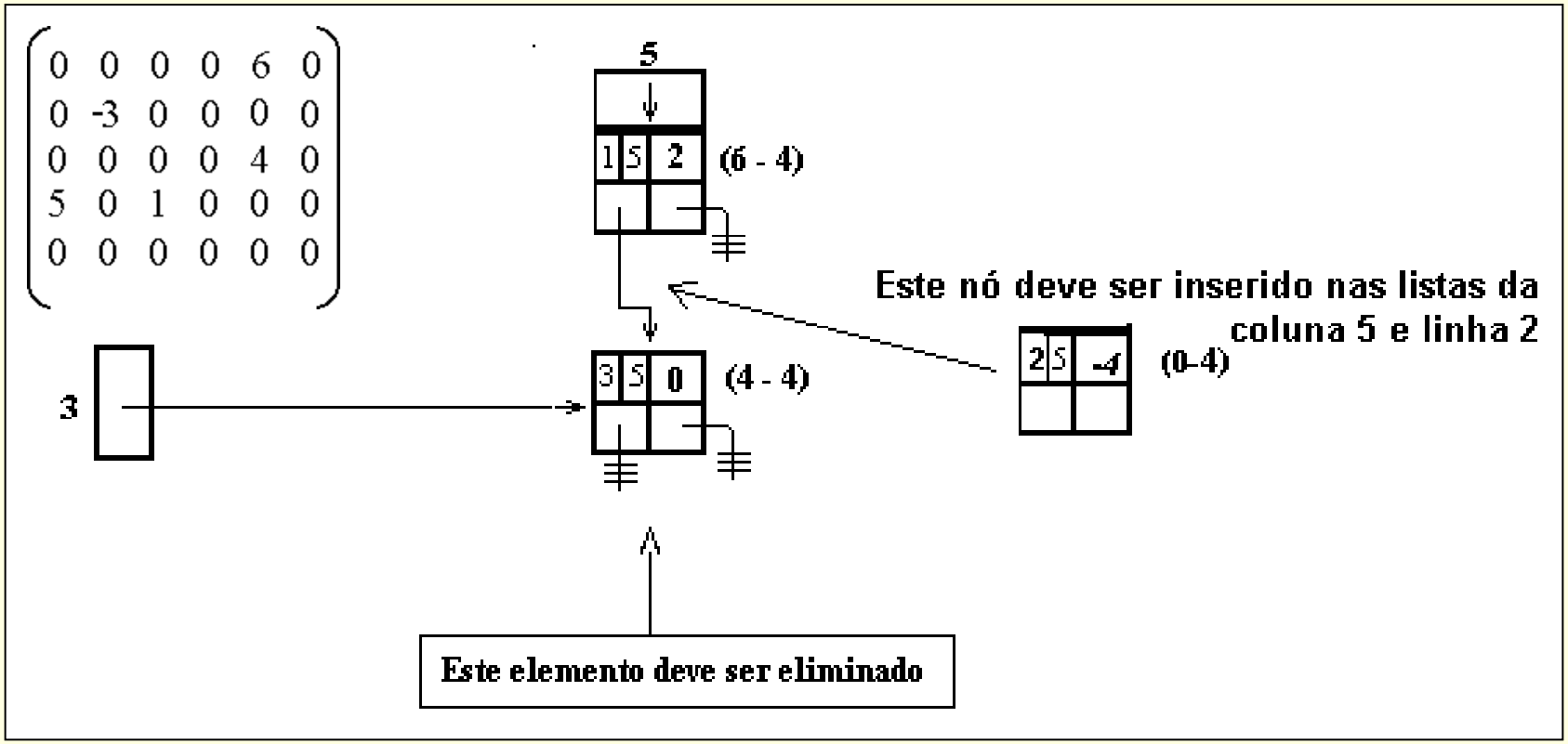
- Em geral
  - Multiplicar uma dada linha ou coluna por uma constante
  - Somar uma constante a todos os elementos de uma linha ou coluna
  - Somar duas matrizes esparsas de igual dimensão
  - Multiplicar matrizes esparsas
  - Transpor matrizes esparsas
  - Inserir, remover ou alterar elementos
  - Etc.

Operações sobre matrizes esparsas

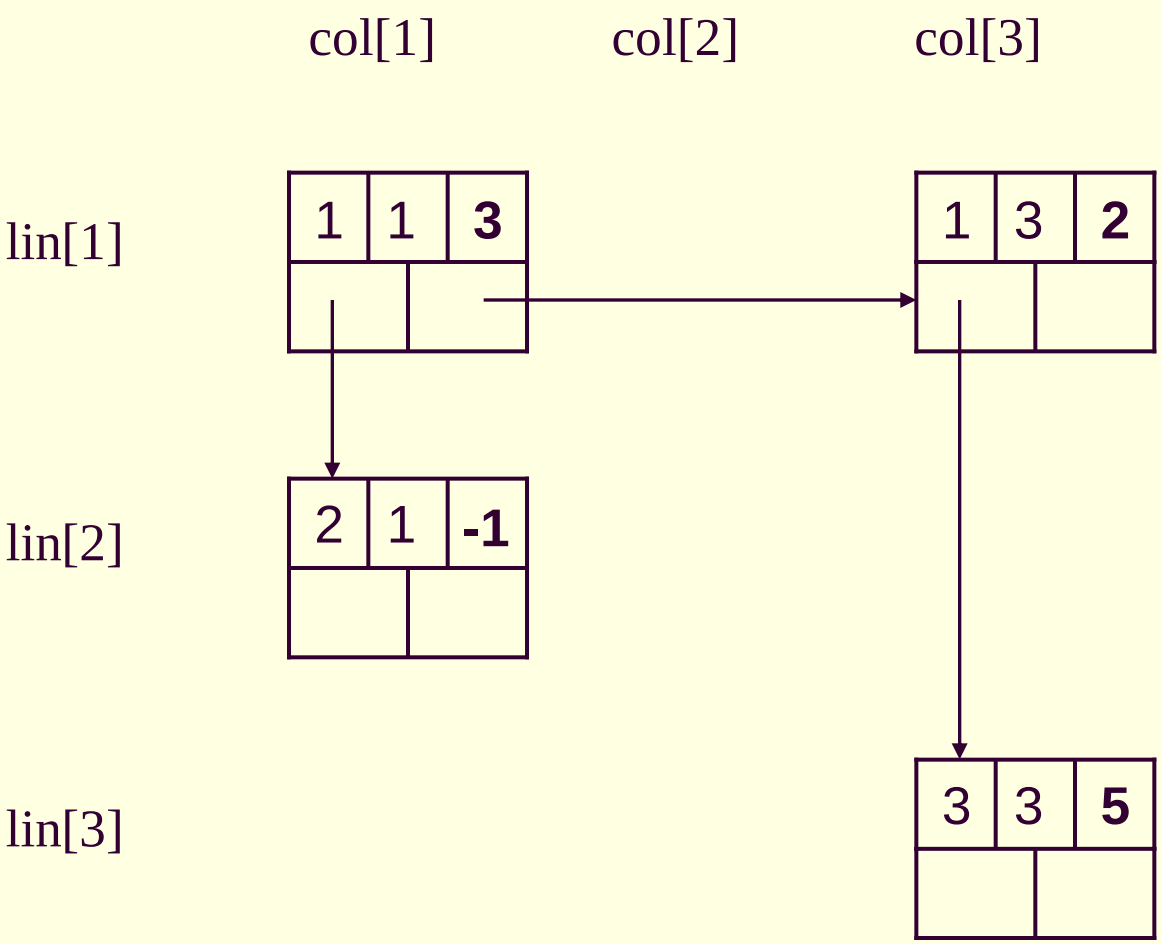
- Após a realização de alguma operação sobre a matriz
  - Quando um elemento da matriz se torna nulo
    - Remoção do elemento
  - Quando algum elemento se torna não nulo
    - Inserção do elemento

Operações sobre matrizes esparsas

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



Exercício: somar -5 a coluna 3



Exercício

```
struct list_rec {
        int linha,coluna,valor;
        struct list_rec  *proxlin, *proxcol;
};
typedef  struct list_rec  Rec;
Rec *lin[m], *col[n];
```

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

  void soma(Rec *lin[ ], Rec *col[ ], int nl, int **j**, int **k**)

  - Supor que existam 2 rotinas prontas:
    - Inserir (i,j,k,lin,col)
    - Eliminar (i,j,lin,col)
  - Depois implementem Inserir e Eliminar

Casos

- (1) Coluna possui apenas valores nulos
- (2) Caso contrário:
  - (2.1) O valor é nulo
  - (2.2) O valor não é nulo
    - (2.2.1) O valor tornou-se nulo
    - (2.2.2) O valor não se tornou nulo

- A função é bem simples, pois o trabalhão é feito por inserir e eliminar!!!
  - Estas 2 tem que atualizar o vetor linha e o vetor coluna

```
void soma(Rec *lin[], Rec *col[], int nl, 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;
        }
    }
}
```

Checar se j está dentro dos limites da matriz. Passem o parâmetro nc também.

Inserindo A[i,j]=k nas listas cruzadas

```
void inserir(int i, int j, int k, Rec *lin[], Rec *col[]){

Rec *p;      /*aponta registro criado */
Rec *q, *qa;      /*ponteiros para pecorrer listas*/

p = malloc(sizeof(Rec));
p->linha = i; p->coluna = j; p->valor = k;

/* inserir na lista da coluna j */
q = col[j]; qa = NULL;
while (q != NULL) {
    if (q->linha < i) { qa = q; q = q ->proxlin;
    }else{  /* achou linha maior */
        if (qa == NULL) /* inserir como 1o. da coluna j */
            col[j] = p;
        else
            qa->proxlin = p; /*inserir entre qa e q*/
        p->proxlin = q;
        break;
    }
}                          /* ... */
```

```
 /*inserir como ultimo da lista col */
 if(q == NULL)
     if (qa = NULL) col[j] = p;
     else qa->proxlin = p; /*após qa*/

 /* inserir na lista da linha i */
 q = lin[i]; qa = NULL;

 while (q != NULL) {
     if (q->coluna < j) {
         qa = q;
         q = q ->proxcol;
     } else { /* achou coluna maior */
          if (qa == NULL) /* inserir como 1o. da linha i */
              lin[ i ] = p;
          else
               qa->proxcol = p; /* inserir entre qa e q */
          p -> proxcol = q;
          break;
     }
 }
/*inserir como ultimo da lista lin */
if(q == NULL);
    if (qa == NULL)
        lin[i] = p;
    else    /*após qa*/
        qa->proxcol = p;

}
```

Removendo A[i,j] das listas cruzadas

```
boolean eliminar(int i, int j, Rec *lin[], Rec *col[]){

Rec *q, *qa;      /*ponteiros para pecorrer listas*/

/* remove da lista da coluna j */
q = col[j]; qa = NULL;
while (q != NULL) {
    if (q->linha < i) {
        qa = q;  q = q ->proxlin;
    }else{  /* achou linha */
        if (qa == NULL)
            /* remove da primeira posição da coluna j */
            col[j] = q->proxlin;
        else /*remove ligaçoes pra q*/
            qa->proxlin = q->proxlin;
        break;
    }
}
```

```
/* se não achou elemento retorna FALSE*/
if(q == NULL)
    return FALSE;

/* remove da lista da linha i */
q = lin[i];
qa = NULL;
while (q != NULL) {
    if (q->coluna < j) {
        qa = q; q = q ->proxcol;
    } else { /* achou coluna*/
        if (qa == NULL)
             /* remove da primeira posição da linha i */
            lin[i] = q->proxcol;
        else /*remove ligaçoes pra q*/
            qa->proxcol = q->proxcol;
        break;
    }
}
/*libera a posição apontada por q*/
return FALSE;
}
```

Exercício

- Implementem o TAD matriz esparsa com listas circulares com nó cabeça