Listas e matrizes esparsas

SCC-202– Algoritmos e Estruturas de Dados I

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)

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

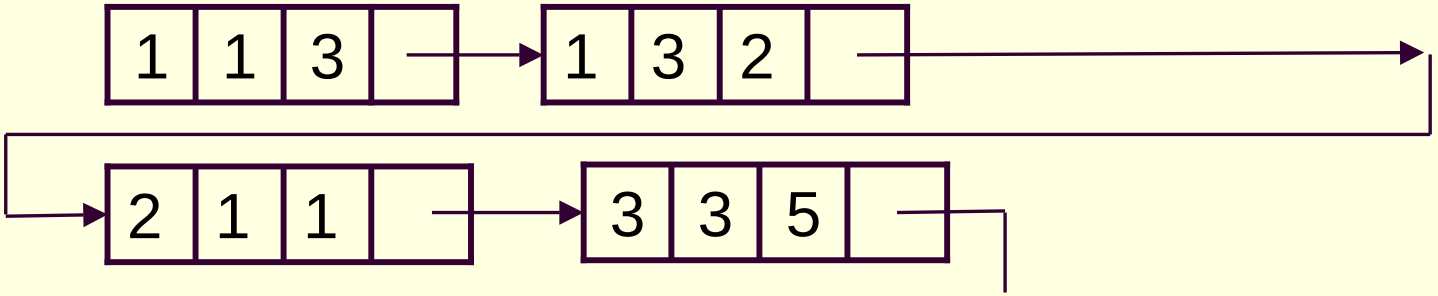
- Listas simples encadeadas

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

linha	coluna	valor	next

Estrutura de um Nó:

- linha, coluna: posição
- valor: ≠ zero
- next: próx nó



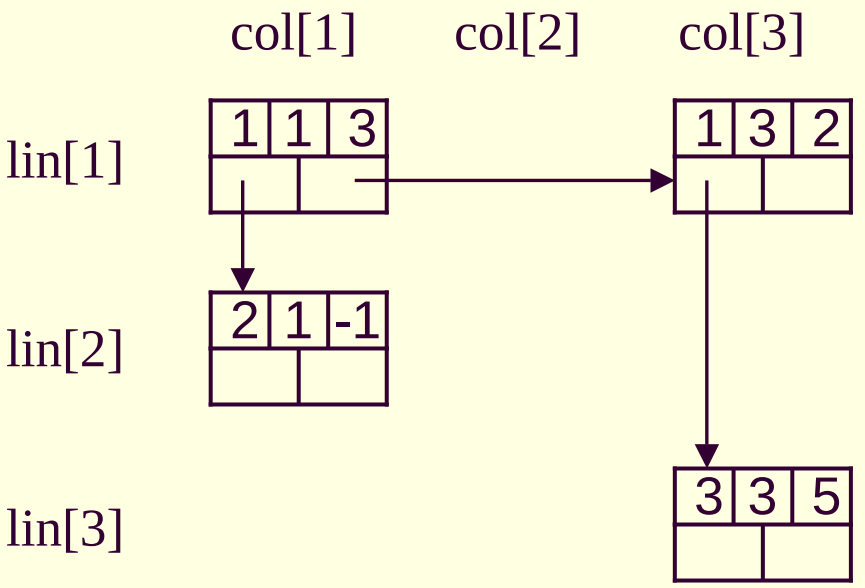
Solução 1

- 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

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
  - Necessidade de avaliação tempo-espaço para cada aplicação
  - Em geral, usam-se listas cruzadas quando no máximo 1/5 dos elementos forem não nulos
    - De onde vem isso?

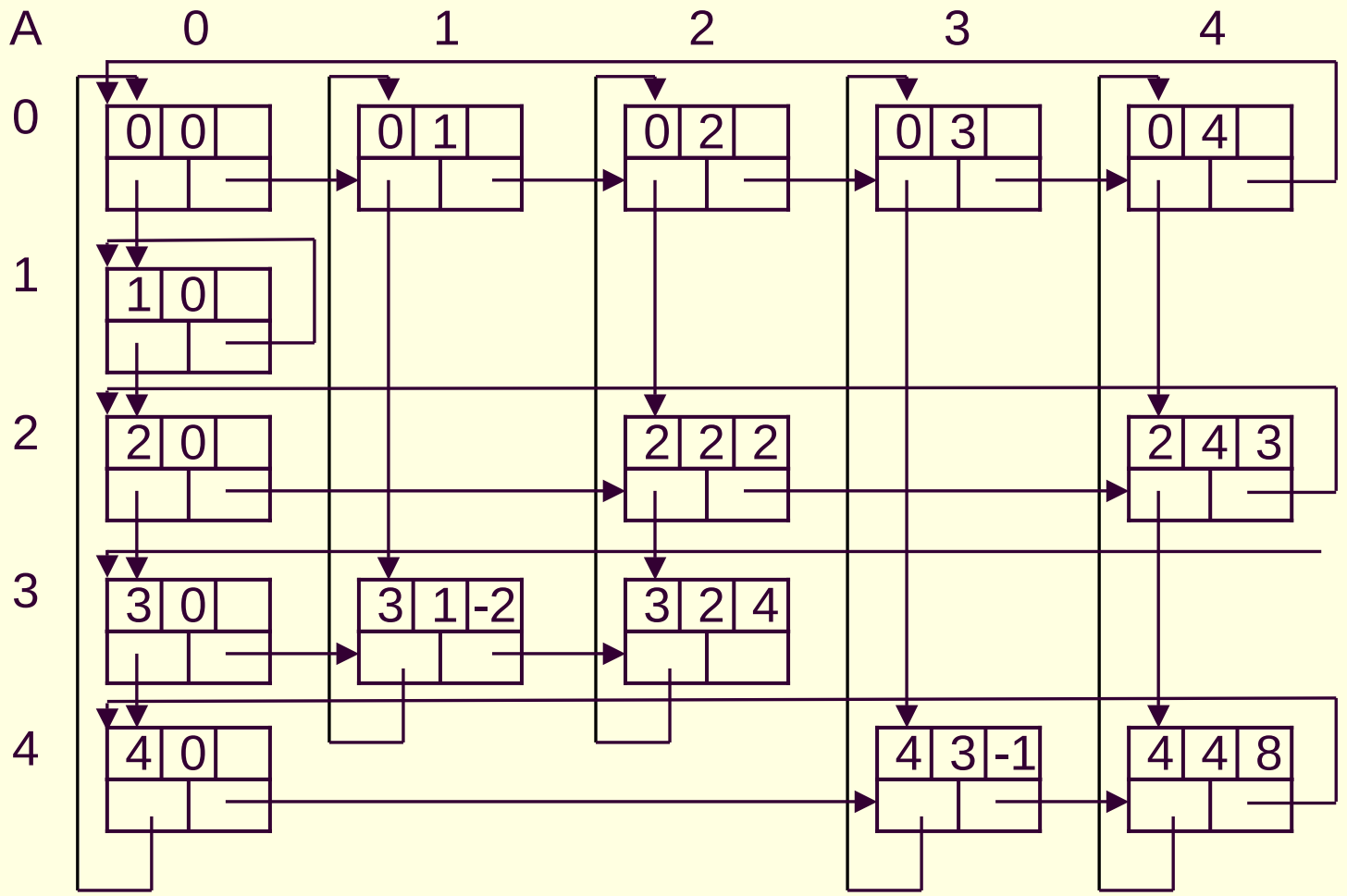
Dica: nl+nc+5n < nl*nc

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 no campo linha ou coluna
      - 1 único ponteiro para a matriz: navegação em qualquer sentido
  - Exemplo

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

Outra solução



Outra solução

- 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?

Operações sobre matrizes esparsas

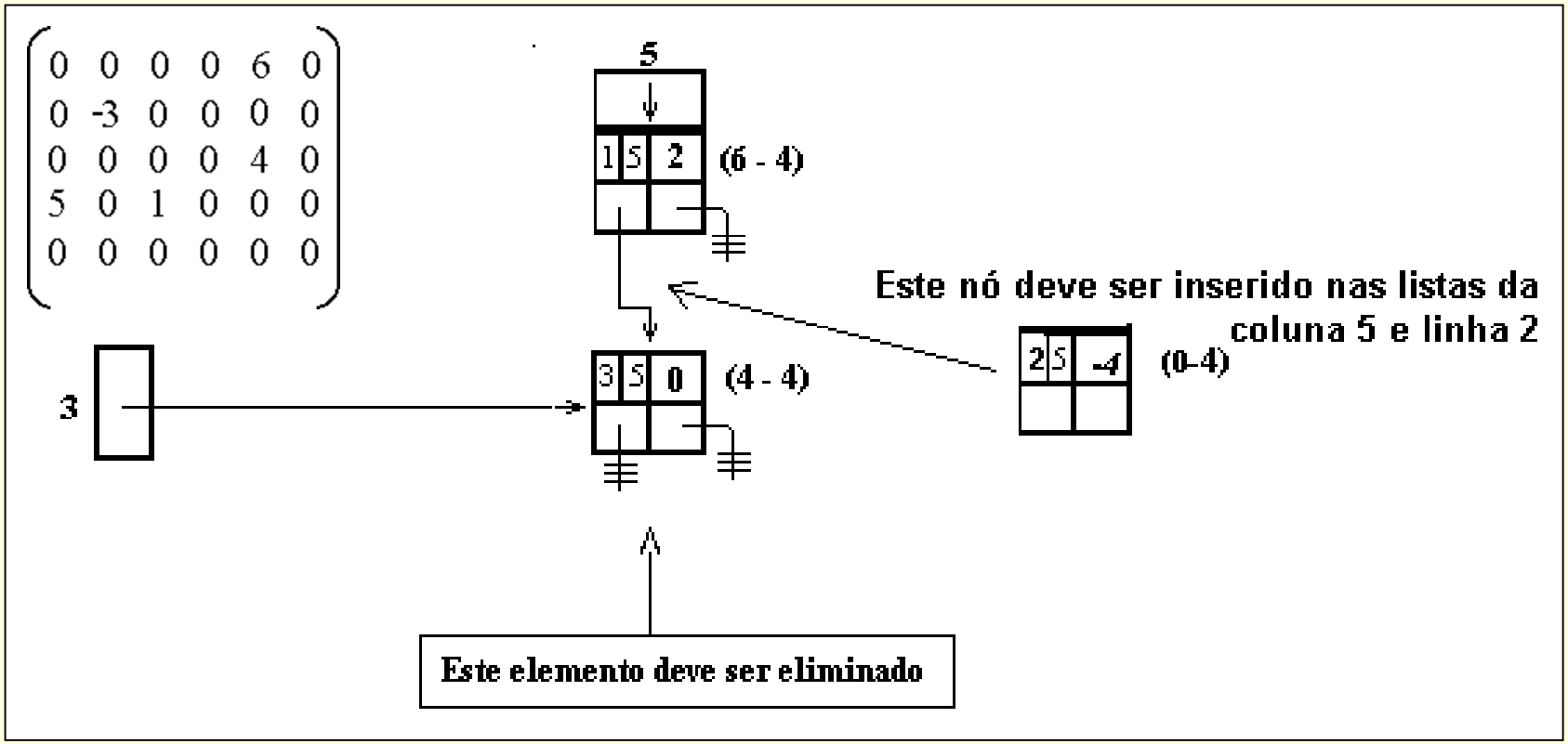
- Em geral
  - Multiplicar uma dada linha ou coluna por uma constante
  - Somar uma constante a todos os elementos de uma linha ou coluna
  - Somar duas matrizes esparsas de igual dimensão
  - Multiplicar matrizes esparsas
  - Transpor matrizes esparsas
  - Inserir, remover ou alterar elementos
  - Etc.

Operações sobre matrizes esparsas

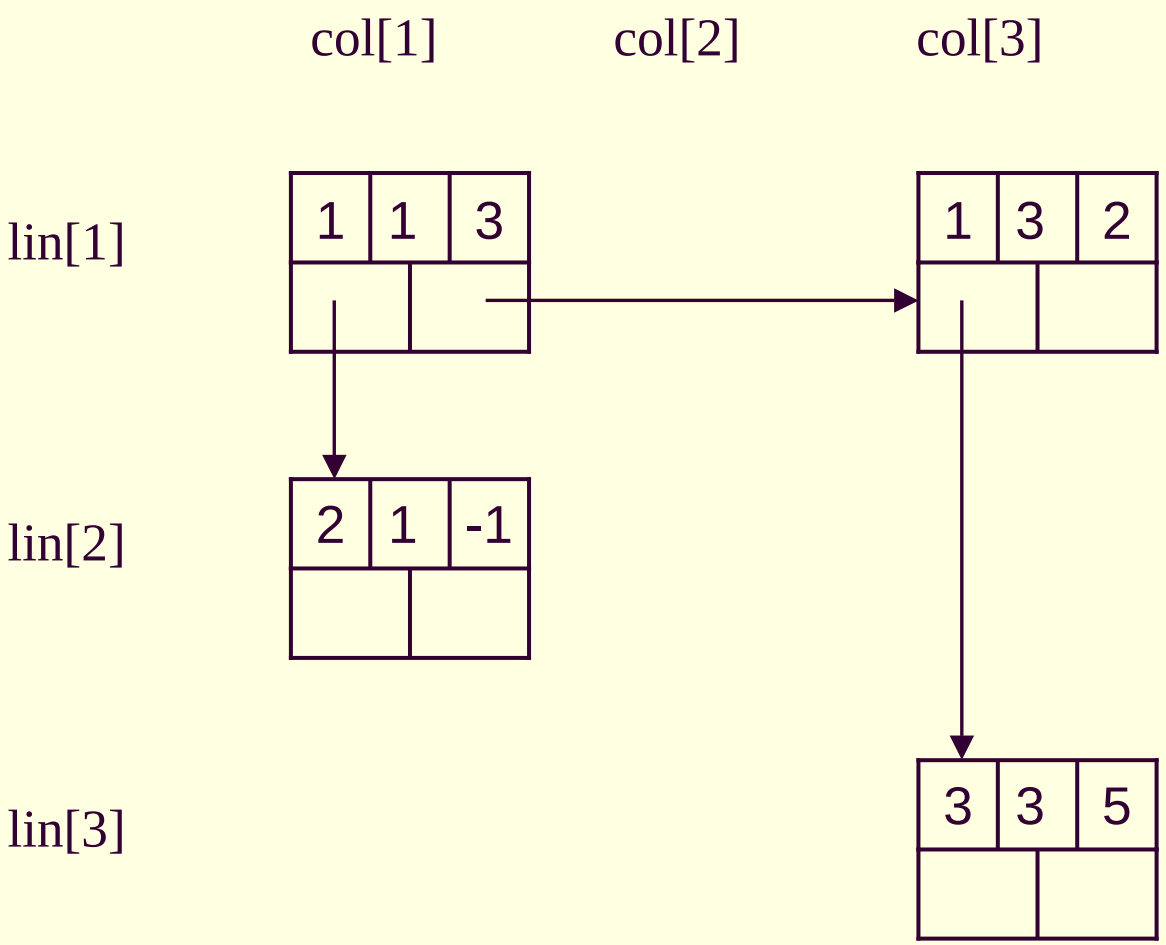
- Após a realização de alguma operação sobre a matriz
  - Quando um elemento da matriz se torna nulo
    - Remoção do elemento
  - Quando algum elemento se torna não nulo
    - Inserção do elemento

Operações sobre matrizes esparsas

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



Exercício: somar -5 a coluna 3



Declare a estrutura 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
```

Exercício

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

```
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;
        }
    }
}
```

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;
    }
}                       /* ... */
```

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

```
/*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;
    }
}
```

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

```
/*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;
    }
}

                        /* ... */
```

Removendo A[i,j] das listas cruzadas

```
/* 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;
    }
}
free(q); /*libera a posição apontada por q*/
return FALSE;
}
```