

SCC0202 – Algoritmos e Estruturas de Dados I

Listas Cruzadas

Prof.: Dr. Rudinei Goularte

(rudinei@icmc.usp.br)

Instituto de Ciências Matemáticas e de Computação - ICMC

Sala 4-229

Conteúdo



- Matrizes Esparsas – Listas Cruzadas
- Representação Alternativa – Listas Cruzadas Circulares

○ Problema

- Representação de matrizes com muitos elementos nulos
 - ▣ P.ex., matriz abaixo, de 5 linhas por 6 colunas: apenas 5 dos 30 elementos são não nulos
 - ▣ Precisamos de uma representação que evite o armazenamento de tantos zeros
 - ▣ Solução: utilizar listas cruzadas como estruturas de dados

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

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

Matrizes esparsas - solução 1

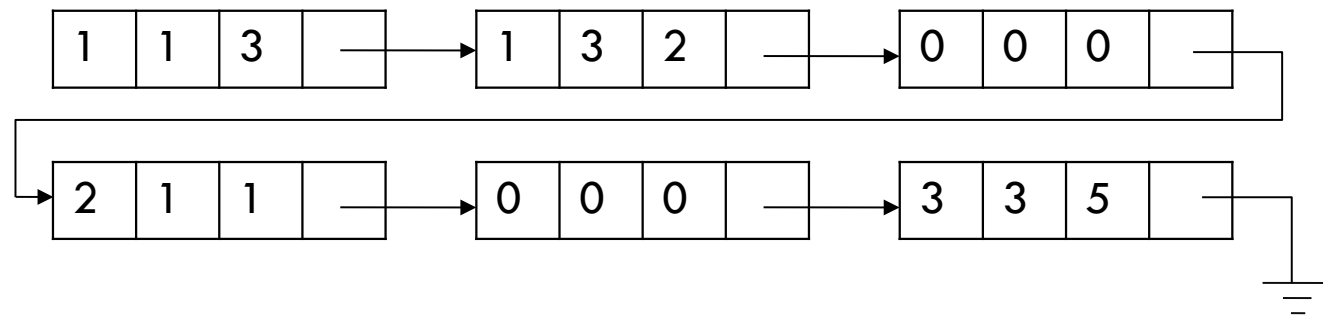
□ Listas simples encadeadas

$$A_{3 \times 3} = \begin{pmatrix} 3 & 0 & 2 \\ 1 & 0 & 0 \\ 0 & 0 & 5 \end{pmatrix}$$

linha	coluna	valor	próximo
-------	--------	-------	---------

Estrutura de um nó:

- linha, coluna: posição
- valor: \neq zero
- próximo: próximo nó



Nós zerados **opcionais**
para auxiliar na
divisão de linhas

Matrizes esparsas - solução 1

□ Desvantagens

▣ Perda da natureza bidimensional de matrizes

■ 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, tem que se passar por várias outras antes

□ Questão

- ▣ Como organizar essa lista, preservando a natureza bidimensional de matriz?

Matrizes esparsas - solução 2

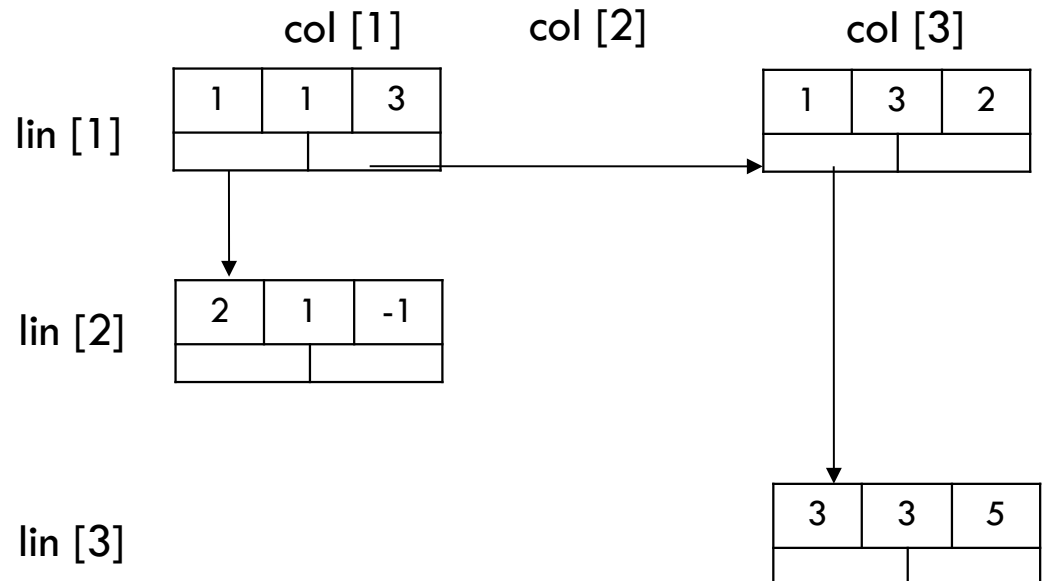
□ Listas cruzadas

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

$$A_{3 \times 3} = \begin{pmatrix} 3 & 0 & 2 \\ -1 & 0 & 0 \\ 0 & 0 & 5 \end{pmatrix}$$

Estrutura de um nó:

linha	coluna	valor
abaixo		direita



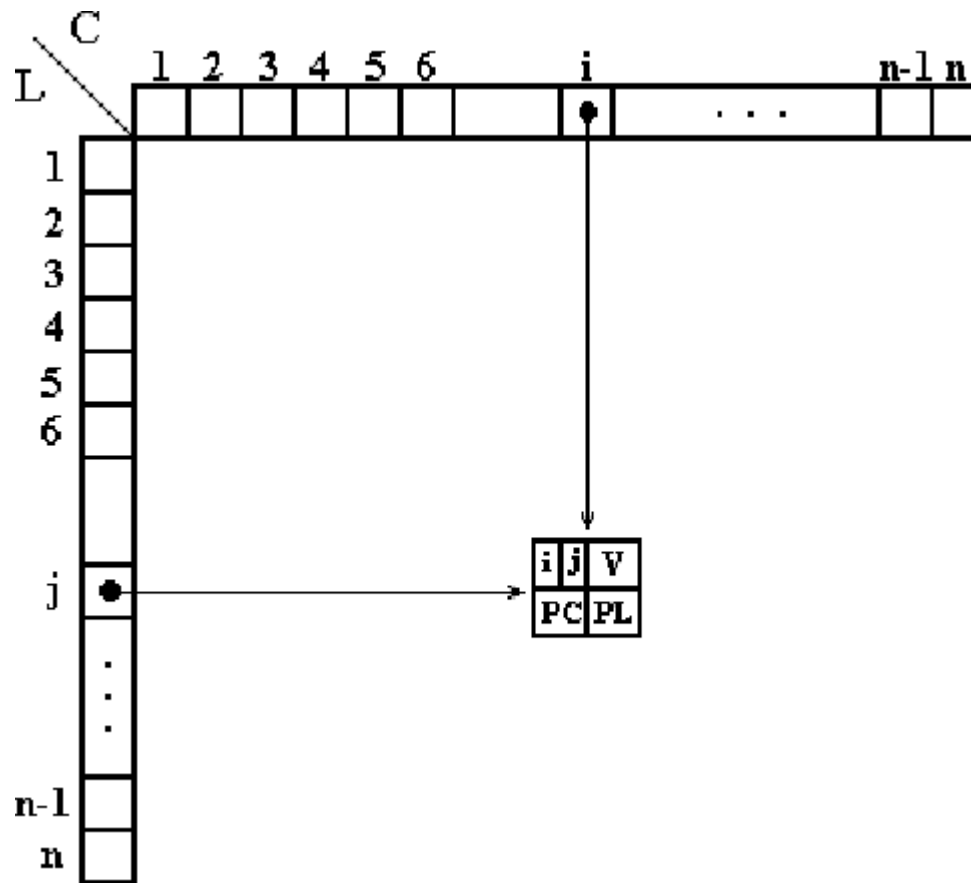
Matrizes esparsas - solução 2

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

Representação por Listas Cruzadas

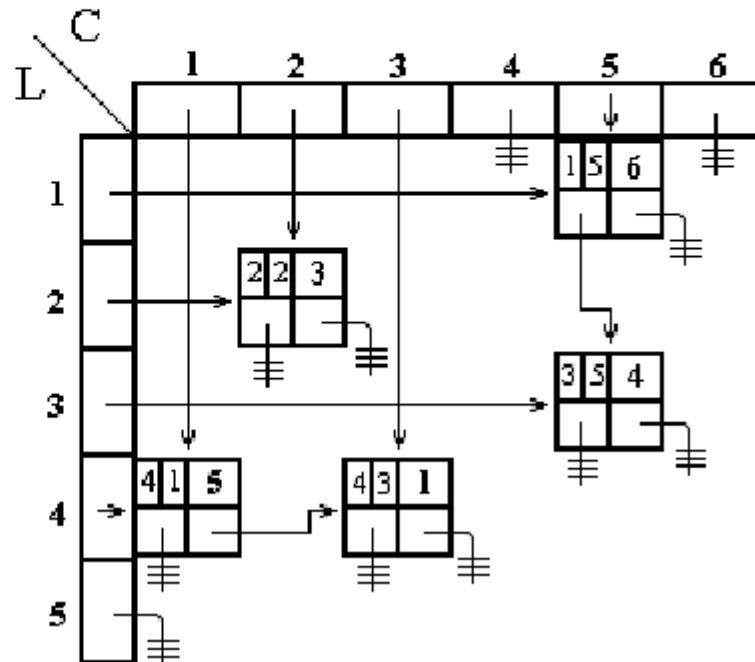
- Cada elemento identificado pela sua linha, coluna, e valor
- Cada elemento a_{ij} não-nulo pertence a uma lista de valores não nulos da linha i e também a uma lista de valores não nulos da coluna j
- Assim, para matriz de nl linhas e nc colunas, teremos nl listas de linhas e nc listas de colunas

Listas Cruzadas



Listas Cruzadas

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



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
 - ▣ Necessidade de avaliação tempo-espaco para cada aplicação
 - ▣ Em geral, usa-se listas cruzadas quando **no máximo** $1/5$ dos elementos forem não nulos
 - De onde vem isso?
 - Dica: $n_l + n_c + 5n < n_l * n_c$

TAD Matriz Esparsa

- Pode-se criar um TAD bastante simples para matrizes esparsas
- Operações principais
 - ▣ *criar_matriz(n, m)* : cria uma nova matriz esparsa vazia com n linhas e m colunas
 - ▣ *set($M, lin, col, valor$)* : define um valor na posição (lin, col) da matriz esparsa M
 - ▣ *get(M, lin, col)* : retorna o valor na posição (lin, col) da matriz esparsa M

TAD Matriz Esparsa

- Operações auxiliares (podem ser criadas a partir das operações principais)
 - ▣ *somar_matriz*($M1$, $M2$, R) : Soma as matrizes $M1$ e $M2$ e armazena o resultado em R
 - ▣ *multiplicar_matriz*($M1$, $M2$, R) : Multiplica as matrizes $M1$ e $M2$ e armazena o resultado em R
 - ▣ *somar_coluna*(M , V , col) : Soma uma constante V a todos os elementos da coluna col da Matriz M
 - ▣ *somar_linha*(M , V , lin) : Soma uma constante V a todos os elementos da linha lin da Matriz M
 - ▣ E mais: inverter, transpor, calcular determinante, etc...

Estrutura de Dados

- A implementação é facilitada se as listas contêm o nó cabeça

```
1 typedef struct matriz_esparsa MATRIZ_ESPARSA;
2
3 typedef struct CELULA {
4     int linha;
5     int coluna;
6     float valor;
7     struct CELULA *direita;
8     struct CELULA *abaixo;
9 } CELULA;
10
11 struct matriz_esparsa {
12     CELULA **linhas;
13     CELULA **colunas;
14     int nr_linhas;
15     int nr_colunas;
16 };
```

Operações

- Vamos implementar as operações `criar_matriz(...)`, `apagar_matriz(...)`, `set(...)` e `get(...)` do conjunto de operações principais
- As demais operações principais e auxiliares ficam como exercício
- Entretanto, vamos discutir alguns aspectos importantes dessas operações

TAD Matriz Esparsa

```
1  #ifndef MATRIZ_ESPARSA_H
2  #define MATRIZ_ESPARSA_H
3
4  typedef struct matriz_esparsa MATRIZ_ESPARSA;
5
6  MATRIZ_ESPARSA *criar_matriz(int nr_linhas, int nr_colunas);
7  void apagar_matriz(MATRIZ_ESPARSA **matriz);
8
9  int set(MATRIZ_ESPARSA *matriz, int lin, int col, float val);
10 float get(MATRIZ_ESPARSA *matriz, int lin, int col);
11
12 void imprimir_matriz(MATRIZ_ESPARSA *matriz);
13
14 #endif
```

Criar Matriz

12

```
1  MATRIZ_ESPARSA *criar_matriz(int nr_linhas, int nr_colunas) {
2      MATRIZ_ESPARSA *mat = (MATRIZ_ESPARSA *) malloc(sizeof (MATRIZ_ESPARSA));
3
4      if (mat != NULL) {
5          int i;
6          mat->nr_colunas = nr_colunas;
7          mat->nr_linhas = nr_linhas;
8          mat->colunas = (CELULA **) malloc(sizeof (CELULA *) * nr_colunas);
9          mat->linhas = (CELULA **) malloc(sizeof (CELULA *) * nr_linhas);
10
11         if (mat->colunas != NULL && mat->linhas != NULL) {
12             for (i = 0; i < nr_colunas; i++)
13                 mat->colunas[i] = NULL;
14
15             for (i = 0; i < nr_linhas; i++)
16                 mat->linhas[i] = NULL;
17         }
18     }
19
20     return (mat);
21 }
```

Apagar Matriz

```
1 void apagar_matriz(MATRIZ_ESPARSA **matriz) {
2     int i;
3     for (i = 0; i < (*matriz)->nr_linhas; i++) {
4         if((*matriz)->linhas[i] != NULL){
4             CELULA *paux = (*matriz)->linhas[i]->direita;
5
6             while (paux != NULL) {
7                 CELULA *prem = paux;
8                 paux = paux->direita;
9                 free(prem); prem = NULL;
10            }
11        }
12        free((*matriz)->linhas[i]); (*matriz)->linha[i] = NULL;
13    }
15    free((*matriz)->linhas); (*matriz)->linhas = NULL;
16    free((*matriz)->colunas); (*matriz)->colunas = NULL;
17    free((*matriz));
18    *matriz = NULL;
19}
```

Definir Valor

```
1 int set(MATRIZ_ESPARSA *matriz, int lin, int col, float val) {
2     CELULA *p, *q, *qa;
3
4     p = (CELULA *) malloc(sizeof(CELULA));
5     if ((p == NULL) || (lin > matriz->nr_linhas) || (col > matriz->nr_colunas)) return(0);
6     p->linha = lin; p->coluna = col; p->valor = val;
7
8     //inserir na lista da coluna col
9     q = matriz->colunas[col]; qa = NULL;
10    while(q != NULL){
11        if (q->linha < lin){
12            qa = q;
13            q = q->abaixo;
14        }else{ //achou linhas maior
15            if (qa == NULL)
16                matriz->colunas[col] = p;
17            else
18                qa->abaixo = p;
19            p->abaixo = q;
20            break;
21        }
22    }
23    //inserir como último da lista colunas
24    if (q == NULL)
25        if (qa == NULL)
26            matriz->colunas[col] = p;
27        else
28            qa->abaixo = p;
29
30    // algoritmo simetrico para as linhas
```

Retornar Valor

```
1 float get(MATRIZ_ESPARSA *matriz, int lin, int col) {
2     if (lin < matriz->nr_linhas && col < matriz->nr_colunas) {
3         CELULA *paux = matriz->linhas[lin];
4         if (paux != NULL){
5             while (paux->direita != NULL && paux->direita->coluna <= col)
6                 paux = paux->direita;
7
8             if (paux->coluna == col)
9                 return (paux->valor);
10        }
11    }
12    return (0);
13 }
```

Operações

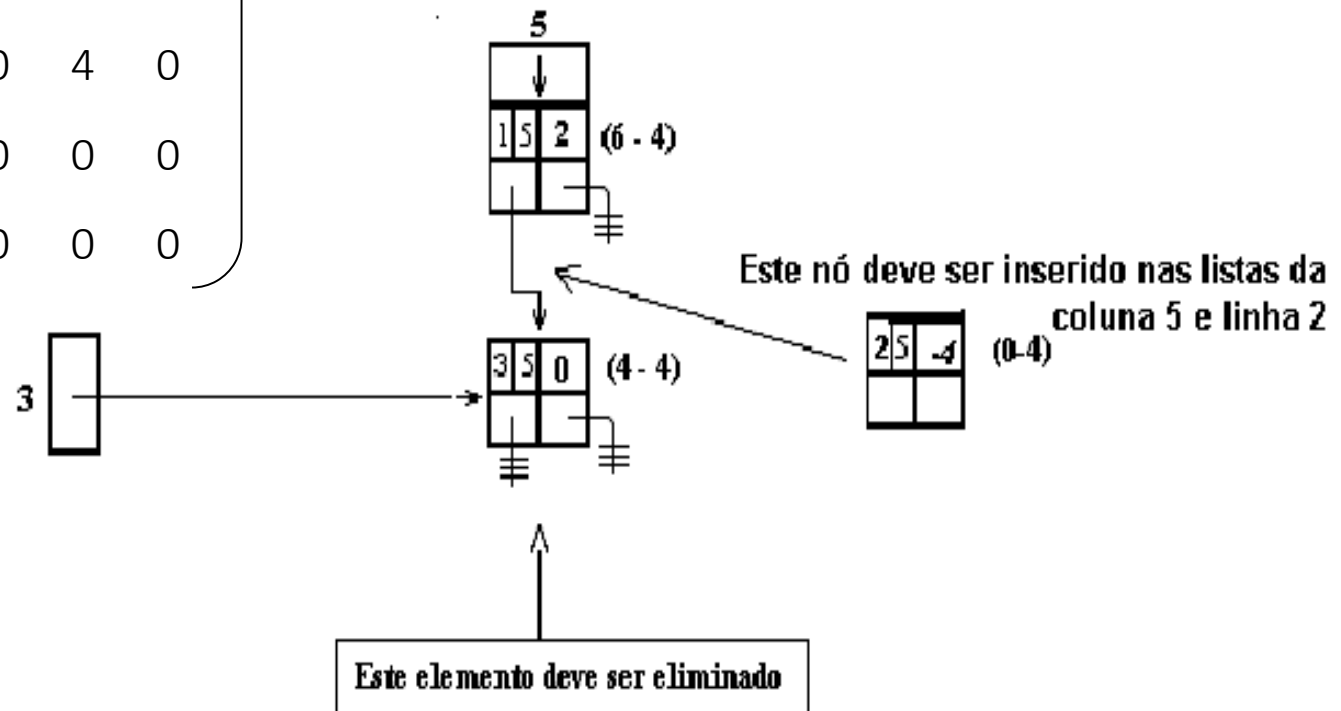
- E quando um elemento da matriz original se torna não nulo, em consequência de alguma operação? É necessário inserir na estrutura?
- E quando um elemento da matriz original se tornar nulo? É necessário eliminar da estrutura?

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

Operações

- P.ex., somar -4 à coluna 5

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



Desempenho (Fator Espaço)

- Quando a representação por listas cruzadas é vantajosa em relação à representação convencional?
- Fator Espaço. Suponhamos
 - ▣ matriz esparsa que armazena inteiros
 - ▣ ponteiro ocupa o mesmo espaço de memória que um inteiro
- Matriz Esparsa (Listas Cruzadas)
 - ▣ Espaço ocupado por matriz de nl linhas, nc colunas e n valores não-nulos
 - ▣ há ganho de espaço, quando um número inferior a $1/5$ dos elementos da matriz forem não nulos
- Na representação bidimensional: espaço total: $nl \times nc$

Desempenho (Fator Tempo)

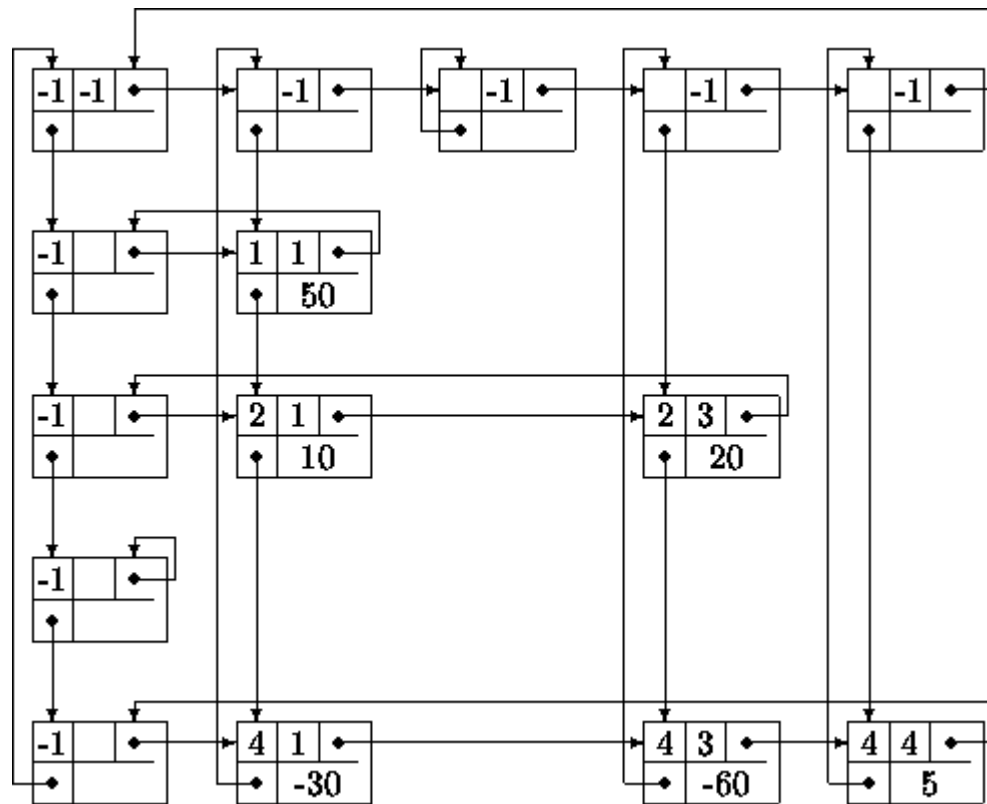
- As operações sobre listas cruzadas podem ser mais lentas e complexas do que para o caso bidimensional
- Portanto, para algumas aplicações, deve ser feita uma avaliação do compromisso entre tempo de execução e espaço alocado

Representação Alternativa – Listas Cruzadas Circulares

- Existem ocasiões nas quais não se sabe a princípio qual será o número máximo de linhas ou colunas da matriz esparsa
- Nessas situações, os vetores Coluna e Linha podem ser substituídos por listas ligadas circulares


Representação alternativa

$$\begin{pmatrix} 50 & 0 & 0 & 0 \\ 10 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 \\ -30 & 0 & -60 & 5 \end{pmatrix}$$



Exercícios

- Desenvolva procedimentos para (listas não circulares):
 - ▣ Atualizar o elemento a_{ij}
 - Modificar função “set” para não criar, desnecessariamente, um novo nó nesse caso.
 - ▣ Somar a constante c todos os elementos da coluna j
 - Pode resultar em inserção ou eliminação nas listas.

- 
- Material baseado nos originais produzidos pelos professores:
 - ▣ Gustavo E. de A. P. A. Batista
 - ▣ Fernando V. Paulovich