SCC-223 Estruturas de Dados I

Matrizes Esparsas

Profa. Elaine Parros Machado de Sousa

ICMC USP SÃO CARLOS

Aula baseada em material do professor Rudinei Goularte.

Grupo de Bases de Dados e Imagens
USP - São Carlos

O Problema...

- Representação de matrizes com muitos elementos nulos => **MATRIZ ESPARSA**
  - Ex: matriz de 5 linhas por 6 colunas: apenas 5 dos 30 elementos são não nulos
  - Representação que evite o armazenamento de tantos zeros?

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

Matriz tradicional

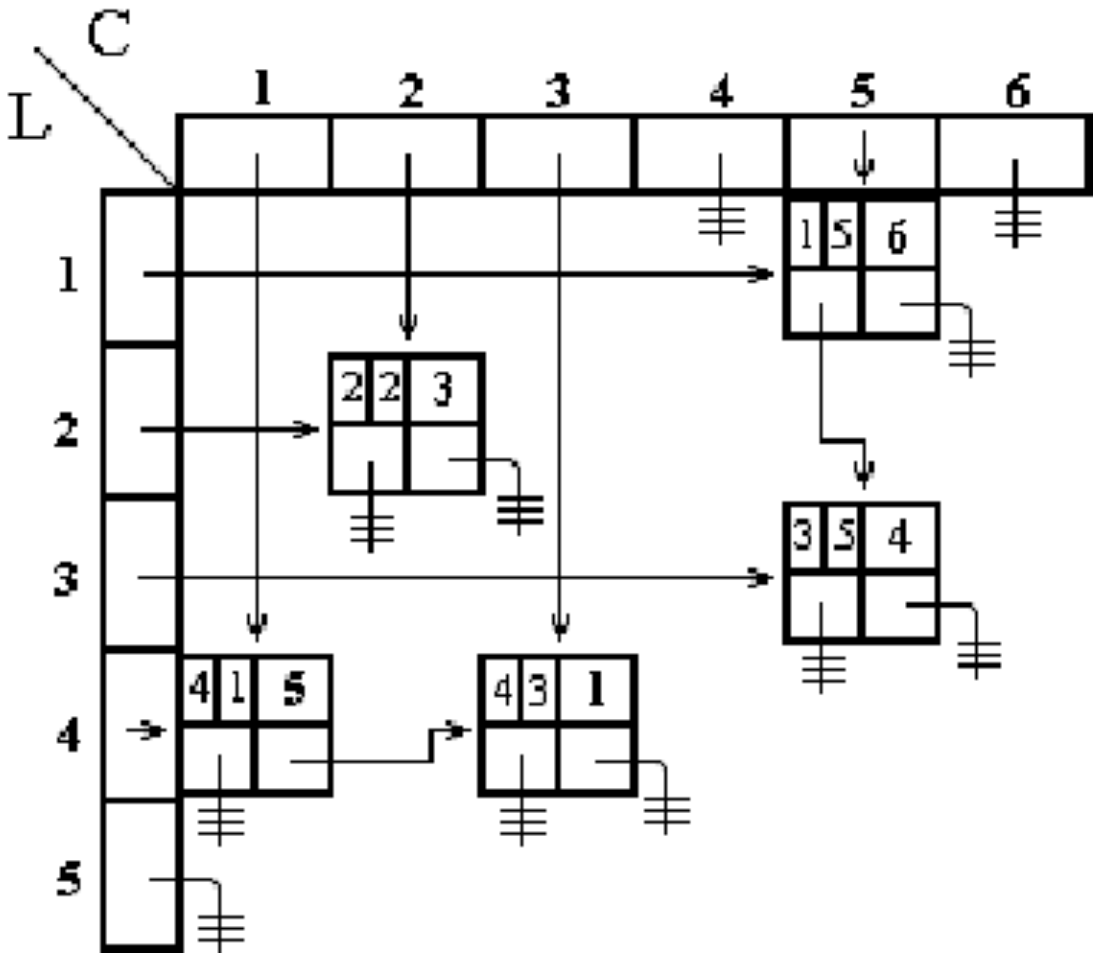
- Vantagem
  - Acesso direto a cada elemento da matriz
    - Algoritmos simples
- Desvantagem em Matrizes Esparsas
  - Muito espaço para armazenar zeros

Matrizes esparsas

- Necessidade
  - Método alternativo para representação de matrizes esparsas
- Solução => **Listas cruzadas**
  - estrutura de lista encadeada contendo somente os elementos não nulos
  - para cada matriz:
    - um vetor com N ponteiros para as linhas
    - um vetor com M ponteiros para as colunas

Matrizes esparsas

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



Estrutura de um nó:

linha	coluna	valor
abaixo		direita

Representação por Listas Cruzadas

- Cada elemento identificado pela sua **linha**, **coluna**, e **valor**
- Cada elemento a_{ij} não-nulo pertence a duas listas
  - uma de valores não nulos da linha ***i*** e
  - outra de valores não nulos da coluna ***j***
- Para matriz de ***nl*** linhas e ***nc*** colunas
  - ***nl*** listas de linhas
  - ***nc*** listas de colunas

Matrizes esparsas

- Listas cruzadas vs. matriz tradicional
  - **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
  - **Tempo**
    - Operações mais lentas em listas cruzadas **=>** acesso não é direto
  - Necessidade de avaliação tempo-espaço para cada aplicação
  - Em geral, usa-se listas cruzadas quando **no máximo** 1/5 dos elementos forem não nulos, para matrizes "muito grandes" (*nl*nc >> nl + nc*)

nl+nc+5n < nl * nc

TAD Matriz Esparsa

- Um TAD 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
  - ***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...

Exemplo de implementação do TAD

```
typedef struct celula_ CELULA;

typedef struct celula_ {
   int linha;
   int coluna;
   float valor;
   CELULA *direita;
   CELULA *abaixo;
};

struct matriz_esparsa_ {
  CELULA **linhas;
  CELULA **colunas;
  int nr_linhas;
  int nr_colunas;
};
```

Exemplo de implementação do TAD

```
#ifndef MATRIZ_ESPARSA_H
#define MATRIZ_ESPARSA_H

typedef struct matriz_esparsa_ MATRIZ_ESPARSA;

MATRIZ_ESPARSA* criar_matriz(int nr_linhas, int nr_colunas);
void apagar_matriz(MATRIZ_ESPARSA **matriz);
int set(MATRIZ_ESPARSA *matriz, int lin, int col, float val);
float get(MATRIZ_ESPARSA *matriz, int lin, int col);
void imprimir_matriz(MATRIZ_ESPARSA *matriz);

#endif
```

```
MATRIZ_ESPARSA *criar_matriz(int nr_linhas, int nr_colunas) {

  MATRIZ_ESPARSA *mat = (MATRIZ_ESPARSA *) malloc(sizeof (MATRIZ_ESPARSA));
  if (mat != NULL) {
       int i;
       mat->nr_colunas = nr_colunas;
       mat->nr_linhas = nr_linhas;
       mat->colunas = (CELULA **) malloc(sizeof (CELULA *) * nr_colunas);
       mat->linhas = (CELULA **) malloc(sizeof (CELULA *) * nr_linhas);

      if (mat->colunas != NULL && mat->linhas != NULL) {
         for (i = 0; i < nr_colunas; i++)
            mat->colunas[i] = NULL;

         for (i = 0; i < nr_linhas; i++)
            mat->linhas[i] = NULL;
      }
   }
   return (mat);
 }
```

```
void apagar_matriz(MATRIZ_ESPARSA **matriz) {
    int i;
    for (i = 0; i < (*matriz)->nr_linhas; i++) {
       if((*matriz)->linhas[i] != NULL){
         CELULA *paux = (*matriz)->linhas[i]->direita;
         while (paux != NULL) {
            CELULA *prem = paux;
            paux = paux->direita;
            free(prem);
         }
       }
       free((*matriz)->linhas[i]);
    }
   free((*matriz)->linhas);
   free((*matriz)->colunas);
   free((*matriz));
  *matriz = NULL;
}
```

Exercício – não precisa entregar

- Implemente as funções ***set*** e ***get*** para o TAD Matriz Esparsa