Matrizes Esparsas – Utilização de Listas por meio de Estruturas Auto-referenciadas

Objetivos: Sedimentar os conceitos de Listas Encadeadas através da implementação de Matrizes Esparsas.

Descrição: Matrizes esparsas são matrizes nas quais a maioria das posições é preenchida por zeros. Para essas matrizes, podemos economizar um espaço significativo de memória se apenas os termos diferentes de zero forem armazenados. As operações usuais sobre essas matrizes (somar, multiplicar, inverter, pivotar) também podem ser feitas em tempo muito menor se não armazenarmos as posições que contêm zeros.

Uma maneira eficiente de representar estruturas com tamanho variável e/ou desconhecido é com o emprego de alocação encadeada, utilizando listas. Vamos usar essa representação para armazenar as matrizes esparsas. Cada coluna da matriz será representada por uma lista linear circular com uma célula cabeça. Da mesma maneira, cada linha da matriz também será representada por uma lista linear circular com uma célula cabeça. Cada célula da estrutura, além das células cabeça, representará os termos diferentes de zero da matriz e deverá ser como no código abaixo:

```
Class Celula {
      Celula direita, abaixo;
      int linha, coluna;
      float valor;
}
```

O campo *abaixo* deve ser usado para referenciar o próximo elemento diferente de zero na mesma coluna. O campo *direita* deve ser usado para referenciar o próximo elemento diferente de zero na mesma linha. Dada uma matriz *A*, para um valor *A(i,j)* diferente de zero, deverá haver uma célula com o campo *valor* contendo *A(i,j)*, o campo *linha* contendo *i* e o campo *coluna* contendo *j*. Essa célula deverá pertencer à lista circular da linha *i* e também deverá pertencer à lista circular da coluna *j*. Ou seja, cada célula pertencerá a duas listas ao mesmo tempo. Para diferenciar as células cabeça, coloque -1 nos campos *linha* e *coluna* dessas células.

Considere a seguinte matriz esparsa:

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

A representação da matriz *A* pode ser vista na Figura 1. Com essa representação, uma matriz esparsa *m* x *n* com *r* elementos diferentes de zero gastará (*m* + *n* + *r*) células. É bem verdade que cada célula ocupa vários bytes na memória; no entanto, o total de memória usado será menor do que as *m* x *n* posições necessárias para representar a matriz toda, desde que *r* seja suficientemente pequeno.

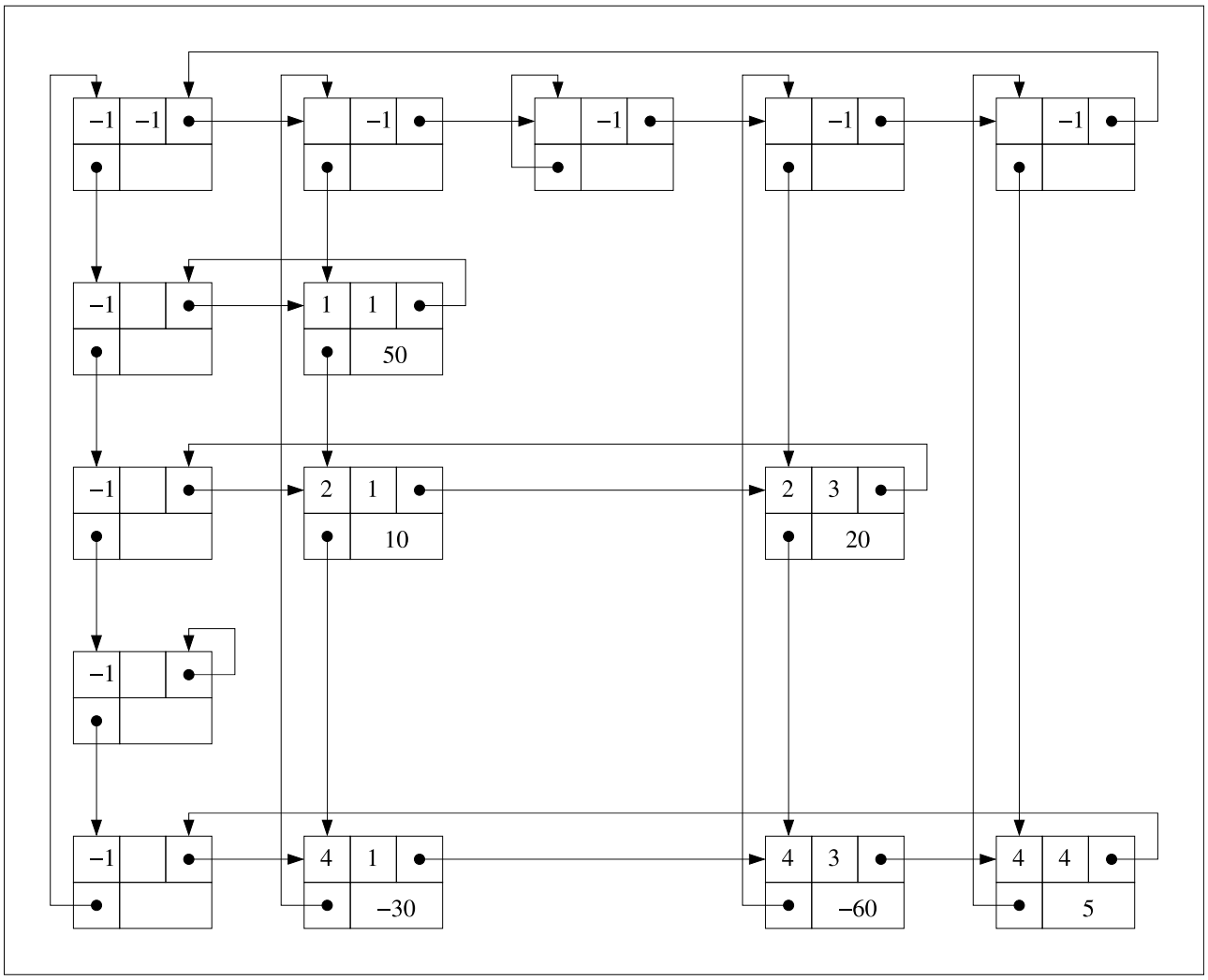


**Figura 1.** Exemplo de Matriz Esparsa.

Dada a representação de listas duplamente encadeadas, o trabalho consiste em desenvolver em Java um tipo abstrato de dados Matriz com as seguintes operações, conforme esta especificação:

a) `void imprimeMatriz()`
Esse método imprime a matriz *A* (uma linha da matriz por linha na saída), inclusive os elementos iguais a zero.

b) `void leMatriz()`
Esse método lê, de algum arquivo de entrada, os elementos diferentes de zero de uma matriz e monta a estrutura especificada anteriormente. Considere que a entrada consiste dos valores de *m* e *n* (número de linhas e de colunas da matriz) seguidos de triplas (*i, j, valor*) para os elementos diferentes de zero da matriz. Por exemplo, para a matriz anterior, a entrada seria:

```
4,      4
1,      1,      50.0
2,      1,      10.0
2,      3,      20.0
4,      1,      -30.0
4,      3,      -60.0
4,      4,      -5.0
```

c) `Matriz somaMatriz(Matriz A, Matriz B)`
Esse método recebe como parâmetros as matrizes *A* e *B*, devolvendo uma matriz *C* que é a soma de *A* com *B*.

d) `Matriz multiplicaMatriz(Matriz A, Matriz B)`
Esse método recebe como parâmetros as matrizes *A* e *B*, devolvendo uma matriz *C* que é o produto de *A* por *B*.

Para inserir e retirar células das listas que formam a matriz, crie métodos especiais para esse fim. Por exemplo, o método

```
void insere(int i, int j, double v)
```

para inserir o valor v na linha *i*, coluna *j* da matriz *A* será útil tanto na função `leMatriz` quando na função `somaMatriz`.

As matrizes a serem lidas para testar os métodos são:

a) A mesma da Figura 1 deste enunciado;

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

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

É obrigatório o uso de alocação dinâmica de memória para implementar as listas de adjacência que representam as matrizes.

As funções deverão ser testadas utilizando-se um programa similar à listagem abaixo:

```
public class TestaMatrizesEsparsas {
   public static void main(String[] args) {
      …
      A.leMAtriz();
      A.imprimeMatriz();
      B.leMatriz();
      B.imprimeMatriz();
      C = somaMatriz(A,B);
      C.imprimeMatriz();
      B.leMatriz();
      A.imprimeMatriz();
      B.imprimeMatriz();
      C = somaMatriz(A,B);
      C.imprimeMatriz();
      C = multiplicaMatriz(A,B);
      C.imprimeMatriz();
      C = multiplicaMatriz(B,B);
      A.imprimeMatriz();
      B.imprimeMatriz();
      C.imprimeMatriz();
      …
}
```

Envie todos os arquivos fonte e resultados dos testes executados em um pacote .zip.