

Exercício-Programa 2

Adição em Matrizes Esparsas

1º Semestre 2003

1 Introdução

Matrizes esparsas contêm grande número de elementos nulos, bem como grande número de linhas e colunas.

Para evitar o desperdício de espaço causado pela alocação de tantos elementos nulos, uma solução é representar a matriz com alocações ligadas, onde apenas elementos não-nulos são inseridos em uma matriz com essa representação.

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

Figura 1: Exemplo de matriz esparsa.

O objetivo deste exercício-programa é construir uma representação de matriz esparsa, bem como implementar a operação de soma entre matrizes esparsas.

2 Classes e descrições

Arquivos componentes:

- Celula2D.java
- Lista2D.java
- Operacao.java
- LeituraEscrita.java
- Soma.java
- Matrizes.java

2.1 Classe Celula2D

A `Celula2D` é o elemento fundamental da `Lista2D`, e cada uma encapsula uma informação do tipo `int`.

Além da informação, cada `Celula2D` contém referências inteiras sobre linha e coluna em que o elemento estaria originalmente localizado.

Cada `Celula2D` está ligada a uma célula anterior (que pode ser cabeça de linha ou coluna que são compostas por *arrays* de `Celula2D`) e a uma célula superior. Deste modo, há alocações circulares ligadas em linhas e colunas da matriz esparsa.

2.2 Classe Lista2D

Implementação da representação de matriz esparsa, composta por objetos `Celula2D` ligados.

2.3 Interface Operacao

Descreve uma interface para que operações entre matrizes sejam implementadas.

A classe `Soma` implementa esta interface. Desta forma, se for necessário adicionar mais operações entre matrizes esparsas, basta que a nova classe implemente esta interface.

2.4 Classe LeituraEscrita

Fornece rotinas de entrada e saída para o programa principal.

Carrega arquivos do disco, que devem estar no seguinte formato:

```
6
6
0 0 15
3 2 11
0 5 -1
1 1 42
```

A primeira linha indica o número de linhas da matriz, bem como a segunda indica o número de colunas. As outras linhas indicam linha e coluna a ser inserida a informação, dada pelo terceiro número.

Argumentos mal-passados (excesso de linhas em branco, excesso de espaços, tentativa de inserção em posições inválidas, etc.) gerarão erros que serão indicados pelo programa.

2.5 Classe Soma

Implementa a interface `Operacao` e realiza a operação entre duas matrizes esparsas dadas.

2.6 Classe Matrizes

Classe condutora da execução do exercício-programa.

Deve-se chamar esta classe para que a operação de soma entre duas matrizes seja realizada.

Sintaxe:

```
java Matrizes Soma matriz1.txt matriz2.txt saida.txt
```

3 Códigos-fonte

Estão apresentados a seguir os códigos-fonte das classes relacionadas.

3.1 Celula2D.java

```

1  /* Celula2D.java
2     Copyright (C) 2004 Rodolpho Iemini Atoji
3
4     This program is free software; you can redistribute it and/or modify
5     it under the terms of the GNU General Public License as published by
6     the Free Software Foundation; either version 2 of the License, or
7     (at your option) any later version.
8
9     This program is distributed in the hope that it will be useful,
10    but WITHOUT ANY WARRANTY; without even the implied warranty of
11    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
12    GNU General Public License for more details.
13
14    You should have received a copy of the GNU General Public License
15    along with this program; if not, write to the Free Software
16    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
17 */
18
19 // *****
20 // **   MAC323 - Estruturas de Dados                **
21 // **   IME-USP - Primeiro Semestre de 2004        **
22 // **   Turma 45 - Professor Siang Wun Song        **
23 // **                                               **
24 // **   Exercício-Programa 2 -- Adição em Matrizes Esparsas **
25 // **                                               **
26 // **   Arquivo: Celula2D.java                     **
27 // **                                               **
28 // **   Rodolpho Iemini Atoji           4894631    **
29 // **                                               **
30 // **   Data de entrega: 03/05/2004              **
31 // *****
32
33 /**
34  * A 'Celula2D' é o elemento fundamental da "lista ligada 2D".
35  * Cada 'Celula2D' possui referências vertical e horizontal para outras
36  * 'Celula2D', bem como dois parâmetros inteiros, além do dado contido,
37  * armazenado em um tipo 'int'.
38  *
39  * @author Rodolpho Iemini Atoji
40  * @version 0.1
41  */
42 public class Celula2D {

```

```
43
44  /**
45   * Conteúdos de uma célula 2D pertencente a uma lista 2D.
46   * informacao: inteiro que se deseje armazenar numa célula
47   * linha:      parâmetro de localização da célula
48   * coluna:    parâmetro de localização da célula
49   * anterior:  referência para célula anterior em uma mesma linha em uma
50   *            lista 2D.
51   * superior:  referência para célula anterior em uma mesma coluna em uma
52   *            lista 2D.
53   */
54 private int informacao;
55 private int linha, coluna;
56 private Celula2D anterior, superior;
57
58  /**
59   * Construtor de um só argumento, apontando para referência nula.
60   *
61   * @param info Inteiro para inicializar a célula.
62   */
63 public Celula2D(int info) {
64     this(info, null, null, 0, 0);
65 }
66
67  /**
68   * Construtor-padrão: cria objeto com informação e aponta para a
69   * próxima célula da lista ligada.
70   *
71   * @param info Número para inicializar a célula.
72   * @param ant Referência para célula anterior em uma lista 2D.
73   * @param sup Referência para célula superior em uma lista 2D.
74   * @param lin Parâmetro de identificação de linha em uma lista 2D.
75   * @param col Parâmetro de identificação de coluna em uma lista 2D.
76   */
77 public Celula2D(int info, Celula2D ant, Celula2D sup, int lin, int col) {
78     this.informacao = info;
79     this.anterior = ant;
80     this.superior = sup;
81     this.linha = lin;
82     this.coluna = col;
83 }
84
85  /**
86   * Devolve a informação principal do conteúdo da Celula.
87   *
88   * @return Informação principal.
89   */
```

```
90     public int devolveInfo() {
91         return informacao;
92     }
93
94     /**
95      * Devolve a informação principal do conteúdo da célula em formato
96      * de String.
97      *
98      * @return String com a informação.
99      */
100    public String devolveInfoString() {
101        return new String(new Integer(informacao).toString());
102    }
103
104    /**
105     * Devolve o parâmetro 'linha' de identificação da célula.
106     *
107     * @return Linha em que a célula está localizada.
108     */
109    public int devolveLinha() {
110        return linha;
111    }
112
113    /**
114     * Devolve o parâmetro 'coluna' de identificação da célula.
115     *
116     * @return Coluna em que a célula está localizada.
117     */
118    public int devolveColuna() {
119        return coluna;
120    }
121
122    /**
123     * Devolve a referência para a célula anterior.
124     *
125     * @return Referência para a célula anterior.
126     */
127    public Celula2D devolveAnterior() {
128        return anterior;
129    }
130
131    /**
132     * Devolve a referência para a célula superior.
133     *
134     * @return Referência para a célula superior.
135     */
136    public Celula2D devolveSuperior() {
```

```

137     return superior;
138 }
139
140 /**
141  * Muda a referência para a célula anterior.
142  *
143  * @param ant Referência para a célula anterior.
144  */
145 public void mudaAnterior(Celula2D ant) {
146     this.anterior = ant;
147 }
148
149 /**
150  * Muda a referência para a célula superior.
151  *
152  * @param sup Referência para a célula anterior.
153  */
154 public void mudaSuperior(Celula2D sup) {
155     this.superior = sup;
156 }
157
158 /**
159  * Muda informação da célula.
160  *
161  * @param info Informação a ser modificada.
162  */
163 void mudaInfo(int info) {
164     this.informacao = info;
165 }
166
167 } // class Celula2D

```

3.2 Lista2D.java

```

1  /* Lista2D.java
2  Copyright (C) 2004 Rodolpho Iemini Atoji
3
4  This program is free software; you can redistribute it and/or modify
5  it under the terms of the GNU General Public License as published by
6  the Free Software Foundation; either version 2 of the License, or
7  (at your option) any later version.
8
9  This program is distributed in the hope that it will be useful,
10 but WITHOUT ANY WARRANTY; without even the implied warranty of
11 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
12 GNU General Public License for more details.
13
14 You should have received a copy of the GNU General Public License

```

```

15     along with this program; if not, write to the Free Software
16     Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
17 */
18
19 // *****
20 // **  MAC323 - Estruturas de Dados                **
21 // **  IME-USP - Primeiro Semestre de 2004        **
22 // **  Turma 45 - Professor Siang Wun Song        **
23 // **                                              **
24 // **  Exercício-Programa 2 -- Adição em matrizes esparsas **
25 // **                                              **
26 // **  Arquivo:Lista2D.java                        **
27 // **                                              **
28 // **  Rodolpho Iemini Atoji          4894631      **
29 // **                                              **
30 // **  Data de entrega: 03/05/2004                **
31 // *****
32
33 /**
34  * Implementação de uma "lista ligada 2D" (que pode representar uma matriz)
35  * composta de objetos do tipo 'Celula2D'.
36  * Cada objeto do tipo 'Celula2D' pode guardar como informação apenas variáveis
37  * do tipo 'byte'.
38  *
39  * @author Rodolpho Iemini Atoji
40  * @version 0.1
41  * @see Celula2D
42  */
43 public class Lista2D {
44
45     /**
46      * Arrays de mapeamento da lista.
47      * Os objetos indicam início da lista/matriz, com dados
48      * inválidos.
49      */
50     private Celula2D[] linhas, colunas;
51
52     /**
53      * Inicializa os atributos de classe.
54      *
55      * @param lin Número de linhas da lista/matriz.
56      * @param col Número de colunas da lista/matriz.
57      */
58     public Lista2D(int lin, int col) {
59         int i;
60         linhas = new Celula2D[lin];
61         colunas = new Celula2D[col];

```

```
62
63     /*
64     * Inicializa as linhas da matriz, criando os objetos e em
65     * seguida definindo seus apontadores de 'anterior' para eles mesmos.
66     * Os parâmetros de localização dentro da matriz são definidos como
67     * números negativos para indicar início de linha.
68     */
69     for (i = 0; i < linhas.length; i++) {
70         linhas[i] = new Celula2D(0, null, null, -(i-1), -(i-1));
71         linhas[i].mudaAnterior(linhas[i]);
72     }
73
74     /*
75     * Inicializa as colunas da matriz, criando os objetos e em
76     * seguida definindo seus apontadores de 'superior' para eles mesmos.
77     * Os parâmetros de localização dentro da matriz são definidos como
78     * números negativos para indicar início de coluna.
79     */
80     for (i = 0; i < colunas.length; i++) {
81         colunas[i] = new Celula2D(0, null, null, -(i-1), -(i-1));
82         colunas[i].mudaSuperior(colunas[i]);
83     }
84 }
85
86 /**
87  * Devolve número de linhas da matriz.
88  *
89  * @return Número de linhas da matriz.
90  */
91 public int numLinhas() {
92     return linhas.length;
93 }
94
95 /**
96  * Devolve número de colunas da matriz.
97  *
98  * @return Número de colunas da matriz.
99  */
100 public int numColunas() {
101     return colunas.length;
102 }
103
104 /**
105  * Devolve referência para primeira célula da linha.
106  *
107  * @param i Índice da linha a ter a primeira célula devolvida.
108  * @return Referência para primeira célula da linha da lista ligada 2D.
```

```
109     */
110     public Celula2D devolveInicioLinha(int i) {
111         return linhas[i];
112     }
113
114     /**
115     * Devolve referência para primeira célula da coluna.
116     *
117     * @param j Índice da coluna a ter a primeira célula devolvida.
118     * @return Referência para primeira célula da coluna da lista ligada 2D.
119     */
120     public Celula2D devolveInicioColuna(int j) {
121         return colunas[j];
122     }
123
124     /**
125     * Insere uma célula em uma linha da lista.
126     *
127     * @param cel Célula a ser inserida.
128     */
129     public void insereLinha(Celula2D cel) {
130         if (cel.devolveInfo() == 0)
131             return;
132
133         try {
134             int i = cel.devolveLinha();
135             Celula2D atual = linhas[i].devolveAnterior();
136             Celula2D anterior;
137             int colEsquerda, colDireita;
138             boolean continuar = true;
139
140             /*
141             * Insere célula em uma linha, colocando-a na ordem das coordenadas
142             * de linha e coluna.
143             */
144             if (atual == linhas[i]) {
145                 linhas[i].mudaAnterior(cel);
146                 cel.mudaAnterior(linhas[i]);
147             } else {
148                 colDireita = atual.devolveColuna();
149                 colEsquerda = atual.devolveAnterior().devolveColuna();
150
151                 while (continuar) {
152
153                     /*
154                     * Caso o último elemento da linha esteja em uma coluna de
155                     * posição anterior à posição que deve ser inserido o
```

```
156         * elemento, basta mudar a referência de anterior da
157         * cabeça de lista para o novo elemento inserido.
158         */
159     if (colDireita < cel.devolveColuna()) {
160         linhas[i].mudaAnterior(cel);
161         cel.mudaAnterior(atual);
162         continuar = false;
163     }
164
165     /*
166     * Se a coluna do elemento que deve ser inserido estiver
167     * entre dois elementos, inserir o mesmo entre os dois.
168     */
169     else if (colDireita > cel.devolveColuna() &&
170             colEsquerda < cel.devolveColuna()) {
171         anterior = atual.devolveAnterior();
172         atual.mudaAnterior(cel);
173         cel.mudaAnterior(anterior);
174         continuar = false;
175     }
176
177     /*
178     * Caso contrário, verificar um novo par de elementos
179     * da linha.
180     */
181     else {
182         atual = atual.devolveAnterior();
183         if (atual == linhas[i])
184             continuar = false;
185         colDireita = atual.devolveColuna();
186         colEsquerda = atual.devolveAnterior().devolveColuna();
187     }
188     } // while (continuar)
189 } // else
190
191 /*
192 * Chamada para fazer a ligação com os elementos da mesma coluna.
193 */
194 insereColuna(cel);
195
196 } catch (ArrayIndexOutOfBoundsException erro) {
197     System.out.println("Tentativa de inserção além dos limites da " +
198                       "matriz. Execução terminada.");
199     System.exit(1);
200 }
201 } // insereLinha
202
```

```
203     /**
204     * Insere uma célula em uma coluna da lista.
205     *
206     * @param cel Célula a ser inserida na coluna.
207     */
208     public void insereColuna(Celula2D cel) {
209         try {
210             int i = cel.devolveColuna();
211             int linSuperior, linInferior;
212             Celula2D atual = colunas[i].devolveSuperior();
213             Celula2D anterior;
214             boolean continuar = true;
215
216             /*
217             * Insere célula em uma coluna, colocando-a na ordem das
218             * coordenadas de linha e coluna.
219             */
220             if (atual == colunas[i]) {
221                 colunas[i].mudaSuperior(cel);
222                 cel.mudaSuperior(colunas[i]);
223             } else {
224                 linInferior = atual.devolveLinha();
225                 linSuperior = atual.devolveSuperior().devolveLinha();
226
227                 while (continuar) {
228
229                     /*
230                     * Caso o último elemento da coluna esteja em uma linha de
231                     * posição anterior à posição que deve ser inserido o
232                     * elemento, basta mudar a referência de anterior da
233                     * cabeça de lista para o novo elemento inserido.
234                     */
235                     if (linInferior < cel.devolveLinha()) {
236                         colunas[i].mudaSuperior(cel);
237                         cel.mudaSuperior(atual);
238                         continuar = false;
239                     }
240
241                     /*
242                     * Se a linha do elemento que deve ser inserido estiver
243                     * entre dois elementos, inserir o mesmo entre os dois.
244                     */
245                     else if (linInferior > cel.devolveLinha() &&
246                             linSuperior < cel.devolveLinha()) {
247                         anterior = atual.devolveSuperior();
248                         atual.mudaSuperior(cel);
249                         cel.mudaSuperior(anterior);
```

```
250         continuar = false;
251     }
252
253     /*
254     * Caso contrário, verificar um novo par de elementos
255     * da coluna.
256     */
257     else {
258         atual = atual.devolveSuperior();
259         if (atual == colunas[i])
260             continuar = false;
261         linInferior = atual.devolveColuna();
262         linSuperior = atual.devolveAnterior().devolveColuna();
263         System.out.println("método3");
264     }
265     } // while (continuar)
266 } // else
267
268 } catch (ArrayIndexOutOfBoundsException erro) {
269     System.out.println("Tentativa de inserção além dos limites da " +
270         "matriz. Execução terminada.");
271     System.exit(1);
272 }
273 } // insereColuna
274
275 /**
276  * Procura pela existência de um elemento na i-ésima linha e j-ésima
277  * coluna.
278  *
279  * @param i I-ésima linha a ser pesquisada.
280  * @param j J-ésima linha a ser pesquisada.
281  * @return Elemento da posição (i,j). Caso não encontrado, devolve 'null'.
282  */
283 public Celula2D procuraColuna(int i, int j) {
284     Celula2D pesquisa = null;
285
286     try {
287         pesquisa = colunas[j].devolveSuperior();
288
289         while (pesquisa.devolveLinha() != i && pesquisa != colunas[j])
290             pesquisa = pesquisa.devolveSuperior();
291
292         if (pesquisa == colunas[j])
293             return null;
294     } catch (ArrayIndexOutOfBoundsException erro) {
295         System.out.println("Tentativa de busca além dos limites da " +
296             "matriz. Execução terminada.");
```

```
297         System.exit(1);
298     }
299
300     return pesquisa;
301 }
302
303 /**
304  * Imprime a representação de uma matriz.
305  *
306  * @param matriz Matriz no formato pronto para ser impresso.
307  */
308 public void imprime(String[] matriz) {
309     for (int i = 0; i < matriz.length; i++)
310         System.out.println(matriz[i]);
311 }
312
313 /**
314  * Gera uma representação matricial a partir de uma Lista2D.
315  *
316  * @return Representação da matriz.
317  */
318 public String[] listaString() {
319     Celula2D atual;
320     String linhaAtual = "";
321     String[] representacao = new String[linhas.length];
322     String temp;
323
324     for (int i = 0; i < linhas.length; i++) {
325         atual = linhas[i].devolveAnterior();
326
327         for (int j = colunas.length - 1; j >= 0; j--) {
328             if (atual.devolveColuna() == j) {
329                 linhaAtual =
330                     atual.devolveInfoString().
331                     concat(" ").concat(linhaAtual);
332                 atual = atual.devolveAnterior();
333             }
334             else
335                 linhaAtual = "0 ".concat(linhaAtual);
336         }
337
338         representacao[i] = linhaAtual;
339         linhaAtual = "";
340     }
341
342     return representacao;
343 }
```

```

344
345 } // class Lista2D

```

3.3 Operacao2D.java

```

1  /* Operacao.java
2     Copyright (C) 2004 Rodolpho Iemini Atoji
3
4     This program is free software; you can redistribute it and/or modify
5     it under the terms of the GNU General Public License as published by
6     the Free Software Foundation; either version 2 of the License, or
7     (at your option) any later version.
8
9     This program is distributed in the hope that it will be useful,
10    but WITHOUT ANY WARRANTY; without even the implied warranty of
11    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
12    GNU General Public License for more details.
13
14    You should have received a copy of the GNU General Public License
15    along with this program; if not, write to the Free Software
16    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
17 */
18
19 // *****
20 // **   MAC323 - Estruturas de Dados                **
21 // **   IME-USP - Primeiro Semestre de 2004        **
22 // **   Turma 45 - Professor Siang Wun Song        **
23 // **                                               **
24 // **   Exercício-Programa 2 -- Adição em matrizes esparsas **
25 // **                                               **
26 // **   Arquivo:Operacao.java                      **
27 // **                                               **
28 // **   Rodolpho Iemini Atoji           4894631    **
29 // **                                               **
30 // **   Data de entrega: 03/05/2004              **
31 // *****
32
33 /**
34  * Define operações a serem feitas entre matrizes.
35  *
36  * @author Rodolpho Iemini Atoji
37  */
38 public interface Operacao {
39
40     /**
41     * Definição da operação a ser realizada.
42     * Obs.: operação binária.
43     *

```

```

44     * @param matriz1 Uma das matrizes a ser utilizada na operação.
45     * @param matriz2 A outra matriz a ser utilizada.
46     * @return Matriz resultado.
47     */
48     Lista2D operacao(Lista2D matriz1, Lista2D matriz2);
49
50 } // interface Operacao

```

3.4 LeituraEscrita.java

```

1  /* LeituraEscrita.java
2     Copyright (C) 2004 Rodolpho Iemini Atoji
3
4     This program is free software; you can redistribute it and/or modify
5     it under the terms of the GNU General Public License as published by
6     the Free Software Foundation; either version 2 of the License, or
7     (at your option) any later version.
8
9     This program is distributed in the hope that it will be useful,
10    but WITHOUT ANY WARRANTY; without even the implied warranty of
11    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
12    GNU General Public License for more details.
13
14    You should have received a copy of the GNU General Public License
15    along with this program; if not, write to the Free Software
16    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
17 */
18
19 // *****
20 // **   MAC323 - Estruturas de Dados                **
21 // **   IME-USP - Primeiro Semestre de 2004        **
22 // **   Turma 45 - Professor Siang Wun Song        **
23 // **                                               **
24 // **   Exercício-Programa 2 -- Adição em matrizes esparsas **
25 // **                                               **
26 // **   Arquivo:LeituraEscrita.java                **
27 // **                                               **
28 // **   Rodolpho Iemini Atoji           4894631    **
29 // **                                               **
30 // **   Data de entrega: 03/05/2004                **
31 // *****
32
33 import java.io.*;
34
35 /**
36  * Fornece rotinas de acesso a arquivos com os dados necessários para
37  * a classe SomaMatrizes.
38  *

```

```
39  * @see SomaMatrizes
40  * @author Rodolpho Iemini Atoji
41  * @version 0.1
42  */
43  public class LeituraEscrita {
44
45      public static int tamanho = 0;
46
47      /**
48       * Carrega uma matriz guardada em um arquivo-texto.
49       *
50       * @param arquivo Nome do arquivo de entrada.
51       * @return Objeto 'Lista2D' com a representação da matriz de entrada.
52       */
53      public static Lista2D carregaLista(String arquivo) {
54          Lista2D matriz = null;
55
56          try {
57              BufferedReader leitor =
58                  new BufferedReader(new FileReader(arquivo));
59              int linhas = Integer.parseInt(leitor.readLine());
60              int colunas = Integer.parseInt(leitor.readLine());
61              String aux = leitor.readLine();
62              String[] linhaLida = null;
63              matriz = new Lista2D(linhas, colunas);
64              if (aux == null) {
65                  System.out.println("Matriz em " +arquivo +" sem conteúdo. " +
66                      "Execução interrompida.");
67                  System.exit(1);
68              }
69              else
70                  linhaLida = aux.split(" ");
71
72              while (aux != null) {
73                  matriz.insereLinha(new Celula2D
74                      (Integer.parseInt(linhaLida[2]),
75                       null,
76                       null,
77                       Integer.parseInt(linhaLida[0]),
78                       Integer.parseInt(linhaLida[1])));
79
80                  aux = leitor.readLine();
81                  if (aux != null)
82                      linhaLida = aux.split(" ");
83              }
84
85          } catch (IOException erro) {
```

```
86         System.out.println("Problemas com a leitura do arquivo " +
87                             arquivo +" Execução interrompida.");
88         System.exit(1);
89     } catch (ArrayIndexOutOfBoundsException erro) {
90         System.out.println("Matriz em " +arquivo +" sem conteúdo ou " +
91                             "com linhas em branco.\n" +
92                             "Execução interrompida.");
93         System.exit(1);
94     } catch (NumberFormatException erro) {
95         System.out.println("Problemas com o arquivo de entrada " +arquivo +
96                             "\nVerifique se não há números digitados de " +
97                             "forma errada, mais de um \nespaço em branco" +
98                             " entre as colunas, ou espaço em branco" +
99                             " entre linhas.");
100        System.exit(1);
101    }
102
103    return matriz;
104 }
105
106 /**
107  * Grava uma matriz em um arquivo-texto, em formato idêntico ao necessário
108  * de entrada.
109  *
110  * @param arquivo Nome do arquivo de saída.
111  * @param matriz Matriz a ser gravada.
112  */
113 public static void gravaLista(Lista2D matriz, String arquivo) {
114     try {
115         BufferedWriter gravador =
116             new BufferedWriter(new FileWriter(arquivo));
117         String[] linhas = matriz.listaString();
118
119         for (int i = 0; i < matriz.numLinhas(); i++) {
120             gravador.write(linhas[i]);
121             gravador.newLine();
122             gravador.flush();
123         }
124
125     } catch (IOException erro) {
126         System.out.println("Não foi possível gravar no arquivo " +
127                             arquivo +" Execução interrompida.");
128         System.exit(1);
129     }
130 }
131
132 } // class LeituraEscrita
```

3.5 Soma.java

```

1  /* Soma.java
2     Copyright (C) 2004 Rodolpho Iemini Atoji
3
4     This program is free software; you can redistribute it and/or modify
5     it under the terms of the GNU General Public License as published by
6     the Free Software Foundation; either version 2 of the License, or
7     (at your option) any later version.
8
9     This program is distributed in the hope that it will be useful,
10    but WITHOUT ANY WARRANTY; without even the implied warranty of
11    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
12    GNU General Public License for more details.
13
14    You should have received a copy of the GNU General Public License
15    along with this program; if not, write to the Free Software
16    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
17 */
18
19 // *****
20 // **  MAC323 - Estruturas de Dados                **
21 // **  IME-USP - Primeiro Semestre de 2004        **
22 // **  Turma 45 - Professor Siang Wun Song        **
23 // **                                              **
24 // **  Exercício-Programa 2 -- Adição em matrizes esparsas **
25 // **                                              **
26 // **  Arquivo:Soma.java                          **
27 // **                                              **
28 // **  Rodolpho Iemini Atoji          4894631      **
29 // **                                              **
30 // **  Data de entrega: 03/05/2004              **
31 // *****
32
33 /**
34  * Implementação da soma de matrizes esparsas.
35  * Utiliza representação de matrizes esparsas da classe 'Lista2D'.
36  *
37  * @see Lista2D
38  * @author Rodolpho Iemini Atoji
39  * @version 0.1
40  */
41 public class Soma implements Operacao {
42
43     /**
44     * Soma duas matrizes contidas em dois arquivos-texto.
45     * A matriz soma é internamente armazenada na segunda matriz passada como
46     * argumento.

```

```
47      *
48      * Obs.: pela definição de soma de matrizes, ambas devem ter o mesmo
49      *       número de linhas e colunas. Caso esta condição seja violada,
50      *       o programa pára a execução.
51      *
52      * @param matriz1 Uma das matrizes a ser somada.
53      * @param matriz2 A outra matriz a ser somada.
54      * @return Matriz soma.
55      */
56 public Lista2D operacao(Lista2D matriz1, Lista2D matriz2) {
57     try {
58         if (matriz1.numLinhas() != matriz2.numLinhas() ||
59             matriz2.numColunas() != matriz2.numColunas()) {
60             System.out.println("Matrizes de dimensões diferentes. " +
61                                 "Execução terminada.");
62             System.exit(1);
63         }
64
65         Celula2D cel1, cel2;
66
67         for (int i = 0; i < matriz1.numLinhas(); i++) {
68             for (int j = 0; j < matriz1.numColunas(); j++) {
69                 cel1 = matriz1.procuraColuna(i, j);
70                 cel2 = matriz2.procuraColuna(i, j);
71
72                 if (cel1 == null && cel2 == null)
73                     continue;
74                 else if (cel1 != null && cel2 == null)
75                     matriz2.insereLinha(new Celula2D
76                                             (cel1.devolveInfo(),
77                                              null,
78                                              null,
79                                              cel1.devolveLinha(),
80                                              cel1.devolveColuna()));
81                 else if (cel1 != null && cel2 != null)
82                     cel2.mudaInfo(cel1.devolveInfo() + cel2.devolveInfo());
83                 else
84                     continue;
85             }
86         }
87     } catch (NullPointerException erro) {
88         System.exit(1);
89     }
90
91     return matriz2;
92 }
93
```

```
94 } // class Soma
```

3.6 Matrizes.java

```
1  /* Matrizes.java
2     Copyright (C) 2004 Rodolpho Iemini Atoji
3
4     This program is free software; you can redistribute it and/or modify
5     it under the terms of the GNU General Public License as published by
6     the Free Software Foundation; either version 2 of the License, or
7     (at your option) any later version.
8
9     This program is distributed in the hope that it will be useful,
10    but WITHOUT ANY WARRANTY; without even the implied warranty of
11    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
12    GNU General Public License for more details.
13
14    You should have received a copy of the GNU General Public License
15    along with this program; if not, write to the Free Software
16    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
17 */
18
19 // *****
20 // **  MAC323 - Estruturas de Dados                **
21 // **  IME-USP - Primeiro Semestre de 2004        **
22 // **  Turma 45 - Professor Siang Wun Song        **
23 // **                                             **
24 // **  Exercício-Programa 2 -- Adição em matrizes esparsas **
25 // **                                             **
26 // **  Arquivo:Matrizes.java                      **
27 // **                                             **
28 // **  Rodolpho Iemini Atoji           4894631    **
29 // **                                             **
30 // **  Data de entrega: 03/05/2004              **
31 // *****
32
33 /**
34  * Coordena a execução de operações com matrizes esparsas.
35  *
36  * @author Rodolpho Iemini Atoji
37  * @version 0.1
38  */
39 public class Matrizes {
40
41     public static void main(String[] args) {
42         Operacao op = null;
43
44         if (args.length != 4) {
```

```

45         exibeAjuda();
46         System.exit(0);
47     }
48     else {
49         try {
50             op = (Operacao)Class.forName(args[0]).newInstance();
51         } catch (ClassNotFoundException classe) {
52             System.out.println("Operação não disponível.");
53         } catch (InstantiationException instancia) {
54             System.out.println("Erro ao instanciar classe.");
55         } catch (IllegalAccessException ilegal) {
56             System.out.println("Acesso ilegal.");
57         }
58
59         if (op == null)
60             System.exit(1);
61
62         LeituraEscrita.gravaLista(op.operacao
63                                 (LeituraEscrita.carregaLista(args[1]),
64                                 LeituraEscrita.carregaLista(args[2])),
65                                 args[3]);
66     }
67
68     System.out.println("Resultado salvo em " +args[3]);
69     System.exit(0);
70 }
71
72 /**
73  * Exibe ajuda para utilizar o programa.
74  */
75 public static void exibeAjuda() {
76     System.out.println("\nMAC323 - Estruturas de Dados\n" +
77                       "Operacoes com matrizes esparsas\n\n" +
78                       "Uso: java Matrizes <Operacao> matriz1.txt " +
79                       "matriz2.txt saida.txt\n");
80 }
81
82 } // class Matrizes

```

4 Testes

4.1 Teste 1

Duas entradas e sua respectiva saída:

6	6	1 0 0 0 0 0
6	6	0 3 0 0 0 0
0 0 0	0 0 1	0 0 5 0 0 0

1 1 1	1 1 2	0 0 0 7 0 0
2 2 2	2 2 3	0 0 0 0 9 0
3 3 3	3 3 4	0 0 0 0 0 8
4 4 4	4 4 5	
5 5 4	5 5 4	

4.2 Teste 2

Duas entradas e sua respectiva saída, agora com números negativos e matriz não-quadrada:

8	8	0 0 0 0 0 0 0 0 0 0
10	10	0 0 0 0 0 0 0 0 0 0
1 2 3	1 2 -3	0 0 0 0 0 0 0 0 0 5
3 4 5	3 4 -2	0 0 0 0 3 0 0 0 0 0
5 6 7	5 7 7	0 0 0 0 0 0 0 0 0 0
7 8 9	7 7 1	0 0 0 0 0 0 7 7 0 0
0 0 -1	0 0 1	0 0 0 0 0 0 0 0 0 0
2 9 -4	2 9 9	0 0 1 0 0 0 0 0 1 9 0
7 2 8	7 2 -7	

4.3 Teste 3

Entradas maiores e respectiva saída:

20
26
0 0 1
4 5 7
3 3 9
10 11 2
15 19 -1
19 25 44
14 22 7
12 11 5
5 1 9

20
26
0 0 3
4 5 -3
3 3 -4
10 11 2
15 19 6
19 25 44
14 22 2
12 11 3
5 1 -2
11 25 11
1 1 2
7 7 7
3 2 1
1 2 2
9 8 7
6 6 6

