Java: Matrizes

Estruturas indexadas de dados homogêneos (todas do mesmo tipo) , onde são utilizados dois ou mais índices para identificar univocamente um particular elemento, costumam ser chamadas matrizes. Uma vez devidamente identificado um elemento de uma estrutura indexada, ele pode ser manipulado da mesma forma como manipulamos valores associados a variáveis simples.

O número de índices necessários para discriminar elementos em uma estrutura indexada representa a dimensão de tal estrutura. Assim, uma estrutura que requer um índice para identificar um de seus elementos (o caso de vetores) é de dimensão um, uma de dois índices de dimensão dois e assim sucessivamente.

Com relação a uma estrutura na qual empregamos dois índices falamos em linhas e colunas e adota−se uma convenção na qual um dos índices representa a linha e o outro a coluna. Usualmente o primeiro índice representa a linha e o segundo a coluna. Mas isto é apenas uma convenção. Poderíamos perfeitamente fazer o contrário (o primeiro índice representa a coluna e o segundo a linha) contanto que utilizemos de forma consistente tal convenção em todo o código de uma aplicação.

Quando utilizamos uma estrutura de dimensão três, costumamos falar em planos, linhas e colunas. A convenção usualmente adotada é que o primeiro índice represente um plano na estrutura, o segundo uma linha em tal plano e o terceiro uma coluna em naquela linha. Estruturas indexadas de dimensões maiores podem ser utilizadas caso isto seja interessante do ponto de vista algorítmico.

Java: Soma de Matrizes

Escrever uma aplicação que calcule a soma de duas matrizes com valores inteiros de `m` linhas e `n` colunas.

Aplicação

```
 001   public class SomaMatrizes
002   {
003
004     public static void main(String argv[])
005     {
006       int m, n, i, j, m1[][], m2[][], m3[][];
007       m = IO.readInt();
008       n = IO.readInt();
009       m1 = new int[m][n];
010       m2 = new int[m][n];
011       m3 = new int[m][n];
012       for (i = 0; i < m; i = i + 1)
013         for (j = 0; j < n; j = j + 1) m1[i][j] = IO.readInt();
014       for (i = 0; i < m; i = i + 1)
015         for (j = 0; j < n; j = j + 1) m2[i][j] = IO.readInt();
016       IO.writeln();
017       for (i = 0; i < m; i = i + 1)
018       {
019         for (j = 0; j < n; j = j + 1)
020         {
021           m3[i][j] = m1[i][j] + m2[i][j];
022           IO.write(m3[i][j], 3);
023         }
024         IO.writeln();
025       }
026     }
027
028   }
```

Java: Produto de Matrizes

Escrever uma aplicação que calcule o produto de uma matriz com valores inteiros de `m` linhas e `n` colunas por outra matriz com valores inteiros de `n` linhas e `p` colunas.

Aplicação

```
001  public class ProdutoMatrizes
002  {
003
004    public static void main(String argv[])
005    {
006      int m, n, p, i, j, k, m1[][], m2[][], m3[][];
007      m = IO.readInt();
008      n = IO.readInt();
009      p = IO.readInt();
010      m1 = new int[m][n];
011      m2 = new int[n][p];
012      m3 = new int[m][p];
013      for (i = 0; i < m; i = i + 1)
014        for (j = 0; j < n; j = j + 1) m1[i][j] = IO.readInt();
015      for (i = 0; i < n; i = i + 1)
016        for (j = 0; j < p; j = j + 1) m2[i][j] = IO.readInt();
017      IO.writeln();
018      for (i = 0; i < m; i = i + 1)
019      {
020        for (j = 0; j < p; j = j + 1)
021        {
022          m3[i][j] = 0;
023          for (k = 0; k < n; k = k + 1)
024            m3[i][j] = m3[i][j] + m1[i][k] * m2[k][j];
025          IO.write(m3[i][j], 3);
026        }
027        IO.writeln();
028      }
029    }
030
031  }
```

Java: Matriz transposta

Escrever um método que calcule a transposta de uma dada matriz. Se `At` representa a transposta da matriz `A`, então `A[i,j]=At[j,i]` para `1<=i<=m` e `1<=j<=n`, onde `m` representa o número de linhas e `n` o número de colunas da matriz `A`.

Aplicação

```
001  public class Transposta
002  {
003
004    static void transposta(int a[][], int at[][], int m, int n)
005    {
006      for (int i = 0; i < m; i = i + 1)
007        for (int j = 0; j < n; j = j + 1) at[j][i] = a[i][j];
008    }
009
010    public static void main(String argv[])
011    {
012      int i, j, m, n, mat[][], matTransp[][];
013      m = IO.readInt();
014      n = IO.readInt();
015      mat = new int[m][n];
016      matTransp = new int[n][m];
017      for (i = 0; i < m; i = i + 1)
018        for (j = 0; j < n; j = j + 1) mat[i][j] = IO.readInt();
019      IO.writeln();
020      transposta(mat, matTransp, m, n);
021      for (i = 0; i < n; i = i + 1)
022      {
023        for (j = 0; j < m; j = j + 1) IO.write(matTransp[i][j], 3);
024        IO.writeln();
025      }
026    }
027
028  }
```

Java: Matriz simétrica

Escrever um método que verifica se uma matriz é simétrica. Uma matriz `a` é simétrica se `a[i,j] = a[j,i]` para todo `1<=i,j<=n`.

Aplicação

```
001  public class Simetrica
002  {
003
004    static boolean simetrica(int a[][], int n)
005    {
006      int i = 1, j;
007      boolean sim = true;
008      while (sim && i < n)
009      {
010        j = 0;
011        while (sim && j < i)
012        {
013          sim = a[i][j] == a[j][i];
014          j = j + 1;
015        }
016        i = i + 1;
017      }
018      return sim;
019    }
020
021    public static void main(String argv[])
022    {
023      int i, j, n, mat[][];
024      n = IO.readInt();
025      mat = new int[n][n];
026      for (i = 0; i < n; i = i + 1)
027        for (j = 0; j < n; j = j + 1) mat[i][j] = IO.readInt();
028      if (simetrica(mat, n)) IO.writeln("sim");
029      else IO.writeln("nao");
030    }
031
032  }
```

Java: Quadrado mágico

Uma matriz quadrada inteira é chamada de "quadrado mágico" se a soma dos elementos de cada linha, a soma dos elementos de cada coluna e a soma dos elementos das diagonais principal e secundária são todos iguais. Exemplo: A matriz abaixo representa um quadrado mágico:

$$\begin{bmatrix} 8 & 0 & 7 \\ 4 & 5 & 6 \\ 3 & 10 & 2 \end{bmatrix}$$

Escrever um método que verifica se uma matriz de n linhas e colunas representa um quadrado mágico.

Aplicação

```
001  public class QuadradoMagico
002  {
003
004    static boolean quadradoMagico(int a[][], int n)
005    {
006      int i, j, val = 0, v;
007      boolean qM = true;
008      for (j = 0; j < n; j++) val = val + a[1][j];
009      i = 1;
010      while (qM && i < n)
011      {
012        v = 0;
013        for (j = 0; j < n; j++) v = v + a[i][j];
014        i = i + 1;
015        qM = v == val;
016      }
017      j = 0;
018      while (qM && j < n)
019      {
020        v = 0;
021        for (i = 0; i < n; i++) v = v + a[i][j];
022        j = j + 1;
023        qM = v == val;
024      }
025      if (qM)
026      {
027        v = 0;
028        for (i = 0; i < n; i++) v = v + a[i][i];
029        qM = v == val;
030        if (qM)
031        {
032          v = 0;
```

```
033           for (i = 0; i < n; i++) v = v + a[i][n - i - 1];
034           qM = v == val;
035         }
036       }
037       return qM;
038     }
039
040     public static void main(String argv[])
041     {
042       int i, j, n, mat[][];
043       n = IO.readInt();
044       mat = new int[n][n];
045       for (i = 0; i < n; i++)
046         for (j = 0; j < n; j++) mat[i][j] = IO.readInt();
047       if (quadradoMagico(mat, n)) IO.writeln("sim");
048       else IO.writeln("nao");
049     }
050
051   }
```

Java: Oito rainhas

Escrever uma aplicação que posicione 8 rainhas em um tabuleiro de xadrez de tal forma que cada uma delas nao esteja ameçada por qualquer uma das outras.

Aplicação

```
001  public class OitoRainhas
002  {
003
004    static void apresenteSolucao(boolean tabuleiro[][])
005    {
006      boolean branca = true;
007      IO.writeln();
008      for (int i = 0; i < 8; i = i + 1)
009      {
010        for (int j = 0; j < 8; j = j + 1)
011        {
012          if (tabuleiro[i][j]) IO.write("  W");
013          else
014            if (branca) IO.write("  O");
015            else IO.write("  #");
016          branca = !branca;
017        }
018        branca = !branca;
019        IO.writeln(); IO.writeln();
020      }
021    }
022
023    static boolean naoAmeacada(boolean tabuleiro[][],
024      int linha, int coluna)
025    {
026      int i, j;
027      boolean posicaoLegal = true;
028      i = linha - 1;
029      while (i >= 0 && posicaoLegal)
030      {
031        posicaoLegal = !tabuleiro[i][coluna];
032        i = i - 1;
033      }
034      i = linha - 1;
035      j = coluna + 1;
036      while (i >= 0 && j < 8 && posicaoLegal)
037      {
038        posicaoLegal = !tabuleiro[i][j];
039        i = i - 1;
```

```
040         j = j + 1;
041       }
042       i = linha - 1;
043       j = coluna - 1;
044       while (i >= 0 && j >= 0 && posicaoLegal)
045       {
046         posicaoLegal = !tabuleiro[i][j];
047         i = i - 1;
048         j = j - 1;
049       }
050       return posicaoLegal;
051     }
052
053     static boolean coloqueRainha(boolean tabuleiro[][], int linha)
054     {
055       int coluna = 0;
056       boolean boaPosicao = false;
057       if (linha >= 8) return true;
058       else
059       {
060         while (coluna < 8 && !boaPosicao)
061         {
062           tabuleiro[linha][coluna] = true;
063           if (naoAmeacada(tabuleiro,linha,coluna))
064             boaPosicao = coloqueRainha(tabuleiro,linha + 1);
065           if (!boaPosicao)
066           {
067             tabuleiro[linha][coluna] = false;
068             coluna = coluna + 1;
069           }
070         }
071         return boaPosicao;
072       }
073     }
074
075     public static void main(String argv[])
076     {
077       boolean tabuleiro[][] = new boolean[8][8];
078       for (int i = 0; i < 8; i = i + 1)
079         for (int j = 0; j < 8; j = j + 1) tabuleiro[i][j] = false;
080       if (coloqueRainha(tabuleiro,0)) apresenteSolucao(tabuleiro);
081       else IO.writeln("Solucao nao encontrada");
082     }
083
084   }
```