Fundamentos de programação

aula 08: Manipulação de matrizes

1 Introdução

Agora que nós já temos alguma prática com a manipulação de listas, o próximo passo é trabalhar com matrizes.

Porque, afinal de contas, uma matriz é só uma lista de listas

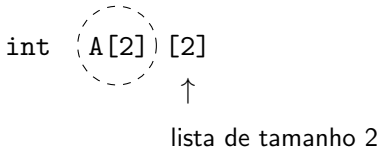
$$\mathsf{A} \quad = \quad \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Ou, pelo menos, é assim que elas são representadas na linguagem C.

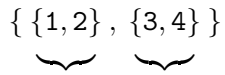
Por exemplo, se nós quisermos trabalhar com a matriz acima no nosso programa, nós fazemos a seguinte declaração

```
int   A[2][2]   = { {1,2} , {3,4} };
```

Quer dizer, na primeira parte nós estamos definindo uma lista de tamanho 2, onde os elementos são listas de tamanho 2



E na segunda parte nós estamos especificando as listas que fazem parte da nossa lista



Mas, felizmente, a gente não precisa se preocupar com nada disso na hora em que está programando.

Quer dizer, nós só precisamos usar os índices corretamente.

Vejamos um primeiro exemplo.

Imagine que nós queremos imprimir a matriz A na tela.

Para fazer isso, nós começamos imprimindo a primeira linha

```
for (j=0; j<2; j++)
   printf ("%d ", A[0][j]);
printf ("\n");
```

Para imprimir a segunda linha, basta modificar o `printf` para

```
printf ("%d ", A[1][j]);
```

E para imprimir as duas linhas, basta colocar o trecho de programa acima dentro de um outro

comando `for`

```
for (i=0; i<2; i++)
 {
    for (j=0; j<2; j++)
       printf ("%d ", A[i][j]);
    printf ("\n");
 }
```

Legal.

É bem fácil ver que essa lógica funciona com matrizes de qualquer tamanho.

O programa abaixo declara uma matriz com dimensões genéricas `M` × `N`, coloca alguns valores nessa matriz, e depois imprime a matriz na tela

```
#include <stdlib.h>
#include <stdio.h>

#define M 3
#define N 4

int A[M][N];
int i,j,k;

int main()
  {
    // ETAPA 1: inicializa a matriz
    k = 1;
    for (i=0; i<M; i++)
       for (j=0; j<N; j++)
        {
          A[i][j] = k;   k++;
        }

    // ETAPA 2: imprime a matriz na tela
    for (i=0; i<M; i++)
     {
       for (j=0; j<N; j++)
          printf ("%d ", A[i][j]);
       printf ("\n");
  }  }
```

Observação

- Esse programa produz o resultado

  ```
  1 2 3 4
  5 6 7 8
  9 10 11 12
  ```

  Isso está correto, mas não se parece com uma matriz.

  Mas, modificando a penúltima linha para: `printf ("%3d ", A[i][j])`, nós obtemos o resultado

  ```
  1   2   3   4
  5   6   7   8
  9  10  11  12
  ```

  Bem melhor, não é?

2 Operações sobre matrizes

Agora considere a operação de soma de matrizes

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

Como todo mundo sabe, para somar duas matrizes A e B, nós somamos elementos da mesma posição nas matrizes A e B para obter o elemento dessa posição na matriz resultado C

$$\begin{bmatrix} \textcircled{1} & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} \textcircled{5} & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1+5 & \\ & \end{bmatrix}$$

Isso pode ser feito por meio da instrução

```
C[i][j] = A[i][j] + B[i][j]
```

E agora, basta colocar essa instrução dentro de dois comandos `for`

```
for (i=0; i<2; i++)
   for (j=0; j<2; j++)
      C[i][j] = A[i][j] + B[i][j];
```

Para testar se a coisa funciona, nós podemos inicializar a matriz a apenas com 1's e a matriz B apenas com 2's, para ver se nós obtemos uma matriz resultado C que contém apenas 3's.

É isso o que o programa abaixo faz

```
#include <stdlib.h>
#include <stdio.h>

#define M 2
#define N 2

int A[M][N], B[M][N], C[M][N];
int i,j;

int main()
  {
    // ETAPA 1: inicializa as matrizes A e B
    for (i=0; i<M; i++)
       for (j=0; j<N; j++)
        {
          A[i][j] = 1;   B[i][j] = 2;
        }

    // ETAPA 2: realiza a soma de A e B
    for (i=0; i<M; i++)
       for (j=0; j<N; j++)
        {
          C[i][j] = A[i][j] + B[i][j];
        }
```

```
    // ETAPA 3: imprime a matriz C na tela
    for (i=0; i<M; i++)
     {
       for (j=0; j<N; j++)
          printf ("%d ", C[i][j]);
       printf ("\n");
  }  }
```

2.1 A operação de transposição

A operação de transposição consiste em trocar linhas e colunas de lugar

$$\mathsf{A} \quad = \quad \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad \Rightarrow \qquad \mathsf{A}^{\mathsf{T}} \quad = \quad \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

Quer dizer, as linhas de A viraram colunas em A^{T} (e vice-versa).

Se você pensar um pouquinho, não é difícil ver que a transposição corresponde a trocar os seguintes pares de elementos de lugar

$$\texttt{A[i][j]} \quad \longleftrightarrow \quad \texttt{A[j][i]}$$

Apesar disso, o seguinte programa não funciona

```
for (i=0; i<2; i++)
   for (j=0; j<2; j++)
    {
      aux = A[i][j];   A[i][j] = A[j][i];   A[j][i] = aux;
    }
```

Porque?

Bom, porque ele até faz o trabalho corretamente, mas depois ele desfaz.

Quer dizer, o programa troca os elementos

$$\texttt{A[i][j]} \qquad \text{e} \qquad \texttt{A[j][i]}$$

de lugar, mas depois ele destroca.

Para ver isso, note que os elementos

$$\texttt{A[0][1]} \qquad \text{e} \qquad \texttt{A[1][0]}$$

são trocados de lugar uma vez quando `i=0` e `j=1`, e depois são trocados outra vez quando `i=1` e `j=0`.

Daí que, duas trocas deixam as coisas no mesmo lugar que antes.

Mas, isso é fácil de resolver.

Nós podemos começar observando que os seguintes elementos não precisam ser trocados de lugar

$$\texttt{A[i][i]}$$

Quer dizer, quando `j=i` não é preciso fazer a troca.

E daí, para evitar as trocas duplicadas, basta trocar os elementos apenas quando `j < i`

```
for (i=0; i<2; i++)
   for (j=0; j<i; j++)
    {
      aux = A[i][j];   A[i][j] = A[j][i];   A[j][i] = aux;
    }
```

Certo.

Mas, essa maneira de pensar só funciona com matrizes quadradas.

Até porque, no caso retangular A e A^T são matrizes com dimensões diferentes

$$\mathsf{A} \quad = \quad \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \qquad \Rightarrow \qquad \mathsf{A}^\mathsf{T} \quad = \quad \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

Mas, esse caso também é fácil.

Basta copiar os elementos de A para a sua posição correta em A^T

```
AT[j][i]  ←  A[i][j]
```

O programa abaixo implementa essa ideia

```
#include <stdlib.h>
#include <stdio.h>

#define M 2
#define N 3

int A[M][N], AT[N][M];
int i,j,k;

int main()
  {
    // ETAPA 1: inicializa a matriz A
    k = 1;
    for (i=0; i<M; i++)
       for (j=0; j<N; j++)
        {
          A[i][j] = k;   k++;  }

    // ETAPA 2: cálculo da transposta
    for (i=0; i<M; i++)
       for (j=0; j<N; j++)
        {
          AT[j][i] = A[i][j];  }

    // ETAPA 3: imprime a matriz AT na tela
    for (i=0; i<N; i++)
     {
       for (j=0; j<M; j++)
          printf ("%d ", AT[i][j]);
       printf ("\n");
  }  }
```

3 Multiplicação de matrizes

A operação de multiplicação de matrizes é um pouquinho mais complicada.

Mas fazendo as coisas passo a passso, a gente chega lá.

Todo mundo aprende na escola que cada posição da matriz resultado C é obtida multiplicando uma linha de A por uma coluna de B

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1 \cdot 5 + 2 \cdot 7 & \\ & \end{bmatrix}$$

Nós podemos começar programando essa operação.

Quer dizer, para multiplicar a primeira linha de A pela primeira coluna de B basta fazer

```
C[0][0] = 0;
for (k=0; k<2; k++)
   C[0][0]  +=  A[0][k] * B[k][0]
```

onde nós lembramos que a instrução

```
C[0][0] += A[0][k] * B[k][0]
```

é só uma abreviação para

```
C[0][0] = C[0][0] + A[0][k] * B[k][0]
```

Certo.

Para obter o próximo elemento da primeira linha de C, basta fazer

```
C[0][1] = 0;
for (k=0; k<2; k++)
   C[0][1]  +=  A[0][k] * B[k][1]
```

Daí, observando o padrão repetitivo, nós escrevemos o seguinte trecho de programa que calcula toda a primeira linha da matriz resultado C

```
for (j=0; j<2; j++)
 {
   C[0][j] = 0;
   for (k=0; k<2; k++)
      C[0][j]  +=  A[0][k] * B[k][j]
 }
```

Finalmente, basta colocar tudo isso dentro de mais um comando `for` para calcular todas as linhas da matriz resultado

```
for (i=0; i<2; i++)
   for (j=0; j<2; j++)
    {
      C[i][j] = 0;
      for (k=0; k<2; k++)
         C[i][j]  +=  A[i][k] * B[k][j]
    }
```

Legal.

Agora, para testar se a coisa está funcionando direito, nós podemos escrever um programa que calcula uma multiplicação cujo resultado a gente já sabe qual é.

No caso, toda matriz A multiplicada pela matriz identidade

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

(i.e., a matriz onde a diagonal só tem 1's, e todo o resto é 0), sempre dá como resultado a própria matriz A.

O programa abaixo realiza essa multiplicação

```
#include <stdlib.h>
#include <stdio.h>

#define M 3

int A[M][M], I[M][M], C[M][M];
int i,j,k=1;

int main()
  {
    // ETAPA 1: inicializa as matrizes A e I
    for (i=0; i<M; i++)
       for (j=0; j<M; j++)
        {
          A[i][j] = k;   k++;
          if ( i == j )  I[i][j] = 1;
          else           I[i][j] = 0;
        }

    // ETAPA 2: cálculo da multiplicação
    for (i=0; i<M; i++)
       for (j=0; j<M; j++)
        {
          C[i][j] = 0;
          for (k=0; k<M; k++)
             C[i][j]  +=  A[i][k] * I[k][j]
        }

    // ETAPA 3: imprime a matriz AT na tela
    for (i=0; i<M; i++)
     {
       for (j=0; j<M; j++)
          printf ("%d ", C[i][j]);
       printf ("\n");
  }  }
```

4 Resolvendo sistemas lineares

Todo mundo também aprende na escola a resolver sistemas de equação do tipo

$$\begin{cases} 2x + y &= 2 \\ x + 2y &= 4 \end{cases}$$

Um método bem conhecido consiste em multiplicar uma das equações por um certo número, e depois somar as duas equações para fazer uma variável desaparecer

$$\begin{cases} 2x + y &= 2 \\ x + 2y &= 4 \end{cases} \qquad \begin{matrix} \Rightarrow \\ \times(-2) \end{matrix} \qquad \begin{array}{l} \begin{cases} 2x + y &= 2 \\ -2x - 4y &= -8 \end{cases} \\ + \\ \hline \qquad -3y = -6 \end{array}$$

Daí, nós descobrimos que $y = 2$.

Em seguida, nós substituímos esse valor na primeira equação

$$2x + 2 = 2$$

e descobrimos que $x = 0$.

Nessa seção, nós vamos programar esse tipo de raciocínio utilizando matrizes.

4.1 Raciocinando com matrizes

A primeira observação é que o sistema de equações acima corresponde à seguinte matriz

$$\mathsf{A} = \left[\begin{array}{cc:c} 2 & 1 & 2 \\ 1 & 2 & 4 \end{array}\right]$$

Daí, multiplicar a segunda equação por -2 é a mesma coisa que multiplicar a segunda linha por -2

$$\begin{bmatrix} 2 & 1 & 2 \\ 1 & 2 & 4 \end{bmatrix} \qquad \begin{matrix} \Rightarrow \\ (-2) \times \ell_2 \end{matrix} \qquad \begin{bmatrix} 2 & 1 & 2 \\ -2 & -4 & -8 \end{bmatrix}$$

E somar as duas equações é o mesmo que substituir a segunda linha pela soma da primeira com a segunda

$$\begin{bmatrix} 2 & 1 & 2 \\ -2 & -4 & -8 \end{bmatrix} \qquad \begin{matrix} \Rightarrow \\ \ell_2 \leftarrow \ell_1 + \ell_2 \end{matrix} \qquad \begin{bmatrix} 2 & 1 & 2 \\ 0 & -3 & -6 \end{bmatrix}$$

Nesse ponto, nós já podemos descobrir que $y = 2$, e para isso basta dividir a segunda linha por -2

$$\begin{bmatrix} 2 & 1 & 2 \\ 0 & -3 & -6 \end{bmatrix} \quad \underset{\frac{\ell_2}{-2}}{\Rightarrow} \quad \begin{bmatrix} 2 & 1 & 2 \\ 0 & 1 & 2 \end{bmatrix}$$

Certo.

Agora nós vamos trabalhar com a primeira linha da matriz.

E a ideia é aplicar um procedimento semelhante

- primeiro nós multiplicamos a primeira linha por -1

$$\begin{bmatrix} 2 & 1 & 2 \\ 0 & 1 & 2 \end{bmatrix} \quad \underset{(-1) \times \ell_1}{\Rightarrow} \quad \begin{bmatrix} -2 & -1 & -2 \\ 0 & 1 & 2 \end{bmatrix}$$

- daí nós substituimos a primeira linha pela soma da primeira com a segunda

$$\begin{bmatrix} -2 & -1 & -2 \\ 0 & 1 & 2 \end{bmatrix} \quad \underset{\ell_1 \leftarrow \ell_1 + \ell_2}{\Rightarrow} \quad \begin{bmatrix} -2 & 0 & 0 \\ 0 & 1 & 2 \end{bmatrix}$$

- finalmente, nós dividimos a primeira linha por -2

$$\begin{bmatrix} -2 & 0 & 0 \\ 0 & 1 & 2 \end{bmatrix} \quad \underset{\ell_1 \leftarrow \ell_1 + \ell_2}{\Rightarrow} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \end{bmatrix}$$

Pronto.

A matriz

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \end{bmatrix}$$

está basicamente dizendo que $x = 0$ e $y = 2$.

4.2 Descobrindo o padrão repetitivo

Legal.

Mas, vamos fazer mais um exemplo para descobrir o padrão repetitivo, e depois a gente escreve o programa.

Considere o sistema de equações

$$\begin{cases} 2x + y - z = 1 \\ x - y + 2z = 2 \\ x + 2y - z = 3 \end{cases}$$

Escrevendo a coisa como uma matriz, nós obtemos

$$\begin{bmatrix} 2 & 1 & -1 & 1 \\ 1 & -1 & 2 & 2 \\ 1 & 2 & -1 & 3 \end{bmatrix}$$

Talvez já esteja claro que a ideia do método é colocar a matriz no formato

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & ? \\ 0 & 1 & 0 & ? \\ 0 & 0 & 1 & ? \end{array}\right]$$

porque daí, nós já temos os valores das 3 variáveis.

Certo.

O primeiro passo para isso é fazer os 0's aparecerem.

Por exemplo, imagine que nós queremos fazer aparecer um 0 na primeira posição da segunda linha

$$\left[\begin{array}{cccc} 2 & 1 & -1 & 1 \\ \textcircled{1} & -1 & 2 & 2 \\ 1 & 2 & -1 & 3 \end{array}\right]$$

Pensando um pouquinho, não é difícil chegar na ideia de que isso pode ser feito da seguinte maneira

- multiplicar a segunda linha por -2
- substituir a segunda linha pela soma das duas primeiras

$$\left[\begin{array}{cccc} 2 & 1 & -1 & 1 \\ \textcircled{1} & -1 & 2 & 2 \\ 1 & 2 & -1 & 3 \end{array}\right] \quad \begin{array}{c} \Rightarrow \\ (-2) \times \ell_2 \\ \ell_2 \leftarrow \ell_1 + \ell_2 \end{array} \quad \left[\begin{array}{cccc} 2 & 1 & -1 & 1 \\ \textcircled{0} & 3 & -5 & -3 \\ 1 & 2 & -1 & 3 \end{array}\right]$$

Legal.

Uma operação semelhante coloca um 0 na primeira posição da terceira linha

$$\left[\begin{array}{cccc} 2 & 1 & -1 & 1 \\ 0 & 3 & -5 & -3 \\ 1 & 2 & -1 & 3 \end{array}\right] \quad \begin{array}{c} \Rightarrow \\ (-2) \times \ell_3 \\ \ell_3 \leftarrow \ell_1 + \ell_3 \end{array} \quad \left[\begin{array}{cccc} 2 & 1 & -1 & 1 \\ 0 & 3 & -5 & -3 \\ 0 & -3 & 1 & -5 \end{array}\right]$$

Finalmente, trabalhando com as linhas ℓ_2 e ℓ_3, nós fazemos aparecer um 0 na segunda posição de ℓ_3

$$\left[\begin{array}{cccc} 2 & 1 & -1 & 1 \\ 0 & 3 & -5 & -3 \\ 0 & -3 & 1 & -5 \end{array}\right] \quad \begin{array}{c} \Rightarrow \\ (1) \times \ell_3 \\ \ell_3 \leftarrow \ell_2 + \ell_3 \end{array} \quad \left[\begin{array}{cccc} 2 & 1 & -1 & 1 \\ 0 & 3 & -5 & -3 \\ 0 & 0 & -4 & -8 \end{array}\right]$$

Já está claro que nós estamos o tempo todo aplicando a mesma operação, não é?

E a ideia é continuar aplicando essa operação para fazer os 0's aparecerem na parte de cima da

matriz também.

Quer dizer, aplicando a sequência de operações

- $-(-4/-5) \times \ell_2$
  $\ell_2 \leftarrow \ell_3 + \ell_2$
- $-(-4/-1) \times \ell_1$
  $\ell_1 \leftarrow \ell_3 + \ell_1$
- $-(-\frac{12}{5}/-4) \times \ell_1$
  $\ell_1 \leftarrow \ell_2 + \ell_1$

nós chegamos na matriz

$$\begin{bmatrix} 24/5 & 0 & 0 & 36/5 \\ 0 & -12/5 & 0 & -26/5 \\ 0 & 0 & -4 & -8 \end{bmatrix}$$

E agora, basta

- multiplicar ℓ_1 por $5/24$
- multiplicar ℓ_2 por $-5/12$
- multiplicar ℓ_3 por $-1/4$

para obter

$$\begin{bmatrix} 1 & 0 & 0 & 36/24 \\ 0 & 1 & 0 & 26/12 \\ 0 & 0 & 1 & 2 \end{bmatrix}$$

E aqui nós já temos a solução do sistema

$$x = \frac{36}{24} \qquad y = \frac{26}{12} \qquad z = 2$$

4.3 Programando o método

Certo.

Já dá para ver que a operação que é repetida de novo e de novo, é a operação que faz aparecer um 0 em alguma posição de alguma linha.

Então, imagine que nós queremos fazer aparecer um 0 na posição p da linha ℓ_j.

Para fazer isso, nós usamos uma outra linha ℓ_i como ponto de apoio.

E a operação consiste nos seguintes 2 passos

- multiplicar ℓ_j por $-\frac{\ell_i[p]}{\ell_j[p]}$

  (onde $\ell_i[p]$ é o elemento da posição p na linha ℓ_i)

- substituir ℓ_j por $\ell_i + \ell_j$

O seguinte trecho de programa implementa esse procedimento

```
fator = - A[i][p] / A[j][p];
for (k=0; k<=M; k++)
   A[j][k]  =  A[i][k]  +  fator * A[j][k]
```

Certo.

Agora, se nós quisermos fazer aparecer 0's na primeira posição da segunda linha em diante (utilizando a primeira linha como ponto de apoio), basta colocar esse trecho de programa dentro de um comando `for`

```
for (j=1; j<M; j++)
 {
   fator = - A[0][0] / A[j][0];
   for (k=0; k<=M; k++)
      A[j][k]  =  A[i][k]  +  fator * A[j][k]
 }
```

Certo.

E agora, se nós quisermos fazer aparecer 0's em todas as posições abaixo da diagona, basta colocar esse trecho de programa dentro de um outro comando `for`

```
for (i=0; i<M-1; i++)                          <=  linha usada como pt de apoio
   for (j=i+1; j<M; j++)                       <=  linha que está sendo manipulada
    {
      fator = - A[i][i] / A[j][i];
      for (k=0; k<=M; k++)
         A[j][k]  =  A[i][k]  +  fator * A[j][k]
    }
```

Certo.

E para fazer os 0's aparecerem acima da diagonal, é só fazer basicamente a mesma coisa (ao contrário, ou de baixo para cima)

```
for (i=M-1; i<0; i++)                        <=  linha usada como pt de apoio
   for (j=i-1; j<=0; j++)                    <=  linha que está sendo manipulada
    {
      fator = - A[i][i] / A[j][i];
      for (k=0; k<=M; k++)
         A[j][k]  =  A[i][k]  +  fator * A[j][k]
    }
```

Abaixo nós temos o programa completo que implementa o método de solução de sistemas lineares.

```
#include <stdlib.h>
#include <stdio.h>

#define M 3

double  A[M][M+1]  =  { { 2, 1, -1, 1 },
                        { 1, -1, 2, 2 },
                        { 1, 2, -1, 3 }
                      };
int     i,j,k;
double  fator, aux;

int main()
  {

    // ETAPA 1: zerando abaixo da diagonal

    for (i=0; i<M-1; i++)
       for (j=i+1; j<M; j++)
        {
          fator = - A[i][i] / A[j][i];
          for (k=0; k<=M; k++)
             A[j][k]  =  A[i][k]  +  fator * A[j][k];
        }


    // ETAPA 2: zerando acima da diagonal

    for (i=M-1; i>0; i--)
       for (j=i-1; j>=0; j--)
        {
          fator = - A[i][i] / A[j][i];
          for (k=0; k<=M; k++)
             A[j][k]  =  A[i][k]  +  fator * A[j][k];
        }


    // ETAPA 3: colocando 1's na diagonal

    for (i=0; i<M; i++)
       {
          fator =   1 / A[i][i];
          A[i][i] = fator * A[i][i];
          A[i][M] = fator * A[i][M];
       }


    // ETAPA 4: imprimindo o resultado

    printf("\n\n");
    for (i=0; i<M; i++)
     {
       for (j=0; j<=M; j++)
          printf ("%5.2f ", A[i][j]);
       printf("\n");
     }
  }
```