Introdução à Computação

Rosane Minghim e Guilherme P. Telles

9 de Agosto de 2012

Capítulo 5

Vetores e Matrizes

Neste capítulo apresentamos tipos de dados que são composições de outros tipos. As principais características desses tipos de dados compostos são:

1. a composição é homogênea, isto é, cada uma das suas unidades é do mesmo tipo das outras.

2. os elementos da composição são indexáveis individual e diretamente, isto é, para ter acesso a um elemento não é necessário passar por todos os outros.

Tais composições são chamadas em português de vetores e matrizes. Vetores são composições uni-dimensionais e matrizes são composições multi-dimensionais.

O conceito de vetores é similar àquele conhecido em matemática, de coordenadas de elementos de um espaço vetorial qualquer representados por suas coordenadas naquele espaço. No entanto, o conceito é expandido para armazenamento consecutivo com indexação direta de qualquer conjunto de dados com todos os elementos de mesmo tipo.

O conceito de matrizes também é análogo àquele visto em matemática, de valores indexados multiplamente (dois, três, quatro índices) dependendo do espaço em que estão mergulhadas. No entanto, em computação o conceito é expandido para admitir o armazenamento de elementos de mesmo tipo com indexação múltipla (dois ou mais índices).

5.1 Vetores

Um **vetor** (*array* em inglês) pode ser visto como um conjunto de variáveis do mesmo tipo, onde cada variável do conjunto não tem um único nome

próprio. O conjunto tem um nome e através do nome do conjunto cada um dos seus elementos, ou variáveis, pode ser referenciado individualmente. Esta referência é feita através de índices numéricos. Na figura abaixo temos um exemplo desta idéia: um vetor chamado `V` composto por sete inteiros .

Cada um dos inteiros que compõem `V` pode ser referenciado usando o operador de indexação `[ ]` em comandos da forma `V[i]` onde `i` é uma constante inteira, variável inteira ou uma expressão que resulte em um valor inteiro. O índice de `V` pode variar entre 1 e 7 (outros valores de índice também são admitidos, como veremos abaixo). Assim, o terceiro elemento de `V` é `V[3]` e podemos armazenar um valor inteiro em `V[3]` ou usar o valor armazenado por esse elemento em qualquer lugar onde usaríamos uma variável inteira.

No pseudo-código um vetor genérico é declarado da seguinte forma:

```
nome: tipo[índice inicial..índice final]
```

Por exemplo, o vetor `V` descrito acima seria declarado da seguinte forma:

```
V: inteiro[1..7]
```

Um vetor de onze reais com índices variando de -5 até 5 pode ser definido como:

```
abscissas: real[-5..5]
```

Pode-se também definir tipos tendo por base vetores, como no exemplo abaixo:

Exemplo 5.1

```
tipo
  frase = caracter[0..80]

variável
  linha: frase


linha[1] ← ':'
```

Podemos usar os valores armazenados nas posições de um vetor da mesma forma que uma variável simples, isto é, cada elemento de um vetor é uma variável independente das demais. Veja o exemplo abaixo.

Exemplo 5.2

```
variável
  abscissas: real[-5..5]
  i,j:inteiro
  valor:real
  prox_abs:real

abscissas[4] ← 10
abscissas[-1] ← 3
abscissas[0] ← -2
abscissas[-2] ← 3

j ← -5
abscissas[j] ← 0

abscissas[1] ← abscissas[5] + abscissas[-5]

valor ← raiz(abscissas[-1],2)

prox_abs ← abscissas[trunca(abscissas[0])]
```

No caso do exemplo acima, o valor da variável `prox_abs` após a seqüência completa de comandos seria 3 (verifique).

Um vetor ocupa uma porção contínua na memória. Isso faz com que o acesso a cada um de seus elementos seja bastante eficiente. A posição do i-ésimo elemento do vetor é calculada somando-se i vezes o tamanho de cada elemento à posição de memória em que o vetor começa. Assim, o i-ésimo elemento é acessado diretamente, sem a necessidade de que os elementos anteriores sejam inspecionados. Além do desempenho satisfatório, o vetor torna a manipulação de conjuntos de dados do mesmo tipo mais simples porque temos um único nome para o conjunto.

Como exemplo da aplicação de vetores, vamos considerar o seguinte problema: ler n números inteiros do teclado, $1 \leq n \leq 50$, e calcular e imprimir o valor da média. Para resolver este problema não precisamos de um vetor, basta ler os valores e somá-los, como no exemplo abaixo.

Exemplo 5.3

```
Algoritmo média

  variável
    soma, n,i: inteiro
    aux:inteiro
    média:real

  leia(n)
  i ← 0
  enquanto (i<n) faça

    leia(aux)
    soma ← soma + aux
    i ← i + 1
  fim

  média ← soma/n
  escreva(soma,média)
fim
```

Pense o que aconteceria se quiséssemos, além de calcular a média, imprimir uma lista de todos os números que têm valor maior ou igual à média. No algoritmo acima, um dado, após utilizado para a soma, é substituído pelo próximo e, portanto, perdido. Não é possível recuperá-lo para a comparação com a média calculada. Para resolver este caso, pode-se usar um vetor que armazene os valores conforme eles são lidos. Uma vez armazenados, esses valores podem ser comparados com a média, construindo-se a solução ilustrada abaixo.

Exemplo 5.4

```
Algoritmo média
  tipo vetor50 = inteiro[1..50]
  variável
    soma, n: inteiro
    valor: vetor50
    i:inteiro
    média:real

  leia(n)

  soma ← 0
```

```
  para i de 1 até n passo 1 faça
    leia(valor[i])
    soma ← soma + valor[i]
  fim para

  média ← soma/n

  escreva(média)
  para i de 1 até n passo 1 faça
    se (valor[i] ≥ média) então
      escreva(valor[i])
    fim se
  fim para
fim
```

Compare essa solução com a solução abaixo (que foi abreviada), em que não usamos vetor e somos obrigados a fazer uma grande quantidade de testes para resolver o problema para uma quantidade de valores variável entre 1 e 50.

Exemplo 5.5

```
Algoritmo média

  variável
    soma, n: inteiro
    valor1, valor2, valor3, ..., valor50: inteiro
    média:real

  leia(n)

  soma ← 0

  se (n ≥ 1) então
    leia(valor1)
    soma = soma + valor1
  fim se

  se (n ≥ 2) então
    leia(valor2)
    soma ← soma + valor2
  fim se
```

```
  ...

  se (n ≥ ) então
    leia(valor50)
    soma ← soma + valor50
  fim se

  média ← soma/n

  escreva(média)

  se (n ≥ 1 e valor1 ≥ média) então
    escreva(valor1)
  fim se

  se (n ≥ 2 e valor2 ≥ média) então
    escreva(valor2)
  fim se

  ...

  se (n ≥ 50 e valor50 ≥ média) então
    escreva(valor50)
  fim se
fim.
```

Outra alternativa igualmente ruim para resolver o problema sem usar um vetor é ler a seqüência de números duas vezes, uma em que a média é calculada e outra em que os números acima da média são impressos. A construção deste algoritmo é um exercício para o leitor.

Note que, quando um vetor é declarado, o número de posições 'reservadas' para ele é fixo. No Exemplo 5.4, o número de posições do vetor é 50. No entanto, nem todas as posições do vetor são sempre ocupadas, isto é, quando o programa executa para $n < 50$, as posições a partir de $n+1$ ficam obsoletas, isto é, reservadas mas não ocupadas. Por outro lado, se o usuário digitar um valor para n superior a 50, existiria a tentativa do algoritmo de armazenar um dado nesta posição, que não é uma posição válida. Esse erro deve ser previsto pelo programador. É preciso portanto que o programador, no momento de usar um vetor, defina o seu tamanho máximo criteriosamente, para evitar tanto a reserva excessiva de espaço quanto o risco de reservar muito pouco

espaço e ultrapassar o espaço máximo do vetor.

Outra recomendação quanto ao uso de vetores é procurar evitar o seu uso quando o problema é razoavelmente resolvido sem a necessidade de empregá-lo. Este é o caso do exemplo Exemplo 5.3, que poderia ter sido resolvido com o uso de vetor, mas esse armazenamento é totalmente desnecessário de acordo com o enunciado daquele problema. Já o Exemplo 5.4, como vimos, possui alternativas muito ruins de solução sem o uso de vetor. Portanto seu uso aí é perfeitamente justificável.

As duas recomendações acima são destacadas a seguir.

Sugestão 8 *Procure estimar cuidadosamente o tamanho adequado de um vetor para evitar reserva excessiva ou interrupção do programa por reserva insuficiente de espaço.*

Sugestão 9 *No desenvolvimento de um algoritmo, verifique se existe uma lógica razoável que resolva o problema sem a necessidade de armazenamento em tipos compostos (vetores ou matrizes).*

Para o exemplo 5.4 cabe um comentário adicional. Foi estabelecido que o problema não pode ser solucionado de forma razoável sem o uso de vetores. Já que este é o caso, o algoritmo seria melhor organizado se a leitura do vetor fosse separada do cálculo da soma, uma vez que a leitura é de fato independente do cálculo da soma e poderia, em futuras modificações do algoritmo, ser isolada numa fase inicial do algoritmo, e realizada por um subprograma específico de leitura de vetores sem problema algum. A nova versão do algoritmo seria aquela apresentada abaixo:

Exemplo 5.6

```
Algoritmo média
  tipo vetor50 = inteiro[1..50]
  variável
    soma, n: inteiro
    valor: vetor50
    i:inteiro
    média:real

  {leitura da sequência de valores}
  leia(n)
  para i de 1 até n passo 1 faça
    leia(valor[i])
  fim para
```

```
  {cálculo da média aritmética da sequência de valores}
  soma ← 0
  para i de 1 até n passo 1 faça
    soma ← soma + valor[i]
  fim para
  média ← soma/n

  {impressão dos resultados }
  escreva(média)
  para i de 1 até n passo 1 faça
    se (valor[i] ≥ média) então
      escreva(valor[i])
    fim se
  fim para
fim
```

Uso de vetores em subprogramas

Uma função não pode definir que o tipo do seu retorno é um vetor (apenas tipos simples podem ser usados para isso). No entanto, vetores podem ser passados livremente como parâmetros de entrada, saída, e entrada e saída. Como exercício implemente o Exemplo 5.6 na forma de subprogramas para a leitura, a soma, e a impressão dos valores maiores que a soma.

As possíveis passagens de parâmetros do tipo vetor são ilustradas através de exercício e exemplo, apresentados a seguir.

Exercício Resolvido 7 *Cálculo de pontos em uma prova.*

Enunciado*: No Capítulo 4 foi exemplificado o algoritmo de um subprograma para o cálculo de pontos em uma prova de 10 questões prestada por funcionários de uma empresa (ver Exemplo 4.3).*

O subprograma, de nome `calcule_pontos`*, possui o seguinte cabeçalho:*

```
calcule_pontos (questão,resposta,pontos,acerto)
```

e atualiza o parâmetro pontos incrementando-o ou decrementando-o conforme a resposta à questão. Além disso, o parâmetro `acerto` *retorna verdadeiro se a questão está correta e falso caso contrário.*

Desenvolver um subprograma para, dado um vetor de 10 elementos contendo a resposta a cada uma das 10 questões da prova, calcular a soma de pontos de um funcionário. Além disso, o subprograma retorna um vetor de 10 posições contendo, em cada posição, o valor `verdadeiro` *se a questão está*

correta e o valor `falso` *se a questão está errada. Este subprograma chama-se* `calcule_resultado_prova`

Resolução 7 *Para começar a resolver o problema, é necessário definir os dois vetores, isto é, aquele que vai conter as 10 respostas fornecidas pelo funcionário e aquele que vai conter os seus acertos e erros. Para isso, podemos definir dois tipos, ilustrados abaixo:*

```
constante
  N_QUESTÕES = 10

tipo
  vetor_respostas = caracter[1..N_QUESTÕES]

  vetor_acertos = lógico[1..N_QUESTÕES]
```

Então, o algoritmo para o subprograma toma como parâmetro de entrada um vetor das respostas de um funcionário. Na primeira posição ele contém a resposta à questão 1, na segunda posição a resposta à questão 2, e assim por diante. Este vetor é do tipo `vetor_resposta`*.*

Ele devolve o vetor de acertos, do tipo `vetor_acertos`*, além do número de pontos total.*

Seu cabeçalho, então, seria do tipo:

```
calcule_resultado_prova(respostas,acertos,número_de_pontos)
```

Abaixo, uma possível solução para o algoritmo do subprograma.

```
Subprograma
calcule_resultado_prova(respostas,acertos,total_de_pontos)

e: respostas: vetor_resposta {um vetor contendo, em cada posição,
a resposta a uma questão da prova}

s: acertos: vetor_acertos {um vetor contendo, em casa posição,
falso se a resposta à questão associada for errada, e verdadeiro
caso ela seja correta}
   total_de_pontos:inteiro {o número de pontos conseguido na
prova}

{este subprograma utiliza o subprograma calcule_pontos}
```

```
variável
  i:inteiro

início
   total_de_pontos ← 0
   Para i de 1 até N_QUESTÕES passo 1 faça
       calcule_pontos(i,respostas[i],total_de_pontos,acertos[i])
   fim para
fim
```

Exercício Sugerido 19 *Percorrer o subprograma acima para 3 casos de teste.*

Exercício Sugerido 20 *Desenvolver um algoritmo principal que chame o subprograma acima para cada um dos funcionários que prestou a prova e em seguida imprima quais respostas estavam corretas e o total de pontos obtido. Este algoritmo deve também imprimir o número do funcionário que tirou o primeiro lugar.*

Como um exemplo adicional, abaixo é fornecido um subprograma para calcular a média aritmética de um conjunto de n valores.

Exemplo 5.7

```
tipo

    vetor_valores=real[1..100]

Subprograma média_aritmética(v,n):real

e:v:vetor_valores {vetor de valores dos quais se deseja tirar a
média}
  n: número de elementos do vetor

variável
  i:inteiro
  soma,média:real

início
  se n ≤ 0 então
```

```
    média ← 0,0
  senão
    soma ← 0,0
    para i de 1 até n faça
      soma ← soma + v[i]
    fim para
    média ← soma/n
  fim se
  retorne(média)
fim
```

Exercício Sugerido 21 *Desenvolver um algoritmo principal para testar o subprograma acima. Percorrê-lo para vários casos de teste.*

Inicialização de Vetores Constantes

O pseudo-código admite a definição de vetores constantes, isto é, vetores que possuem conteúdo fixo. Um vetor constante é definido da seguinte forma:

```
constante
  nome:tipo = {c1,c2,...c_max}

  ou

  nome:tipo_simples [i1..i2] = {c1,c2,...c_max}
```

onde `tipo` é a definição de um vetor, `c1,c2,...c_max` são constantes do tipo elementar armazenado no vetor, `tipo_simples` é um tipo elementar armazenado no vetor, e [i1..i2] é o intervalo de variação do índice do vetor. Alguns exemplos são oferecidos a seguir:

```
tipo

  vet = inteiro[1..6]

constante
  v: vet = {4,5,4,3,3,1}
  c: caracter[1..3] = {'A', 'B', 'C'}
```

Abaixo, é oferecido um exemplo que implementa uma nova versão do algoritmo do subprograma `calcule_pontos` apresentado no Exemplo 4.3, agora utilizando vetores constantes. Esta versão, embora ligeiramente mais obscura em termos de entendimento, é muito mais compacta e também mais fácil de ser alterada para futuras modificações da prova de aptidão.

Exemplo 5.8

```
Subprograma calcule_pontos (questão, resp, pontos, correta)

e: questão: inteiro {número da questão respondida}
   resp: caracter {resposta dada pelo candidato}

e/s: pontos {pontos obtidos anteriormente pelo candidato,
     atualizado pelo número de pontos conferido pela questão }

s: correta: lógico {retorna verdadeiro se a resposta
             à questão é correta}

constante
  gabarito:caracter[1..10] = {'C','A','B','A','B','C','D','D','C','A'}
  pts_certo:inteiro[1..10] = {15,15,15,15,10,10,10,10,5,5}
  pts_errado:inteiro[1..10] = {10,10,5,5,5,5,5,5,5,2}

início
  se resp = 'F' então
    correta ← falso
    pontos ← pontos - 1
  senão
    Se resp = gabarito[questão] então
        pontos ← pontos + pts_certo[questão]
        correta ← verdadeiro
    senão
        pontos ← pontos - pts_errado[questão]
        correta ← falso
    fim se
  fim se
fim
```

Conforme mencionado nas seções acima, vetores são indexados por um único índice. Diz-se com isso que é um conjunto de dados uni-dimensional.

Linguagens de programação normalmente conseguem expressar conjuntos de dados homogêneos de até três dimensões como parte do próprio elenco de tipos de dados. A forma de expresar esses conjuntos multi-dimensionais em pseudo-código é apresentada na próxima Seção.

5.2 Matrizes

Chamamos de matriz a extensão natural de um vetor para duas ou mais dimensões. A sintaxe da definição e a indexação das matrizes é similar à dos vetores, com o acréscimo de mais dimensões, de acordo com a forma geral abaixo:

```
nome: tipo[índice inicial..índice final]...[índice inicial..índice final]
```

No exemplo abaixo definimos uma matriz (de 2 dimensões) capaz de armazenar as notas de, no máximo, 50 alunos nas disciplinas numeradas de 1 até 5. Cada linha da matriz armazena as notas de um aluno. Cada coluna é uma nota. No mesmo exemplo, atribuímos valores às notas do aluno 1 e do aluno 2.

Exemplo 5.9

```
variável
  notas: real[1..50][1..5]

notas[1][1] ← 6,0
notas[1][2] ← 9,0
notas[1][3] ← 4,9
notas[1][4] ← 7,2
notas[1][5] ← 8,2
notas[2][1] ← 5,0
notas[2][2] ← 9,7
notas[2][3] ← 2,9
notas[2][4] ← 7,2
notas[2][5] ← 10,0
```

Note que uma matriz é declarada com um certo número de dimensões, e uma faixa fixa de valores para o índice de cada uma delas. Isso é obrigatório no momento da declaração. No exemplo acima, a matriz declarada possui 2 dimensões, a primeira variando de 1 até 50 e a segunda de 1 até 5. Para uma matriz bi-dimensional, a primeira dimensão é denominada pelo termo linha e

a segunda pelo termo coluna, como é usual em aplicações matemáticas. Para a matriz definida acima, no momento da declaração são reservadas posições de memória suficientes para armazenar 5×50 números reais para seus dados. Embora a reserva de espaço seja determinada por esses valores declarados, a matriz propriamente pode ser usada com um número menor de elementos. Por exemplo, a matriz do Exemplo 5.9 pode ser usada para um número de estudantes que pode chegar até 50, mas pode ser menor. O mesmo ocorre para o número de notas para cada estudante.

No exemplo abaixo definimos uma matriz de três dimensões[1], que guarda pontos no espaço tri-dimensional definido por coordenadas discretas entre -10 e +10 para os eixos x, y e z. Cada ponto pode ser verdadeiro ou falso, indicando se ele faz ou não parte de um objeto que queremos representar em três dimensões.

Exemplo 5.10

```
variável
  objeto: lógico[-10..10][-10..10][-10..10]

objeto[0][0][0] ← falso
objeto[0][1][1] ← verdadeiro
objeto[1][-5][1] ← falso
```

É sempre interessante que, antes da declaração, sejam definidos tipos para matrizes que serão utilizadas em programas. algumas linguagens de programação tipadas exigem que matrizes passadas como parâmetro em procedimentos tenham um tipo pré-definido. O hábito de definir tipos previamente é saudável em termos de organização e facilidade de executar mofícações futuras. No texto que segue são apresentados exemplos de definição de tipos baseados em matrizes, e do uso de matrizes em subprogramas.

Uso de matrizes em subprogramas

Da mesma forma que vetores, uma função não pode definir que o tipo do seu retorno é uma matriz, mas elas podem ser passadas livremente como parâmetros de entrada, de saída, e de entrada e saída em subprogramas.

É importante para o uso de matrizes em subprogramas que o seu tipo seja definido previamente, por uma questão de consistência, compatibilidade de tipos e organização do código.

[1] Apesar de estranha, essa forma de nos referirmos a composições de dimensões maiores é utilizada comumente em computação. Em inglês usa-se a expressão *multidimensional array*

Em pseudo-código, tipos podem ser definidos para matrizes da mesma forma que para vetores, conforme exemplo abaixo:

Exemplo 5.11 *Declaração de tipos baseados em matrizes*

```
mat_notas = real[1..50][1..5]

objeto_logico = lógico[-10..10][-10..10][-10..10]
```

Desta forma, subprogramas e algoritmos principais podem trabalhar sobre matrizes de mesmo tipo pré-definido.

Utilizando as definições acima, as declarações dos exemplos 5.9 e 5.10 ficariam:

```
variável
   nota: mat_notas
   objeto: objeto_logico
```

O exemplo a seguir ilustra o uso de matrizes em subprogramas.

Exemplo 5.12 *Média dos elementos das linhas de uma matriz*

Suponha que, dada uma matriz de números reais, deseje-se obter a média aritmética dos elementos de cada linha da matriz, armazenando o resultado num vetor, conforme ilustrado abaixo.

Dada a matriz:

$$\begin{pmatrix} 2,0 & 3,2 & 1,1 & 0,0 \\ 5,0 & 1,3 & 7,0 & 3,0 \\ 4,4 & 0,0 & 9,0 & 8,2 \end{pmatrix}$$

O resultado seria:

$$\begin{pmatrix} (2,0+3,2+1,1+0,0)/4 \\ (5,0+1,3+7+3)/4 \\ (4,4+0,0+9,0+8,2)/4 \end{pmatrix} = \begin{pmatrix} 1,575 \\ 4,075 \\ 2,05 \end{pmatrix}$$

Um subprograma para produzir os resultados acima possui uma matriz de valores reais como parâmetro de entrada e um vetor também de valores reais como parâmetro de saída.

Uma possível definição de tipos para este problema seria:

```
constante
   MAX = 20
tipo
   matriz_real = real[1..MAX][1..MAX]
   vetor_real = real[1..MAX]
```

Além da própria matriz, a faixa de variação das dimensões precisa ser passada como parâmetro de entrada. Ou seja, é preciso informar, na lista de argumentos do subprograma, qual o número de linhas e de colunas da matriz, já que uma matriz pode ter qualquer número de linhas e de colunas até o tamanho máximo especificado na declaração ou definição. No caso da indexação iniciar em um valor qualquer (diferente de 1), também seria preciso passar o valor inicial da indexação e ajustar o algoritmo interno ao subprograma de acordo com esse índice.

O algoritmo para resolver o problema acima calcula, para cada linha da matriz, a soma de todos os seus elementos, dividida pelo número de colunas da matriz. O subprograma abaixo apresenta uma solução para o problema.

```
Subprograma calcule_médias (mat, linhas,colunas, med)

e: linhas, colunas: inteiro {dimensões da matriz, linhas também é
                             a dimensão do vetor}
   mat: matriz_real

s: med: vetor_real {armazena medias aritméticas das linhas da
                    matriz}

{pré-condição: colunas > 0}

variável
   lin, col: inteiro
   soma: real

início
   para lin de 1 até linhas faça
     soma ← 0
     para col de 1 até colunas faça
          soma ← soma + mat[lin][col]
     fim para
     med[lin] ← soma / colunas
```

```
  fim para
fim
```

O trecho de algoritmo a seguir chama o subprograma calcule_médias (observe a chamada):

```
variável
   matriz_valores: matriz_real
   vetor_médias: vetor_real
   n,m: inteiro

    ...


    {leitura ou cálculo da matriz}

    ...

    {cálculo da média das colunas }

    calcule_médias(matriz_valores,n,m,vetor_médias)

    {impressão do resultado }

    ...

fim
```

Note que muitos algoritmos de matrizes e vetores podem ser expressos de forma compacta através de expressões matemáticas.

Por exemplo o enunciado do algoritmo do exemplo 5.12, poderia expressar o vetor médias da seguinte forma:

$$med_i = \sum_{j=1}^{colunas} mat_{i,j}, 1 <= i <= linhas$$

A seguir é resolvido um exercício empregando matriz.

Exercício Resolvido 8 *Extração da linha e coluna do maior elemento*

Desenvolver um subprograma para extrair a linha e a coluna completas do maior elemento de uma matriz de valores reais.

Ou seja, dados:
$i_1, j_1 | m_{i_1,j_1} > m_{i,j} \forall 1 <= i <= n_{linhas}, i \neq i_1, \forall 1 <= j <= n_{colunas}, j \neq j_1$
Obter:
$l_k = m_{i_1,k}, 1 <= k <= n_{colunas}$
$c_k = m_{k,j_1}, 1 <= k <= n_{linhas}$

Resolução 8 *Para copiar a linha e a coluna do maior elemento da matriz, primeiramente é preciso encontrá-lo, registrando o índice de sua linha e de sua coluna. Em seguida, um laço deve copiar a linha para o vetor* `'linha'` *e a coluna para o vetor* `'coluna'`*. Esta abordagem para solução pode ser vista no algoritmo abaixo.*

```
Subprograma copie_dados(m,nlinhas,ncolunas,vet_lin,vet_col)

e: m: matriz_real
   nlinhas,ncolunas: inteiro

s: vet_lin: vetor_real
   vet_col: vetor_real

variável
  i,j: inteiro
  max: real
  max_lin,max_col: inteiro
begin
  max ← m[1][1]
  max_lin ← 1
  max_col ← 1
  para i de 1 até nlinhas faça
     para j de 1 até ncolunas faça
         Se max < m[i][j] então
            max ← m[i][j]
            max_lin ← i
            max_col ← j
         fim se
     fim para
  fim para
  para i de 1 até nlinhas faça
     v[i] ← m[i][max_col]
  fim para
  para j de 1 até ncolunas faça
```

```
    v[j] ← m[max_lin][j]
  fim para
end
```

Pergunta 1 *Se, na matriz submetida ao algoritmo acima, existirem dois valores 'maiores', isto é, o maior valor da matriz é replicado em mais de uma posição, qual elemento da matriz é escolhido? Resposta no final do exercício*

Para testar o subprograma acima seria interessante que o algoritmo principal utilizasse subprogramas específicos para leitura e escrita de matrizes. Um programa para leitura de uma matriz digitada pelo usuário pode ler seus alementos a partir do seu número de colunas e do seu número de linhas. Um programa para impressão da matriz toma como entradas a própria matriz e suas dimensões, e imprime os seus elementos. A seguir é apresentado um algoritmo principal para teste do subprograma `copie_dados` *assumindo a leitura e escrita de matrizes através de subprogramas. Ele também utiliza um subprograma para escrita de elementos de um vetor do tipo vet, e executa repetidamente até que o usuário responda que quer parar o processo (através do carecter lido na variável resposta_usuário).*

```
Algoritmo testa_matrizes

constante
   MAX = 20
tipo
   matriz_real = real[1..MAX][1..MAX]
   vetor_real = real[1..MAX]

variável
   nlin, ncol: inteiro
   m: matriz_real
   vet_l,vet_c: vetor_real
   resposta_usuário: caractere

   repita
      leia(nlin,ncol)
      leia_matriz(m,nlin,ncol)
      copie_dados(m,nlin,ncol,vet_l,vet_c)
      escreva_matriz(m,nlin,ncol)
```

```
      escreva_vetor(vet_l,ncol)
      escreva_vetor(vet_c,nlin)
      leia (resposta_usuário)
   até que resposta_usuário = 'S' ou resposta_usuário = 's'
```

Um subprograma rudimentar (ainda carente de formatação e de mensagens adequadas) para leitura e outro para escrita de uma matriz são apresentados a seguir

```
Subprograma leia_matriz(mat, linhas,colunas)

e: linhas, colunas: inteiro dimensões da matriz

s: mat: matriz_real valores lidos

variável
   lin, col: inteiro
inicio
   para lin de 1 até linhas faça
     para col de 1 até colunas faça
        leia(mat[lin][col])
     fim para
  fim para
fim


Subprograma escreva_matriz(mat, linhas,colunas)

e:   linhas, colunas: inteiro dimensões da matriz
     mat: matriz_real notas

variável
   lin, col: inteiro
inicio
   para lin de 1 até linhas faça
     para col de 1 até colunas faça
        escreva(mat[lin][col])
     fim para
  fim para
fim
```

Resposta 1 *Se, na matriz submetida ao algoritmo* `copie_dados`*, o maior valor da matriz estiver replicado em mais de uma posição, a escolha recai sobre o primeiro deles a ser encontrado (menor número de linha e menor número de coluna).*

Exercício Sugerido 22 *Completar a resolução do exercício acima inserindo diálogos com o usuário e desenvolvendo o subprograma para escrita do vetor.*

Exercício Sugerido 23 *Refazer o subprograma copie_dados. Realizar a busca do maior elemento da matriz, com retorno da linha e coluna onde foi encontrado, por um outro subprograma, que é chamado pelo subprobrama copie_dados.*

Para concluir este capítulo, a seguir é desenvolvido o algoritmo para multiplicação de matrizes.

Exercício Resolvido 9 *Multiplicação de Matrizes*

Enunciado*: Dadas duas matrizes $M_{m\times l}$ e $O_{l\times n}$ de valores reais, desenvolver um subprograma para calcular a matriz produto $P_{m\times n} = M \times O$.*

Como exemplo, seja:

$$M = \begin{pmatrix} 2 & 3 & 1 \\ 5 & 1 & 7 \\ 4 & 0 & 9 \end{pmatrix} \quad e \quad O = \begin{pmatrix} 8 & 2 & 1 \\ 4 & 3 & 6 \\ 5 & 1 & 0 \end{pmatrix}$$

Para multiplicar uma matriz pela outra, os elementos de cada linha são multiplicados em sequência pelos elementos de cada coluna e somados. O resultado para as matrizes acima é dado por:

$$P = \begin{pmatrix} 2\times 8+3\times 4+1\times 5 & 2\times 2+3\times 3+1\times 1 & 2\times 1+3\times 6+1\times 0 \\ 5\times 8+1\times 4+7\times 5 & 5\times 2+1\times 3+7\times 1 & 5\times 2+1\times 6+7\times 0 \\ 4\times 8+0\times 4+9\times 5 & 4\times 2+0\times 3+9\times 1 & 4\times 2+0\times 6+9\times 0 \end{pmatrix}$$

$$P = \begin{pmatrix} 33 & 14 & 20 \\ 79 & 20 & 16 \\ 77 & 17 & 8 \end{pmatrix}$$

Generalizando, seja:

$$M_{m\times l} = \begin{pmatrix} m_{1,1} & m_{1,2} & \dots & m_{1,l} \\ m_{2,1} & m_{2,2} & \dots & m_{2,l} \\ \dots & & & \\ m_{m,1} & m_{m,2} & \dots & m_{m,l} \end{pmatrix}$$

e

$$O_{l\times n} = \begin{pmatrix} o_{1,1} & o_{1,2} & \dots & o_{1,n} \\ o_{2,1} & o_{2,2} & \dots & o_{2,n} \\ \dots & & & \\ o_{l,1} & o_{l,2} & \dots & o_{l,n} \end{pmatrix}$$

Então, a matriz produto $P_{m\times n} = M_{m\times l} \times O_{l\times n}$ *é dada por:*

$$\begin{pmatrix} p_{1,1} & p_{1,2} & \dots & p_{1,n} \\ p_{2,1} & p_{2,2} & \dots & p_{2,n} \\ & & & \\ \dots & & & \\ p_{m,1} & p_{m,2} & \dots & p_{m,n} \\ & & & \end{pmatrix}$$

Where:

$p_{1,1} = m_{1,1} \times o_{1,1} + m_{1,2} \times o_{2,1} + \ldots + m_{1,l} \times o_{l,1}$

$p_{1,2} = m_{1,1} \times o_{1,2} + m_{1,2} \times o_{2,2} + \ldots + m_{1,l} \times o_{l,2}$

$\ldots$

$p_{1,n} = m_{1,1} \times o_{1,n} + m_{1,2} \times o_{2,n} + \ldots + m_{1,l} \times o_{l,n}$

$p_{2,1} = m_{2,1} \times o_{1,1} + m_{2,2} \times o_{2,1} + \ldots + m_{2,l} \times o_{l,1}$

$p_{2,2} = m_{2,1} \times o_{1,2} + m_{2,2} \times o_{2,2} + \ldots + m_{2,l} \times o_{l,2}$

$\ldots$

$p_{2,n} = m_{2,1} \times o_{1,n} + m_{2,2} \times o_{2,n} + \ldots + m_{2,l} \times o_{l,n}$

$\ldots$

$p_{m,1} = m_{m,1} \times o_{1,1} + m_{m,2} \times o_{2,1} + \ldots + m_{m,l} \times o_{l,1}$

$p_{m,2} = m_{m,1} \times o_{1,2} + m_{m,2} \times o_{2,2} + \ldots + m_{m,l} \times o_{l,2}$

$\ldots$

$p_{m,n} = m_{m,1} \times o_{1,n} + m_{m,2} \times o_{2,n} + \ldots + m_{m,l} \times o_{l,n}$

Ou, de forma abreviada, um elemento da matriz produto $P_{m\times n} = M_{m\times l} \times O_{l\times n}$*, é dada pela equação:*

$p_{i,j} = \sum_{k=1}^{l} m_{i,k} \times o_{k,j}, \forall 1 \leq i \leq m, \forall 1 \leq j \leq n$

O algoritmo a seguir implementa a multiplicação de matrizes.

```
Subprograma multi_mat(m1,o,p,m,l,n)

{e: m1,o:matriz_real matrizes a serem multiplicadas
   m: número de linhas da matriz m1 (e da matriz resultado)
```

```
   l: número de colunas da matriz m1, e de linhas da matriz o.
   n: número de colunas da matriz o (e da matriz resultado)}

{s: p:matriz real -  a matriz produto p = m1 x o}

início
  para i de 1 até m faça
    para j de 1 até n faça
      p[i,j] ← 0,0
      para  k de 1 até l faça
        p[i,j] ← p[i,j] + m1[i,k]*o[k,j]
      fim para
    fim para
  fim para
fim
```

Exercício Sugerido 24 *Desenvolver o algoritmo do programa principal para testar a multiplicação de matrizes. Elaborar casos de teste adequados.*