

ALGORITMOS

VETORES E MATRIZES

LISTA DE FIGURAS

Figura 4.1 – Exemplo de armazenamento de vetor na memória do computador	5
Figura 4.2 – Exemplo de vetor com temperaturas	6
Figura 4.3 – Posição de um Vetor	7
Figura 4.4 – Realizando uma média simples utilizando quatro variáveis.....	8
Figura 4.5 – Realizando uma média simples utilizando quatro variáveis (2)	8
Figura 4.6 – Realizando uma média simples utilizando um vetor de quatro posições	9
Figura 4.7 – Realizando uma média simples utilizando um vetor de quatro posições e lógica mais generalizada	9
Figura 4.8 – Realizando uma média simples utilizando um vetor permitindo até 10 notas.....	10
Figura 4.9 – Exercício de posições de um vetor	11
Figura 4.10 – Respostas.....	11
Figura 4.11 – Algoritmo inverte.....	13
Figura 4.12 – Armazenando apenas números terminados em 5 e 7	14
Figura 4.13 – Armazenando os números de Fibonacci	16
Figura 4.14 – Armazenando os números de Fibonacci de um intervalo informado ..	17
Figura 4.15 – Armazenando os números pares e ímpares em vetores distintos	18
Figura 4.16 – Armazenando os números pares e ímpares em vetores distintos	19
Figura 4.17 – Armazenando os números primos	20
Figura 4.18 – Exemplo de String e sua aplicação em vetores.....	21
Figura 4.19 – Processo de desenvolvimento do algoritmo	28
Figura 4.20 – Planilha eletrônica	30
Figura 4.21 – Exemplo de eixos ortonormais.....	31
Figura 4.22 – Exemplo de planilha do Excel.....	32
Figura 4.23 – Símbolo associado a banco de dados	32
Figura 4.24 – Esquema de salas e andares em um prédio	33
Figura 4.25 – Lógica algorítmica 1.....	34
Figura 4.26 – Lógica algorítmica 2.....	34
Figura 4.27 – Lógica algorítmica 3.....	35
Figura 4.28 – Exemplos de operação	36
Figura 4.29 – Aplicação de matriz	36
Figura 4.30 – Aplicação de matriz	37
Figura 4.31 – Simulação de algoritmo	37

SUMÁRIO

4 VETORES E MATRIZES	4
4.1 Definindo vetores e matrizes	4
4.1.1 Exemplos de usos de vetores	6
4.1.2 Declarando um vetor	6
4.1.3 Exemplos de uso de vetores	7
4.1.4 Exemplo conceitual	7
4.1.5 Exemplos de acesso a posições de um vetor	11
4.1.6 Exemplos de algoritmos com vetores	11
4.1.7 Lista de exercícios com vetores	13
4.2 Strings	20
4.2.1 Copiar trechos de <i>strings</i>	21
4.2.2 Excluir trechos de strings	22
4.2.3 Obter o tamanho de uma <i>string</i> qualquer	22
4.2.4 Transformar caracteres em números	23
4.2.5 Transformar números em caracteres	24
4.2.6 Buscar um ou mais caracteres numa <i>string</i>	25
4.2.7 Separar parte de uma <i>string</i>	25
4.2.8 Exemplo: cálculo dos dígitos de controle do CPF	26
4.2.9 Exercícios	29
4.3 Matrizes	30
4.3.1 Aplicações de Matrizes	32
4.3.2 Matrizes e a lógica algorítmica	33
4.3.3 Representação de matrizes	35
4.3.4 Exemplos	36
4.3.5 Exercícios	37
REFERÊNCIAS	39

4 VETORES E MATRIZES

Quando finalizamos o capítulo anterior, passamos a escrever algoritmos capazes de resolver problemas de grande complexidade. Todavia, em momento algum, pudemos armazenar conjuntos de dados e os processar de acordo com nossas necessidades.

Por exemplo, somos capazes de calcular a média de um aluno que tenha feito três provas, mas não de calcular essa média e, depois, apresentar os valores que serviram para compô-la.

Ora, até aqui, simplesmente colocaríamos a leitura do valor de cada nota dentro de um laço, que também totalizaria o valor da nota numa variável de acumulação. Naturalmente, a cada iteração, esse valor será acumulado e o valor da nota anterior será irremediavelmente perdido.

Com vetores ou matrizes, eliminaremos esse problema na medida em que os valores que armazenarmos não sejam substituídos, a não ser quando assim o quisermos.

4.1 Definindo vetores e matrizes

Vetores e matrizes são variáveis compostas e homogêneas. A distinção entre elas está no fato de os vetores serem um caso particular de matrizes, ou seja, matrizes unidimensionais são também chamadas de vetores.

A maioria das linguagens de programação, como todas as derivadas do BASIC, C., COBOL, entre outras, respeita essa definição, portanto os vetores são um conjunto de variáveis do mesmo tipo, ou seja, são homogêneos. Uma importante exceção a essa regra é a linguagem Clipper, que permite que um vetor tenha elementos não homogêneos, isto é, alguns de seus elementos podem ser números, outros, datas ou compostos por caracteres, por exemplo.

Todos os elementos do vetor têm um mesmo identificador, isto é, um mesmo nome, e são alocados continuamente na memória do computador.

Como qualquer variável, e como todos os elementos de um vetor de mesmo nome, a maneira de nos referir a cada um de seus elementos se faz por meio de um índice que permite obter o valor armazenado em determinada posição desse vetor.

Um índice, que sempre deve ser um número inteiro, nada mais é que a referência da localização do elemento do vetor.

Na inicialização do vetor, na maioria das linguagens, determina-se o seu tamanho que geralmente não se modifica, mesmo que utilizemos menos elementos do que determinado a princípio. Em algumas linguagens de programação, como o Pascal e o COBOL, é impensável “expandir” o tamanho de um vetor, pois essas linguagens tratam os vetores em posições contíguas de memória, o que impede eventuais expansões.

Em outras linguagens, como o BASIC, essa expansão é possível, mas deve ser cuidadosamente utilizada, pois se a extensão do vetor passar de certos limites e violar o espaço de memória disponível, só restará ao sistema operacional suspender a execução do programa gerando um erro de execução.

Finalmente, em outras linguagens, como o C, é possível verificar-se o espaço livre em memória, alocar-se o espaço desejado e expandir-se o vetor.

Todavia, para fins algoritmos, a maioria dos autores prefere partir do princípio que o tamanho de qualquer vetor ou matriz seja definido na área de declaração do programa e não seja expandido, princípio que também adotaremos.

A figura “Exemplo de armazenamento de vetor na memória do computador”, a seguir, representa esquematicamente como se dá o armazenamento de um vetor na memória do computador:

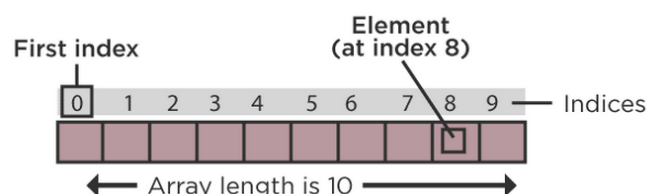


Figura 4.1 – Exemplo de armazenamento de vetor na memória do computador
Fonte: FIAP (2015)

Utilizamos um vetor para representar dados em termos de conjuntos. Um vetor, então, trata-se de uma coleção de variáveis de um mesmo tipo, que compartilham o mesmo nome e que ocupam posições consecutivas de memória.

Cada variável dessa coleção denomina-se elemento, que é identificado por um índice.

4.1.1 Exemplos de usos de vetores

Temperaturas em 10 medições num período

Conteúdo	12	9	10	16	25	13	20	14	13	14
Posição	0	1	2	3	4	5	6	7	8	9

Figura 4.2 – Exemplo de vetor com temperaturas
Fonte: FIAP (2015)

Na figura anterior temos dez tomadas de temperatura durante um período de tempo qualquer. Notar que como a primeira temperatura está armazenada na posição 0 (zero), a décima temperatura está armazenada na posição 9!

Para podermos acessar os dados armazenados no vetor, é necessário fornecer o nome do vetor (seu identificador) e o índice do elemento desejado. Cada posição do vetor contém um e somente um valor.

Assim, se desejarmos saber a temperatura observada na quarta coleta do dia, o valor será 16. Note que não será 25 (índice igual a 4), pois essa é a quinta e não a quarta medição.

A representação dessa situação, se chamarmos o vetor da figura de Temperatura, seria: Temperatura[3] que resulta em 16.

4.1.2 Declarando um vetor

Para se declarar o vetor, deve-se fornecer seu nome (como em qualquer outra variável), precedido por dois pontos, a palavra matriz, o tamanho do vetor entre colchetes (sua capacidade) seguido do tipo de dados que o vetor irá armazenar:

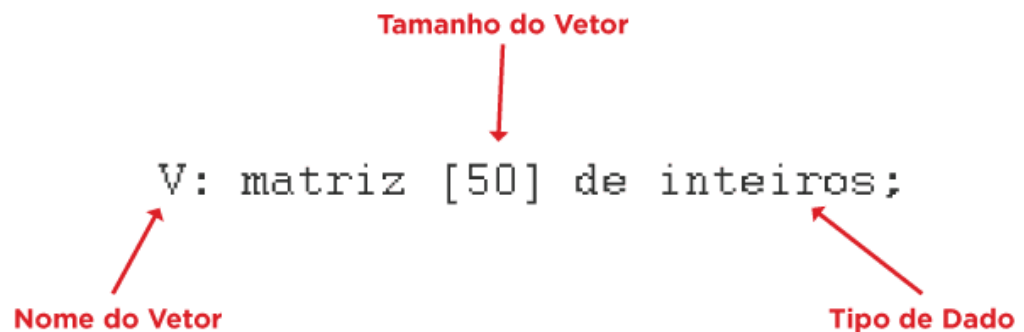


Figura 4.3 – Posição de um Vetor
Fonte: FIAP (2015)

Onde:

Tamanho do Vetor é um número inteiro e positivo, em nosso exemplo será 50, ou seja, nosso vetor terá 50 posições.

4.1.3 Exemplos de uso de vetores

Toda vez que necessitarmos armazenar um conjunto de valores em memória, um vetor parece ser o melhor candidato para a tarefa, são exemplos de uso de vetores:

- Tempos cronometrados para produção de determinada peça, numa fábrica.
- Valores de temperaturas em medições regulares.
- Valores de umidade relativa do ar obtidas ao longo de um dia.
- Valores de notas obtidas por um aluno em provas realizadas.
- Valores de preços de produtos.

4.1.4 Exemplo conceitual

Calcule a média aritmética de um aluno, considere que existam quatro notas.

Solução 1:

```
algoritmo media01;
variáveis
    a, b, c, d : inteiro;
fim-variáveis
início
    a := leia();
    b := leia();
    c := leia();
    d := leia();
    imprima((a+b+c+d)/4);
fim
```

Figura 4.4 – Realizando uma média simples utilizando quatro variáveis
Fonte: Elaborado pelo autor (2015)

Em nossa primeira solução, adotamos quatro variáveis quaisquer e, depois de sua leitura, simplesmente apresentamos o resultado da divisão da respectiva soma, dividindo por quatro, que vem a ser o número de termos lido.

Solução 2:

```
algoritmo media02;
variáveis
    a1, a2, a3, a4 : inteiro;
fim-variáveis
início
    a1 := leia();
    a2 := leia();
    a3 := leia();
    a4 := leia();
    imprima((a1+a2+a3+a4)/4);
fim
```

Figura 4.5 – Realizando uma média simples utilizando quatro variáveis (2)
Fonte: Elaborado pelo autor (2015)

Nessa segunda solução, em suma, nada foi alterado. Somente as variáveis tentam simular a existência de uma continuidade, quase como se tivessem uma forma de indexação através da numeração sequencial.

Solução 3:


```
algoritmo media03;
variáveis
    a : matriz [4] de inteiros;
fim-variáveis
início
    a[0] := leia();
    a[1] := leia();
    a[2] := leia();
    a[3] := leia();
    imprima((a[0]+a[1]+a[2]+a[3])/4);
fim
```

Figura 4.6 – Realizando uma média simples utilizando um vetor de quatro posições
Fonte: Elaborado pelo autor (2015)

Nessa terceira solução, deixamos de simular a existência de uma continuidade, que agora passa efetivamente a existir. Trocamos quatro variáveis por um vetor!

No entanto, a solução pode ser melhorada:

Solução 4:

```
algoritmo media04;
variáveis
    a : matriz [10] de inteiros;
    i, soma : inteiro;
fim-variáveis
início
    para i de 1 até 4 faça
        a[i] := leia();
    fim-para
    soma := 0;
    para i de 1 até 4 faça
        soma := soma + a[i];
    fim-para
    imprima(soma/4);
fim
```

Figura 4.7 – Realizando uma média simples utilizando um vetor de quatro posições e lógica mais generalizada
Fonte: Elaborado pelo autor (2015)

Nessa quarta solução, notamos que o vetor deixou de ser diretamente acessado por instruções que especificavam posições absolutas no programa, para termos acessos feitos através de um contador, nosso índice, que foi criado exatamente para tal finalidade.

Facilmente podemos perceber que essa quarta versão pode ser generalizada para o cálculo de média de um número variável de notas, desde que inferior ao número máximo de notas estipulado na declaração da variável vetor “a”.

Solução 5:

```
algoritmo media_05;  
variáveis  
    a : matriz [10] de inteiros;  
    i, soma, n : inteiro;  
fim-variáveis  
início  
    n := leia();  
    para i de 1 até n faça  
        a[i] := leia();  
    fim-para  
    soma := 0;  
    para i de 1 até n faça  
        soma := soma + a[i];  
    fim-para  
    imprima(soma/n);  
fim
```

Figura 4.8 – Realizando uma média simples utilizando um vetor permitindo até 10 notas
Fonte: Elaborado pelo autor (2015)

Finalmente, chegamos a uma solução “genérica” para o cálculo da média de uma série de notas, sem definirmos previamente quantas serão as notas. Voltamos a enfatizar, o usuário pode escolher fornecer até 10 notas (limite do vetor).

Devemos ressaltar que a grande distinção da “solução 5” quando comparada a “solução 3” e anteriores, reside no fato de que depois da apresentação da média, ainda temos em memória cada uma das notas que foram usadas para compor essa média. Ou seja, não perdemos as partes que serviram para o cálculo que desejávamos fazer, no primeiro momento.

4.1.5 Exemplos de acesso a posições de um vetor

Sendo o vetor V igual a:

V	2	6	8	3	10	9	1	21	33	14
	1	2	3	4	5	6	7	8	9	10

e as variáveis $X=2$ e $Y=4$, escreva o valor correspondente a cada solicitação abaixo:

- | | |
|----------------|---------------------|
| a. $V[X+1]$ | i. $V[X+Y]$ |
| b. $V[X+2]$ | j. $V[8 - V[2]]$ |
| c. $V[X+3]$ | l. $V[V[4]]$ |
| d. $V[X*4]$ | m. $V[V[V[7]]]$ |
| e. $V[X*1]$ | n. $V[V[1] * V[4]]$ |
| f. $V[X*2]$ | o. $V[X+4]$ |
| g. $V[X*3]$ | |
| h. $V[V[X+Y]]$ | |

Figura 4.9 – Exercício de posições de um vetor
Fonte: Google Imagens (2015)

Respostas:

A.	$V[X+1]$	$V[2+1]$	$V[3]$	8	
B.	$V[X+2]$	$V[2+2]$	$V[4]$	3	
C.	$V[X+3]$	$V[2+3]$	$V[5]$	10	
D.	$V[X*4]$	$V[2*4]$	$V[8]$	21	
E.	$V[X*1]$	$V[2*1]$	$V[2]$	6	
F.	$V[X*2]$	$V[2*2]$	$V[4]$	3	
G.	$V[X*3]$	$V[2*3]$	$V[6]$	9	
H.	$V[V[X+Y]]$	$V[V[2+4]]$	$V[V[6]]$	$V[9]$	33
I.	$V[X+Y]$	$V[2+4]$	$V[6]$	9	
J.	$V[8-V[2]]$	$V[8-6]$	$V[2]$	6	
L.	$V[V[4]]$	$V[3]$	8		
M.	$V[V[V[7]]]$	$V[V[1]]$	$V[2]$	6	
N.	$V[V[1]*V[4]]$	$V[2*3]$	$V[6]$	9	
O.	$V[X+4]$	$V[2+4]$	$V[6]$	9	

Figura 4.10 – Respostas
Fonte: Elaborado pelo autor (2015)

4.1.6 Exemplos de algoritmos com vetores

- a) Elabore um algoritmo que receba do usuário “n” números inteiros (até 10), e os imprima na ordem oposta da qual foram informados.

- b) Elabore um algoritmo que receba do usuário “n” números inteiros (até 10) e armazene-os numa matriz (vetor), e imprima apenas os ímpares armazenados.
- c) Elabore um algoritmo que receba do usuário a quantidade “n” de números (até 100) que ele deseja, calcule e armazene os “n” primeiros termos de Fibonacci numa matriz. Considere como termos de Fibonacci a seguinte série: 1,2,3,5,8,13,21,34, etc.
- d) Elabore um algoritmo que receba do usuário “n” números inteiros (até 10), armazenando os números pares informados e uma matriz e os ímpares informados numa segunda matriz. Considere zero par.
- e) Elabore um algoritmo que receba do usuário “n” números inteiros (até 10) e armazene-os em uma matriz. Em seguida, armazene em uma segunda matriz o produto do 1º termo pelo 2º, do 2º pelo 3º e assim sucessivamente.
- f) Elabore um algoritmo que receba do usuário “n” números inteiros (até 10) e armazene apenas os ímpares em uma matriz. Em seguida, armazene numa segunda matriz somente os primos contidos na primeira matriz.
- g) Elabore um algoritmo que receba do usuário dois números inteiros e positivos (com uma diferença de até 100 entre eles). Armazene em uma matriz apenas os ímpares existentes no intervalo entre eles. Em seguida, armazene em uma segunda matriz somente os primos.
- h) Elabore um algoritmo que receba do usuário dois números inteiros e positivos (com uma diferença de até 100 entre eles). Armazene em uma matriz os primos existentes no intervalo entre eles e os imprima-os em ordem invertida.
- i) Elabore um algoritmo que receba do usuário dois números inteiros e positivos (com uma diferença de até 100 entre eles). Armazene em uma matriz (vetor) números que sejam ímpares ou que o número da dezena seja ímpar.

Exemplo: 249 (ímpar), 252 (cinco é ímpar) são armazenados. Já 842 ou 364 não são armazenados (além de pares, o quatro em 842 e o seis em 364 não são ímpares).

4.1.7 Lista de exercícios com vetores

- a) Elabore um algoritmo que receba do usuário “n” números inteiros e armazene em um vetor apenas aqueles que sejam maiores que a média dos números informados, imprimindo posteriormente o conteúdo deste vetor. Sugestão: armazene todos os números em um vetor temporário, calcule a média e, a partir dos elementos existentes no vetor temporário, atenda ao enunciado.

Solução:

```
algoritmo inverte;
variáveis
  i, n : inteiro;
  v : matriz [100] de inteiros;
fim-variáveis
início
  imprima("Total de Números: ");
  n := leia();
  para i de 1 até n faça
    imprima("Digite o ", i, "o. Numero: ");
    v[i] := leia();
  fim-para
  para i de n até 1 passo -1 faça
    imprima(v[i]);
  fim-para
fim
```

Figura 4.11 – Algoritmo inverte
Fonte: Elaborado pelo autor (2015)

Neste exemplo, podemos ressaltar vários aspectos importantes:

- Observemos que o índice “i” é usado primeiramente para ler o vetor e, em seguida, para sua apresentação. Poderíamos facilmente ter usado um índice “j” para sua apresentação. Ou seja, o índice “i” de nenhuma maneira faz parte vetor “v”, como habitualmente os estudantes confundem ao menos num primeiro momento.
- O uso do para é bastante adequado, pois cuida de iniciar e controlar o contador, que no caso também é um índice, para o programador. Sempre que possível, devemos deixar ao encargo da linguagem aqueles processos

que esta pode dar a devida conta, reservando para nosso encargo aqueles aspectos lógicos que são inerentes à solução do problema.

Sugestão: Simular para 4 números, a saber: 7; 5; 2; e 9.

- b) Elabore um algoritmo que solicite do usuário uma série de números, armazene e exiba (DEPOIS DE ARMAZENADO) em um vetor somente aqueles que terminarem em 7 ou em 5.

Solução:

```
algoritmo a7ou5;
variáveis
    i, n, c, j : inteiro;
    v : matriz[100] de inteiros;
fim-variáveis
início
    imprima("Total de Números: ");
    n := leia();
    j := 1;
    para i de 1 até n faça
        imprima("Digite o ", i, "o. Numero: ");
        c := leia();
        se c % 10 = 7 ou c % 10 = 5 então
            v[j] := c;
            j := j + 1;
        fim-se
    fim-para
    para i de 1 até j-1 faça
        imprima(v[i]);
    fim-para
fim
```

Figura 4.12 – Armazenando apenas números terminados em 5 e 7
Fonte: Elaborado pelo autor (2015)

Neste exemplo, dos mais importantes, podemos ressaltar vários aspectos significativos:

- Observemos que o vetor “v” é montado a partir de um índice “j” que não coincide com o índice “i”, a menos que todos os números informados em nosso teste sejam terminados em 7 e 5. De fato, quando o número terminar nesses números, o armazenaremos na posição “j” e em seguida a incrementamos. Assim, na melhor das hipóteses, ou seja, se todos os números informados terminarem em 7 ou 5, ao final do primeiro loop, “i”

valerá “n+1” e “j” também valerá “n+1”, ou seja, “i” terá estourado o contador e, como “j” sempre aponta para uma posição adiante da que está armazenando, também terá como valor “n+1”.

- Em contrapartida, se todos os números informados tiverem seu final diferente de 7 e 5, ao final do processo, “j” terminará valendo 1. Ora, como “j” sempre termina com o valor adicionado de itens que atenderam a necessidade proposta pelo exercício adicionada de 1, nesse caso, o número de termos adicionado foi zero!
- Daí se entende o porquê do contador “i” do laço em que se exibem os números armazenados ir de 1 até “j-1”. Ora, se “j” contém o total de números armazenados adicionados de 1, na pior das hipóteses terá valor 1. Assim, ao subtrairmos 1 de “j”, faríamos um contador que variasse de 1 até 0! como não há nenhum valor nesse intervalo, o laço não será executado, ou seja, nenhum número será exibido.

Sugestão: Fazer três simulações, nas seguintes condições:

5 números, a saber: 17; 13; 22; 25 e 37

5 números, a saber: 17; 35; 5; 35 e 37

5 números, a saber: 11; 13; 22; 21 e 31

c) Elabore um algoritmo que recebe do usuário 3 números inteiros e distintos entre si, armazene num vetor os números existentes entre o número menor e o médio (inclusive) e em outro vetor, os números existentes entre o médio (exclusive) e o maior (inclusive). Posteriormente, exiba os dois conjuntos de números armazenados. Admita que todos os números informados sejam distintos entre si e maiores que zero.

Solução:

Nesse algoritmo, o aspecto mais relevante a ser destacado é exatamente o uso alternado das variáveis “i” e “j”, “k” como índices do vetor. Nos dois primeiros laços notamos que enquanto “i” possui o valor a ser armazenado, “j” e “k” funcionam como índice, dos vetores “v” e “w”.

Quando totalmente carregado o vetor “v”, temos em “j” o total de elementos mais um, armazenado em “v”. O mesmo ocorre com o vetor “w”, somente que em função de “k”.

Finalmente, para apresentar os valores carregados, basta lembrarmos que os totais de elementos armazenados estão guardados em “j” e “k”, dos vetores “v” e “w”, que passam a ser endereçados pelo indexador “i”, nesse caso, o próprio contador dos dois últimos para.

d) Elabore um algoritmo que armazene em um vetor os primeiros “n” números de Fibonacci (até o limite de 100), descontados os termos formadores (0 e 1).

Termos a armazenar: 1,2,3,5,8,13,21,34,55,etc.

Solução:

```
algoritmo fibonacci_vetor;
variáveis
    i, a,b,c, n : inteiro;
    v : matriz[100] de inteiros;
fim-variáveis
início
    imprima("Total de Números: ");
    n := leia();
    a := 0;
    b := 1;
    para i de 1 até n faça
        c := a+b;
        v[i] := c;
        a := b;
        b := c;
    fim-para
    para i de 1 até n faça
        imprima(v[i]);
    fim-para
fim
```

Figura 4.13 – Armazenando os números de Fibonacci
Fonte: Elaborado pelo autor (2015)

Nesse exemplo, retoma-se o algoritmo de Fibonacci visto anteriormente, entretanto em vez de apresentar cada número determinado, simplesmente os armazenamos num vetor “v” que é controlado pelo índice “i”, que também faz o papel de contador de termos de Fibonacci. Note-se que essa coincidência é que permitiu o

uso de “i” tanto como contador de termos de Fibonacci, como também de índice do vetor “v”.

e) Elabore um algoritmo que receba do usuário dois números inteiros e armazene em um vetor apenas números que pertençam à série de Fibonacci e estejam contidos no intervalo destes dois números informados (inclusive os próprios números).

Solução:

```
algoritmo fibonacci_vetor_2;
variáveis
    i, a,b,c, n, m,j : inteiro;
    v : matriz[100] de inteiros;
fim-variáveis
início
    imprima("Número Inicial e Final: ");
    m := leia();
    n := leia();
    a := 0;
    b := 1;
    c := 1;
    i := 1;
    enquanto c <= n faça
        c := a+b;
        se m <= c então
            v[i] := c;
            i := i+1;
        fim-se
        a := b;
        b := c;
    fim-enquanto
    para j de 1 até i-1 faça
        imprima(v[j]);
    fim-para
fim
```

Figura 4.14 – Armazenando os números de Fibonacci de um intervalo informado
Fonte: Elaborado pelo autor (2015)

Diferentemente do exemplo anterior, não sabemos quantos elementos serão armazenados no vetor, portanto, em vez de usarmos para que implica num número determinado de iterações, foi utilizada a instrução enquanto para controle do laço.

f) Elabore um algoritmo que receba do usuário “n” (até 100) números inteiros e armazene no vetor “A” os pares e em “B” os ímpares informados.

Solução:

```
algoritmo vetores_a_e_b;
variáveis
    i, n, iv, iw, c : inteiro;
    a,b : matriz[100] de inteiros;
fim-variáveis
início
    imprima("Total de Números: ");
    n := leia();
    iv := 1;
    iw := 1;
    para i de 1 até n faça
        imprima("Digite o ",i,"o. Numero: ");
        c := leia();
        se c%2 = 0 então
            a[iv] := c;
            iv := iv + 1;
        senão
            b[iw] := c;
            iw := iw + 1;
        fim-se
    fim-para
    imprima("Pares");
    para i de 1 até iv-1 faça
        imprima(a[i]);
    fim-para
    imprima("Impares");
    para i de 1 até iw-1 faça
        imprima(b[i]);
    fim-para
fim
```

Figura 4.15 – Armazenando os números pares e ímpares em vetores distintos
Fonte: Elaborado pelo autor (2015)

O ponto a destacar nesse exemplo é a necessidade da existência de um contador, no caso “i” e dois indexadores “iv” e “iw”, respectivamente para os vetores “a” e “b”, que posteriormente tem seus valores armazenados usando o índice “i”, que também é contador, no momento da exibição dos dados.

- g) Leia dois vetores “A” e “B” com “n” elementos, construindo em “C” a subtração dos elementos correspondentes de “A” com “B”.

Solução:

```
algoritmo vetores_a_b_e_c;
variáveis
    i, n : inteiro;
    a,b,c : matriz[100] de inteiros;
fim-variáveis
início
    imprima("Total de Números de cada vetor: ");
    para i de 1 até n faça
        imprima("Digite o ",i,"o. Numero: ");
        a[i] := leia();
    fim-para
    n := leia();
    para i de 1 até n faça
        imprima("Digite o ",i,"o. Numero: ");
        b[i] := leia ();
    fim-para
    para i de 1 até n faça
        c[i] := a[i] - b[i];
    fim-para
    para i de 1 até n faça
        imprima(c[i]);
    fim-para
fim
```

Figura 4.16 – Armazenando os números pares e ímpares em vetores distintos
Fonte: Elaborado pelo autor (2015)

Embora tenhamos três vetores, existe uma coincidência de posições sendo trabalhadas, implicando na utilização do índice e contador “i” nos três laços existentes no algoritmo.

h) Dados dois números, armazene num vetor os primos existentes entre ambos.

Solução:

```
algoritmo primo_vetor;
variáveis
    a, b, i, j, k : inteiro;
    primo : lógico;
    v : matriz[100] de inteiros;
fim-variáveis
início
    imprima("1o Número: ");
    a := leia();
    imprima("Último Número: ");
    b := leia();
    k := 1;
    para j de a até b faça
        primo := verdadeiro;
        i := 2;
        enquanto (i <= j / 2) e (primo = verdadeiro) faça
            se j % i = 0 então
                primo := falso;
            fim-se
            i := i + 1;
        fim-enquanto
        se primo então
            v[k] := j;
            k := k + 1;
        fim-se
    fim-para
    para i de 1 até k-1 faça
        imprima(v[i]);
    fim-para
fim
```

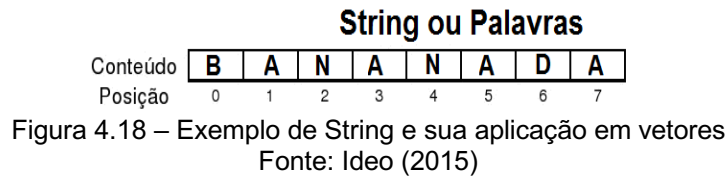
Figura 4.17 – Armazenando os números primos
Fonte: Elaborado pelo autor (2015)

De forma análoga ao que ocorreu no exemplo usando Fibonacci, a estratégia foi armazenar num vetor os primos identificados, em vez de apenas apresentá-los, como foi feito no exemplo em que os primos foram determinados.

4.2 Strings

As strings ou sequência de caracteres são um tipo bastante especial de aplicações de vetores. Basicamente, strings são palavras, ou seja, vetores que armazenam caracteres.

Para entendermos a distinção entre um vetor de caracteres e outro qualquer, provavelmente basta entendermos a estrutura apresentada no esquema a seguir:



A analogia com o esquema apresentado anteriormente, em que números estavam armazenados, em vez de letras é imediata.

Todavia, podemos pensar em acessar essa sequência de letras, como um vetor, ou seja, endereçando cada elemento individualmente, o que apresenta algumas vantagens e várias desvantagens, quando queremos enxergar a “palavra” BANANADA.

Pensando nisso, podemos tratar esse “vetor” como sendo uma variável única, e desde que tenhamos operadores que possam tratar cada ou um conjunto de caracteres por vez, ficamos numa situação bastante vantajosa.

Na verdade, grande parte dos procedimentos que serão realizados com as strings exige manipulação de cadeias de caracteres, algo que podemos definir como uma sequência de letras, de números ou de outros caracteres significativos, que suporta inúmeras operações.

Note-se, entretanto, que se pensarmos em números, cada caractere é tratado individualmente, portanto, se tivermos dígitos numéricos, esses não farão sentido como dígitos, isto é, não suportarão operações como soma, subtração, multiplicação ou divisão, pois não são “números”, mas somente dígitos!

As sintaxes aqui observadas variam, em sua forma, de autor para autor e de linguagem para linguagem, portanto é possível que a ordem de parâmetros, mesmo seu número, seja distinta de alguns livros e linguagens de programação.

Seguindo a linha proposta por Salvetti e Barbosa (1998), adotamos aqui os nomes das instruções na língua inglesa, pela maior proximidade com os comandos das linguagens de programação.

4.2.1 Copiar trechos de *strings*

Comando **COPY** (*string*, posição, quantidade)

Onde:

String é a cadeia de caracteres a serem manipulados.

Posição – Trata-se de número inteiro que indica a posição inicial a partir do qual os caracteres serão copiados.

Quantidade – Trata-se de número inteiro que indica quantos caracteres serão copiados.

Exemplos:

$S \leftarrow \text{"BANANADA"}$

Imprima $\text{Copy}(S, 5, 4) \rightarrow \text{"NADA"}$

Imprima $\text{Copy}(S, 2, 3) \rightarrow \text{"ANA"}$

4.2.2 Excluir trechos de strings

Comando **DELETE** (*string*, posição, quantidade)

Onde:

String é a cadeia de caracteres a serem manipulados.

Posição – Trata-se de número inteiro que indica a posição inicial a partir do qual os caracteres serão excluídos.

Quantidade – Trata-se de número inteiro que indica quantos caracteres serão excluídos.

Exemplo:

$S \leftarrow \text{"BANANADA"}$

$T \leftarrow \text{DELETE}(S, 4, 3)$

Imprima $T \rightarrow \text{"BANDA"}$

4.2.3 Obter o tamanho de uma *string* qualquer

Comando **LENGTH** (*string*)

Onde:

String é a cadeia de caracteres a serem manipulados.

Exemplos:

$S \leftarrow \text{"BANANADA"}$

Imprima $\text{LENGTH}(S) \rightarrow 8$

$S \leftarrow \text{"BANANADA "}$

Imprima $\text{LENGTH}(S) \rightarrow 10$

Observe que a existência de dois espaços em branco, que pode passar despercebida numa rápida leitura é contada, pois o “espaço em branco” é um caractere como qualquer outro. No exemplo a seguir, utilizou-se o “b cortado” para representar o caractere “em branco”. Em letra corrida é usual escrever-se e letra minúscula “b” e em seguida “cortá-la”. Já em texto corrido, usualmente escreve-se a letra “b” e a barra “/” em seguida, como no exemplo a seguir:

$S \leftarrow \text{"BANANADAb/b/"}$

Imprima $\text{LENGTH}(S) \rightarrow 10$

4.2.4 Transformar caracteres em números

Comando **VAL (String)**

Onde:

String é a cadeia de caracteres a serem manipulados.

Exemplos:

$S \leftarrow \text{"12"}$

Imprima $\text{VAL}(S) \rightarrow 12$ (número)

Cabem várias observações com relação à instrução acima, a saber:

- Certas linguagens de programação, como o Pascal, obrigam que os caracteres a serem convertidos sejam dígitos numéricos, sob pena de erro de conversão. O BASIC, por exemplo, ao tentar transformar um caractere que não seja um dígito, o faz transformando-o num valor zero.

- Há linguagens que VAL não se parece com uma função, algo que recebe um parâmetro, no caso uma string, e devolve um valor. Nessas linguagens, como é o caso do Pascal, VAL recebe também a variável que deverá receber o valor numérico, resultante da conversão da string.
- Há outras linguagens, que por serem fracamente tipadas, podem fazer essa conversão diretamente, como é o caso da Linguagem C. Neste caso, a função VAL não existe, pois é desnecessária.

4.2.5 Transformar números em caracteres

Comando **STR (Número)**

Onde:

Número – Trata-se de número inteiro ou real, a ser transformado em sequência de caracteres.

Exemplos:

$N \leftarrow 20$

$S \leftarrow \text{STR}(N)$

Imprima S \rightarrow "20"

Cabem várias observações com relação à instrução acima, a saber:

- Certas linguagens de programação, como o Clipper, COBOL e BASIC possibilitam determinar o número de casas decimais e o tamanho da string resultante. Todavia, se a conversão levar a algum erro podemos ter truncamento da string resultante ou sua substituição por um sinal de erro, que é usualmente o asterisco "*".
- Outras linguagens, como é o caso do Pascal, permitem a passagem de apenas um parâmetro, mas este pode ser formatado, da seguinte forma:

$N \leftarrow 20.35$

$S \leftarrow \text{STR}(N:4)$

Imprima S \rightarrow "20.4"

$S \leftarrow \text{STR}(N:6:2))$

Imprima $S \rightarrow " 20.35"$ (observe a presença de um espaço em branco antes do número vinte e o tamanho total da *string* de seis caracteres, sendo dois reservados para as casas decimais do número).

4.2.6 Buscar um ou mais caracteres numa *string*

Comando **AT (Caractere,String)**

Onde:

Caractere – é o caractere a ser procurado

String – é a cadeia de caracteres a serem manipulados.

A posição onde se encontra o caractere é retornada por AT. Em não existindo, zero é retornado.

Exemplos:

$S \leftarrow \text{"BANANADA"}$

$A \leftarrow \text{AT("N",S)}$

$B \leftarrow \text{AT("X",S)}$

Imprima A " e " B $\rightarrow 3$ e 0

4.2.7 Separar parte de uma *string*

Comando **SubStr(String,Posicao_Inicial,Quantidade)**

Onde:

String – é a cadeia de caracteres a serem manipulados.

Posicao_Inicial – é a posição inicial a partir da qual será separada uma parte da string (essa posição é considerada para efeito do resultado da separação).

Quantidade – indica a quantidade de dígitos a ser separado (observar que em certas linguagens de programação, em vez de quantidade, devemos especificar aqui a posição final a ser considerada).

Exemplos:

$S \leftarrow \text{"BANANADA"}$

$A \leftarrow \text{SubStr}(\text{BANANADA}, 5, 4)$

Imprima $A \rightarrow \text{'NADA'}$

4.2.8 Exemplo: cálculo dos dígitos de controle do CPF

O cálculo dos dígitos de controle do CPF é um algoritmo bastante interessante, para apresentação das técnicas associadas aos comandos de manipulação de conjuntos de caracteres ou strings. No caso do CPF, são válidos os seguintes aspectos:

- O CPF possui 11 dígitos sendo que os dois últimos são dígitos de controle.
- O primeiro dígito de controle é calculado com base nos nove primeiros dígitos.
- O segundo dígito de controle é calculado com base nos nove primeiros dígitos e no primeiro dígito de controle
- A **Região Fiscal** onde é emitido o **CPF** (definida pelo nono dígito) tem a seguinte abrangência: **1** (DF-GO-MS-MT-TO), **2** (AC-AM-AP-PA-RO-RR), **3** (CE-MA-PI), **4** (AL-PB-PE-RN), **5** (BA-SE), **6** (MG), **7** (ES-RJ), **8** (SP), **9** (PR-SC) e **0** (RS).

Solução:

Considere a seguinte representação do CPF em forma de letras: ABC.DEF.GHI-JK. Onde cada uma das letras representa um dígito do CPF.

O **dígito de controle J** é calculado pela seguinte expressão:

$$\text{Soma} = 10*A + 9*B + 8*C + 7*D + 6*E + 5*F + 4*G + 3*H + 2*I$$

Resto = resto(Soma, 11), ou seja, resto da divisão de "Soma" por 11.

Se resto ≤ 1 então

$$J = 0$$

Senão

$$J = 11 - \text{Resto}$$

Uma vez calculado o dígito J (primeiro dígito de controle), devemos calcular o **dígito de controle K** (segundo dígito de controle). Trata-se de uma operação muito semelhante a anterior, como segue:

$$\text{Soma} = 11*A + 10*B + 9*C + 8*D + 7*E + 6*F + 5*G + 4*H + 3*I + 2*J$$

$$\text{Resto} = \text{resto}(\text{Soma}, 11), \text{ ou seja, resto da divisão de "Soma" por 11.}$$

Se $\text{resto} \leq 1$ então

$$K = 0$$

Senão

$$K = 11 - \text{Resto}$$

O processo de desenvolvimento desse algoritmo consiste em, primeiramente, na leitura de um número representando os nove primeiros dígitos do CPF, para o cálculo dos dígitos de controle.

Depois, o algoritmo deverá calcular cada dígito de verificação, concatenando-os a sequência original de nove dígitos. Notar que com ambos os dígitos de controle calculados e feita a concatenação, a variável bCPF, que contém o CPF, passa a ter 11 posições.

```

Variaveis
bCPF String[11]
Tamanho inteira
c Caractere
n inteira
b inteira
s inteira
r inteira
dig1 inteira
rf caractere
inicio
imprima 'Digite uma sequência de Nove Dígitos (CPF): '
Leia bCPF
Tamanho := Length(bCPF);
Se tamanho <> 9 then
  imprima 'Você deve informar 9 dígitos exatamente!
senao
  b := 10
  s := 0
  para i := 1 até 9 faÁa
    c := SubStr(bCPF,i,1)
    n := Val(c)
    s := s + b * n
    b := b - 1
  r := resto(s,11)
  Se r <= 1 ent,,o
    dig1 := 0
  sen,,o
    dig1 := 11 - r
  bCPF := bCPF + Str(dig1)
  b := 11
  s := 0
  para i <- 1 até 10
    c := SubStr(bCPF,i,1)
    n := Val(c)
    s := s + b * n
    b := b - 1
  r := resto(s,11)
  Se r <= 1 ent,,o
    dig1 := 0
  sen,,o
    dig1 := 11 - r
  bCPF := bCPF + Str(dig1)
  b := 11
  s := 0
  para i <- 1 até 10
    c := SubStr(bCPF,i,1)
    n := Val(c)
    s := s + b * n
    b := b - 1
  r := resto(s,11)
  Se r <= 1 ent,,o
    dig1 := 0
  sen,,o
    dig1 := 11 - r
  imprima '1o dígito: ' + dig1
  bCPF := bCPF + Str(dig1)
  rf := SubStr(bCPF,9,1);
Caso rf
  '0' : imprima 'Rio Grande do Sul -->' + bCPF
  '1' : imprima 'Centro-Oeste (MS+DF+GO+TO+MT) -->' + bCPF
  '2' : imprima 'Norte (AC+AM+PA+RR+RO+AP) -->' + bCPF
  '3' : imprima 'Nordeste-Norte (CE+PI+MA) -->' + bCPF
  '4' : imprima 'Nordeste-Leste (AL+PE+PB+RN) -->' + bCPF
  '5' : imprima 'Nordeste-Sul (BA+SE) -->' + bCPF

```

Figura 4.19 – Processo de desenvolvimento do algoritmo

Fonte: Elaborado pelo autor (2015)

4.2.9 Exercícios

a) Considere a string 'BANANADA' e combine as instruções de manipulação de *string* para:

- Imprimir ANA, usando substr.
- Substituir a *string* 'BANANADA' por 'BANDA', usando a instrução delete.
- Indicar as posições de todos os 'A's existentes na palavra 'BANANADA'.

b) Indique se uma expressão é palíndrome.

Exemplo: "AMOR" e "ROMA" não são palíndromes.

"OVO" e "OVO" ou "AMOR ME AMA EM ROMA" e "AMOR ME AMA EM ROMA" são palíndromes.

c) Dado um texto contendo somente palavras separadas por caracteres em branco, terminando em ponto final, escreva apenas as palavras distintas existentes no texto.

Texto = "EU SOU MAIS EU ENQUANTO EU SOU EU."

Resposta = "EU SOU MAIS ENQUANTO"

d) Considere a variável Numero = "UM DOIS TRES QUATRO CINCO SEIS SETE OITO NOVE " e a variável VALOR com um número entre 1 e 9. Imprima o dígito e o número em português.

e) Multiplique um número qualquer de quatro dígitos por um número de apenas um dígito, seguindo o processo descrito a seguir:

NG = '8517'

A = 2

Valor = A * 7 = 14, R = "4" e VaiUm = 1

Valor = A * 1 = 2, R = "34" e VaiUm = 0

Valor = A * 5 = 10, R = "034" e VaiUm = 1

Valor = A * 8 = 16, R = "7034" e VaiUm = 1

R = "17034".

4.3 Matrizes

Matrizes são variáveis compostas homogêneas e multidimensionais, diferindo dos vetores. São formadas por uma sequência de variáveis, todas do mesmo tipo, com o mesmo identificador (mesmo nome), de forma idêntica aos vetores.

Uma matriz de duas dimensões será chamada bidimensional, uma de três dimensões tridimensional e assim por diante. Já um vetor nada mais é que uma matriz unidimensional.

A maior distinção entre os dois conceitos está na necessidade das matrizes usarem índices para cada uma de suas dimensões, sendo que no vetor existe apenas um índice, posto haver apenas uma dimensão.

Devemos não confundir esse fato, com as circunstâncias que nos permitem usar mais de uma variável para indexar um vetor, pois podemos querer um índice indo do primeiro para o último elemento e outro percorrendo o vetor em sentido oposto, por exemplo.

Outra maneira de aprender matrizes é pensá-la como sendo uma tabela ou uma planilha eletrônica, pois ambas são matrizes.

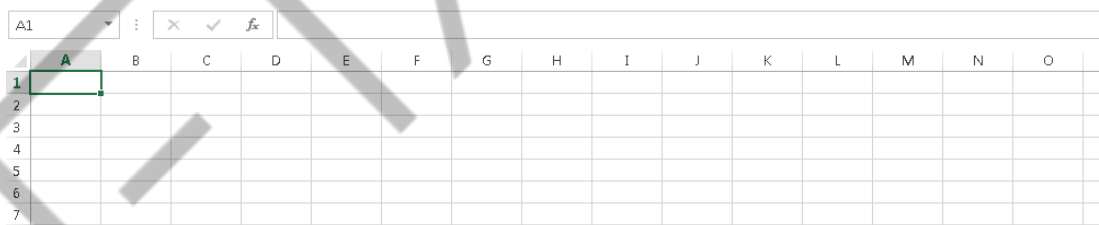


Figura 4.20 – Planilha eletrônica
Fonte: FIAP (2015)

Na imagem da planilha (Figura 6), é possível identificar duas dimensões: linhas numeradas a partir do 1 e colunas identificadas a partir do A.

Vamos observar algumas de suas propriedades comuns:

- Elas (tabelas ou matrizes) possuem duas dimensões $m \times n$, onde m é o número de linhas e n é o número de colunas.

- Ambas assumem valores nos “encontros” entre linhas e colunas. Notar que mesmo um valor nulo, como no caso da planilha exemplo, continua sendo um valor.

Outro exemplo a considerar é pensar numa matriz como sendo um vetor de vetores. Bastaria pensarmos num vetor, porém cada um de seus elementos seria composto por outro vetor. Naturalmente, essa situação, se infinitamente repetida, nos levaria à paradoxal situação de uma matriz infinita, que não seria suportada por qualquer que fosse o computador.

Matrizes de três dimensões podem ser representadas em cubos, quase sempre mencionadas em disciplinas ligadas à Pesquisa Operacional ou em Business Intelligence. Ainda são usadas em física ou matemática, para representar os eixos ortonormais, como mostra a figura a seguir:

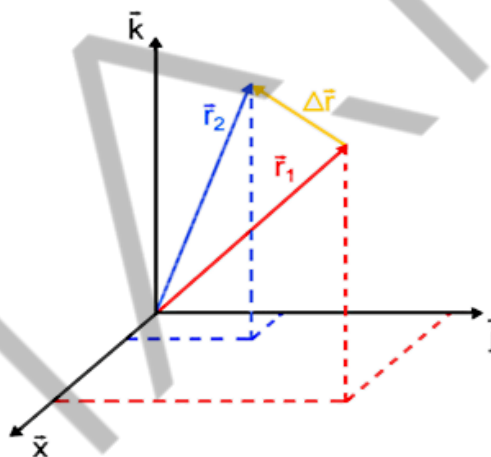


Figura 4.21 – Exemplo de eixos ortonormais
Fonte: FIAP (2015)

Poderíamos chamar de matriz S (de Space) como três eixos, no caso representados pelos vetores x (abscissa), j (ordenada) e k (azimute). Por exemplo, um ponto na posição de repouso seria representado por $S(0,0,0)$.

Podemos imaginar, ainda, uma coleção de tabelas como são várias planilhas do Excel, dentro de um único arquivo.

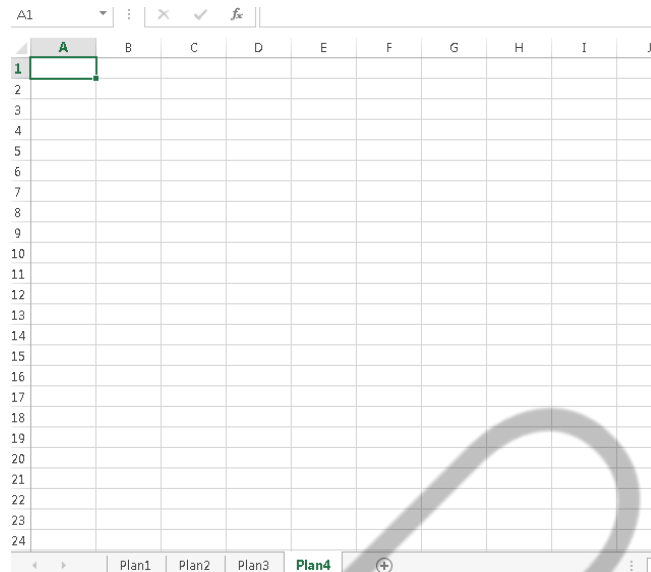


Figura 4.22 – Exemplo de planilha do Excel
Fonte: FIAP (2015)

Nesse caso, além das linhas e colunas, que nos dão a ideia de altura e largura, poderíamos tratar cada uma das planilhas como sendo a profundidade.

Mesmo que algebricamente seja fácil pensar em matrizes com mais de três dimensões, torna-se muito mais difícil representá-las graficamente. Precisamos de algumas abstrações como desenhos em projeção, figuras de hipercubos ou descrições, todavia é bastante simples representar matrizes n-dimensionais, basta acrescentar novos índices à nossa matriz para aumentar seu número de dimensões.

4.3.1 Aplicações de Matrizes

As matrizes são usadas de muitas maneiras em computação.



Figura 4.23 – Símbolo associado a banco de dados
Fonte: Google Imagens (2015)

A figura anterior é um símbolo que associamos a um banco de dados que é composto por um conjunto de tabelas, porque são que matrizes bidimensionais. Estas seguem, rigorosamente, postulados matemáticos relacionados às matrizes.

Em computação gráfica, algumas das aplicações mais comuns são rotação e expansão de imagens, típicas aplicações de matrizes.

A Programação Linear e a Pesquisa Operacional utilizam, em muitas de suas soluções, matrizes. O SIMPLEX, por exemplo, é um método que resolve problemas complexos apenas baseados em matrizes. Nesse campo do conhecimento, inúmeros problemas se resumem a encontrar uma solução para um sistema linear de n equações com n incógnitas, outra típica aplicação de matrizes.

A representação de grafos é bastante comum em matrizes, sendo usadas, por exemplo, para pesquisas de rotas.

Outra aplicação bastante comum é a Criptografia – uma matriz pode ser a chave no processo de criptografia.

Se pensarmos num prédio, com salas numeradas em função dos andares que ocupam, como no esquema adiante apresentado, novamente estamos diante de uma matriz.



Um diagrama de um prédio com uma matriz de salas. O prédio é representado por uma seta curva apontando para cima e para a direita. A matriz de salas é uma tabela 2x3 com as seguintes células:

S 21	S 22	S 23
S 11	S 12	S 13

Figura 4.24 – Esquema de salas e andares em um prédio
Fonte: FIAP (2015)

4.3.2 Matrizes e a lógica algorítmica

Vamos imaginar um problema bastante simples, como a carga de uma matriz com o número 1. Chamemos a matriz de “M” e seus índices de linhas e colunas de “i” e “j”.

Ficaríamos com a seguinte situação:

```
algoritmo carga_matriz;  
variáveis  
    i, j, n : inteiro;  
    m : matriz[10][10] de inteiros;  
fim-variáveis  
início  
    imprima("Total de linhas/colunas: ");  
    n := leia();  
    para i de 0 até n-1 faça  
        para j de 0 até n-1 faça  
            m[i][j] := 1;  
        fim-para  
    fim-para  
fim
```

Figura 4.25 – Lógica algorítmica 1
Fonte: Elaborado pelo autor (2015)

É importante observar que os índices “i” e “j” possibilitam acessar posições da matriz, contudo não têm qualquer relação com ela. Em outras palavras, carregada essa matriz, poderíamos apresentar seus dados usando dois índices totalmente diferentes, como no programa a seguir:

```
algoritmo carga_mostra_matriz;  
variáveis  
    i, j, k, l, n : inteiro;  
    m : matriz[10][10] de inteiros;  
fim-variáveis  
início  
    imprima("Total de linhas/colunas: ");  
    n := leia();  
    para i de 0 até n-1 faça  
        para j de 0 até n-1 faça  
            m[i][j] := 1;  
        fim-para  
    fim-para  
    para k de 0 até n-1 faça  
        para l de 0 até n-1 faça  
            imprima(m[k][l]);  
        fim-para  
    fim-para  
fim
```

Figura 4.26 – Lógica algorítmica 2
Fonte: Elaborado pelo autor (2015)

Raramente, entretanto, isso é feito, pois não há qualquer vantagem em criarmos mais variáveis para essa tarefa. Por isso, usualmente “repetimos” o uso das variáveis, por economia de memória e clareza na codificação, mas jamais por terem os índices qualquer relação com as matrizes e vice-versa.

Note-se que nos dois algoritmos apresentados declaramos e iniciamos a matriz “M”. Vamos agora, em vez de carregar a matriz com um único número, carregá-la a partir da digitação de um pretendo usuário:

```
algoritmo le_matriz;  
variáveis  
  i, j, n : inteiro;  
  m : matriz[10][10] de inteiros;  
fim-variáveis  
início  
  para i de 0 até n-1 faça  
    para j de 0 até n-1 faça  
      imprima("Digite o elemento ",i,"x",j," : ");  
      m[i][j] := leia();  
    fim-para  
  fim-para  
fim
```

Figura 4.27 – Lógica algorítmica 3
Fonte: Elaborado pelo autor (2015)

Em nosso algoritmo, usamos os números das linhas e colunas para orientar o usuário que estivesse digitando, em nosso programa.

4.3.3 Representação de matrizes

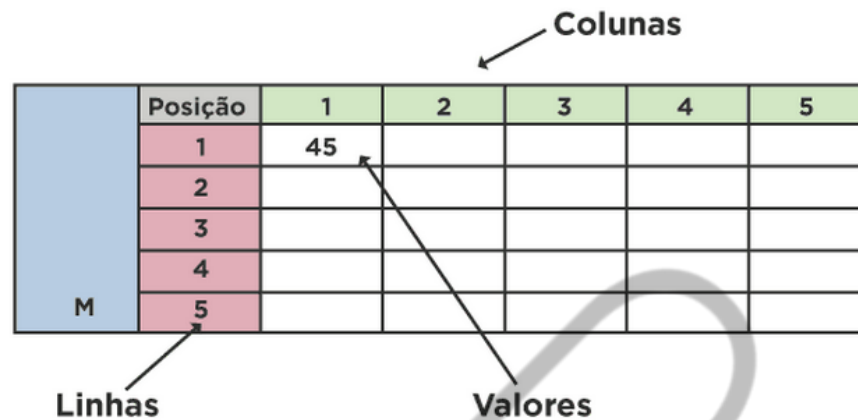
Primeiramente, precisamos **declarar** uma matriz. Essa situação varia de linguagem para linguagem de programação e mesmo entre autores de algoritmos. Vamos usar a seguinte representação de declaração para uma matriz hipotética “M” que contém 5 elementos, sendo que o seu primeiro termo está na posição 1.

M: matriz [1..5,1..5] de inteiro

Observemos agora o que ocorreria, se executássemos a seguinte operação:

M[1,1] := 5

No exemplo o número 45 será armazenado na posição linha 1 e coluna 1 da matriz. Veja a figura ilustrativa:



	Posição	1	2	3	4	5
1	1	45				
2	2					
3	3					
4	4					
5	5					

Figura 4.28 – Exemplos de operação
Fonte: FIAP (2015)

4.3.4 Exemplos

A seguir, apresentamos alguns exemplos de aplicações de matrizes.

- a) Elabore algoritmo que armazene numa matriz o valor de cada linha multiplicado por 5 somado com o número de sua coluna para uma matriz quadrada de 5 posições que começa na posição (1,1)

```
algoritmo exemplo_4_3_4_1;
variáveis
    j,k : inteiro;
    v : matriz[5][5] de inteiros;
fim-variáveis
início
    para j de 1 até 4 faça
        para k de 1 até 4 faça
            v[j][k] := j*5+k;
        fim-para
    fim-para
fim
```

Figura 4.29 – Aplicação de matriz
Fonte: Elaborado pelo autor (2015)

Como destaque nesse exemplo, devemos observar que a matriz armazena uma série de valores que dependem somente de um cálculo envolvendo o número da linha e da coluna que estiver sendo calculada.

- b) Elabore algoritmo que armazene numa matriz um valor “p” que comece com 1 e a cada linha ou coluna adicionado seja multiplicado por um valor “s”, que inicialmente vale zero e a cada iteração tem adicionado um ao seu valor.

```

algoritmo exemplo_4_3_4_2;
variáveis
    i, j, s, p : inteiro;
    v : matriz[5][5] de inteiros;
fim-variáveis
início
s := 0;
p := 1;
para i de 1 até 4 faça
    para j de 1 até 4 faça
        v[i][j] := p;
        p := s * p;
        s := s + 1;
    fim-para
fim-para
para i de 1 até 4 faça
    para j de 1 até 4 faça
        imprima(v[i][j]);
    fim-para
fim-para
fim

```

Figura 4.30 – Aplicação de matriz
Fonte: Elaborado pelo autor (2015)

4.3.5 Exercícios

- a) Simule o algoritmo apresentado a seguir:

```

1  Algoritmo MatrizENADE2008
2  variáveis
3      M[0..2][0..3], I, J, C : inteiro
4  início
5      C ← 0
6      para I ← 0 até 2 passo 1 faça
7          início
8              para J ← 0 até 3 passo 1 faça
9                  início
10                     C ← C + 1
11                     M[I][J] ← C
12                 fim para
13             fim para
14         para I ← 0 até 2 passo 1 faça
15             início
16                 para J ← 0 até 3 passo 1 faça
17                     início
18                         C ← M[2-I][3-J]
19                         M[I][J] ← C
20                     fim para
21                 fim para
22             fim para
23         fim algoritmo

```

Figura 4.31 – Simulação de algoritmo
Fonte: Elaborado pelo autor (2015)

b) Elabore algoritmo que simule o “lápiz tabuada” em uma matriz bidimensional.

- ENADE 2010. Considere uma matriz que simule um tabuleiro de xadrez (Matriz Bidimensional 8×8). Elabore algoritmo que armazene na primeira posição superior o número 1, na posição subsequente seu dobro e assim sucessivamente até a última posição do “tabuleiro”.
- Considere uma matriz A quadrada com quatro elementos e uma segunda matriz B de mesma configuração. Gere, em Português Estruturado, uma matriz C que contenha a soma das matrizes A e B.
- Construa uma matriz quadrada com 5 elementos. Considere como se estivéssemos trabalhando numa planilha eletrônica com a primeira coluna preenchida com o número de cada linha. Na segunda coluna deverá ser armazenado o dobro do número de cada linha, e assim sucessivamente. Elabore algoritmo que realize essa construção.
- Elabore programa que localize o maior elemento existente numa matriz quadrada de 3 linhas. Admita todos os elementos diferentes entre si.

REFERÊNCIAS

- FEOFILOFF, Paulo. **Algoritmos em Linguagem C**. Rio de Janeiro: Campus, 2009.
- FORBELLONE, André L.V.; EBERSPACHER, Henri F. **Construção de Algoritmos e Estruturas de Dados**. São Paulo: Pearson Prentice Hall, 2010.
- FURGERI, Sérgio. **Java 2, Ensino Didático**. São Paulo: Érica, 2002.
- GANE, Chris; SARSON, Trish. **Análise Estruturada de Sistemas**. São Paulo: LTC-Livros Técnicos e Científicos, 1983.
- GONDO, Eduardo. **Apostila: Notas de Aula**. São Paulo, 2008.
- LATORE, Robert. **Aprenda em 24 horas Estrutura de Dados e Algoritmos**. Rio de Janeiro: Campus, 1999.
- MANZANO, José A. N. G.; OLIVEIRA, Jayr F. **Algoritmos: Lógica para o Desenvolvimento de Programação**. 23. ed. São Paulo: Érica, 2010.
- PUGA, Sandra; RISSETTI, Gerson. **Lógica de Programação e Estrutura de Dados**. São Paulo: Pearson Prentice Hall, 2009.
- PIVA JUNIOR, Dilermando et al. **Algoritmos e Programação de Computadores**. Rio de Janeiro: Campus, 2012.
- ROCHA, Antonio Adrego. **Estrutura de Dados e Algoritmos em Java**. Lisboa: FCA-Editora de Informática, 2011.
- RODRIGUES, Rita. **Apostila: Notas de Aula**. 2008.
- SALVETTI, Dirceu Douglas; BARBOSA, Lisbete Madsen. **Algoritmos**. São Paulo: Makron Books, 1998.
- SCHILDT, Herbert. **Linguagem C - Guia Prático**. São Paulo: McGraw Hill, 1989.
- SCRIPTOL. **Documentation**. [s.d.]. Disponível em: <<http://www.scriptol.org/>>. Acesso em: 14 jan. 2011.
- ZIVIANI, Nivio. **Projeto de Algoritmos com implementações em Pascal e C**. São Paulo: Pioneira, 1999.
- WOOD, Steve. **Turbo Pascal – Guia do Usuário**. São Paulo: McGraw Hill, 1987.