Algoritmos II

Ricardo Ribeiro Assink
ricardo.assink@unisul.br
ricardo@equipedigital.com

http://www.ricardoassink.com.br/

Algoritmos II

Vetores e Matrizes

Vimos ser possível dar um Nome para uma posição de memória, sendo que a esta será associado um valor qualquer. Pois bem, acontece que, muitas vezes, esta forma de definição, ou melhor, de alocação de memória, não é suficiente para resolver certos problemas computacionais.

Imagine por Exemplo, como faríamos para construir um algoritmo, para ler o Nome de N Pessoas e que imprimisse um relatório destes mesmos nomes, mas ordenados alfabeticamente?

Algoritmos II

Vetores e Matrizes

Não seria uma tarefa simples, haja vista não ser possível determinar quantos nomes seriam lidos, mesmo que soubéssemos o número de pessoas, digamos 1.000 pessoas, teríamos que definir 1.000 variáveis do tipo Literal, como é mostrado abaixo:

```
public class Loucura {
public static void main(String args[]) {
String Nome1,
        Nome2,
        Nome3,
            ...
       Nome999,
       Nome1000;
<comandos> }
 }
```

Algoritmos II

Vetores e Matrizes

Variável Indexada:

Uma variável indexada corresponde a uma seqüência de posições de memória, a qual daremos único Nome, sendo que cada uma destas pode ser acessada através do que conhecemos por índice. O índice corresponde a um valor numérico (exceto Real). Cada uma das posições de memória de uma variável indexada pode receber valores no decorrer do algoritmo como se fosse uma variável comum, a única diferença reside na Sintaxe de utilização desta variável.

Tipos:

- **Variáveis Indexadas Unidimensionais (Vetor)**
- **Variáveis Indexadas Bidimensionais (Matriz)**

Algoritmos II

Vetores e Matrizes

Variável Indexada Unidimensional (Vetor):

Também conhecida por “Vetor”. Uma variável unidimensional, como o próprio Nome já indica, possui apenas uma dimensão, sendo ser possível definir variáveis com quaisquer tipo de dados.

```
public class Define {
    public static void main(String args[]) {
        <tipo de dado> <nome>[] = new <tipo de dado> [tamanho];
        <comandos>
    }
}
```

Obs.:

a)O valor “Tamanho” correspondem ao tamanho de posições do vetor

b)Uma variável indexada pode ser apenas de um tipo de dado

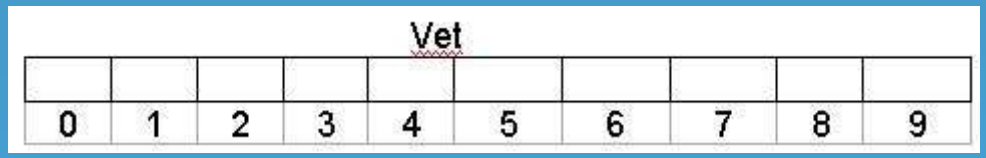
Algoritmos II

Vetores e Matrizes

Variável Indexada Unidimensional (Vetor):

Exemplo: Definir uma variável indexada como sendo do tipo REAL, sendo que a mesma deverá corresponder a 10 posições de memória.

```
public class Exemplo {
    public static void main(String args[]) {
        double vet[] = new double[10];
        <comandos>;
    } }
```



Algoritmos II

Vetores e Matrizes

Variável Indexada Unidimensional (Vetor):

Atribuição:

```
public class Define {
     public static void main(String args[]) {
    <tipo de dado> <nome>[] = new <tipo de dado> [tamanho];
    <nome>[<índice>] = valor;
     }
 }
```

Algoritmos II

Vetores e Matrizes

Variável Indexada Unidimensional (Vetor):

Exemplo nº 1:

```
public class Atribui {
      public static void main(String args[]) {
        String nomes[] = new String[20];
              nomes[0] = "João da Silva";
              nomes[1] = "Ana";
      } }
```

Exemplo nº 2:

```
public class VetorAtribuir {
       public static void main ( String args[]) {
        int X[ ] = new int[20]; // declarando e instanciando o vetor
           X [0] = 100;
           X [1] = X [0] + 3;
        } }
```

Algoritmos II

Vetores e Matrizes

Variável Indexada Unidimensional (Vetor):

Leitura

```
import javax.swing.*;
public class Leitura {
        public static void main(String args[]) {
        <tipo de dado> <nome>[] = new <tipo de dado>[tamanho];
        <nome>[<índice>] = JOptionPane.showInputDialog();
         }
 }
```

Algoritmos II

Vetores e Matrizes

Variável Indexada Unidimensional (Vetor):

Leitura - **Exemplo nº 1**

```
import javax.swing.*;
public class Exemplo1 {
    public static void main(String args[]) {
        int i;
        String nomes[] = new String[20];
        for(i = 0; i < 20; i++) {
            nomes[i] = JOptionPane.showInputDialog("Digite o nome: ");
        }
    }
}
```

Algoritmos II

Vetores e Matrizes

Variável Indexada Unidimensional (Vetor):

Leitura - **Exemplo nº 2**

```
import javax.swing.*;
public class Exemplo2 {
    public static void main ( String args[]) {
        int i;
        int n = Integer.parseInt (JOptionPane.showInputDialog( "Digite o tamanho do
vetor:"));
        int X[ ] = new int[n]; // declarando o vetor
        for (i = 0; i< n; i++ ) {
            X [i] = Integer.parseInt (JOptionPane.showInputDialog("Digite um valor:"));
        }
 } }
```

Algoritmos II

Vetores e Matrizes

Variável Indexada Unidimensional (Vetor):

Escrita:

```
import javax.swing.*;
public class Escrita {
    public static void main(String args[]) {
        <tipo de dado> <nome>[ ] = new <tipo de dado>[tamanho];
          JOptionPane.shoMessageDialog(null, <nome>[<índice>] );
    }
}
```

Algoritmos II

Vetores e Matrizes

Variável Indexada Unidimensional (Vetor):

Escrita - **Exemplo nº 3**

```
import javax.swing.*;
public class Exemplo3 {
	public static void main(String args[]) {
		 int i;
		String nomes[] = new String[20];
		nomes[0] = "Unisul";
		nomes[1] = "Aluno";
		nomes[2] = "Sistema";
		for(i = 0; i < 3; i++) {
		    JOptionPane.showMessageDialog(null, "o nome na posição "+i+ " eh "+ nomes[i]
);
		}
	}
}
```

Algoritmos II

Vetores e Matrizes

Métodos de Pesquisa

Quando se trabalha com vetores, eles poderão gerar grandes tabelas, dificultando localizar um determinado elemento de forma rápida. Imagine um vetor possuindo 4000 elementos (4000 nomes de pessoas).

Será que você conseguirá encontrar rapidamente um elemento desejado de forma manual, mesmo estando a lista de nomes em ordem alfabética? Certamente que não. Para solucionar este tipo de problema, você pode fazer pesquisas com uso de programação.

Algoritmos II

Vetores e Matrizes

Pesquisa Linear

Pesquisa Seqüencial ou Linear: consiste em efetuar a busca da informação desejada a partir do primeiro elemento seqüencialmente até o último.

Localizando a informação no caminho, ela é apresentada. Este método de pesquisa é lento, porém eficiente nos casos em que um vetor encontra-se com seus elementos desordenados.

Algoritmos II

Vetores e Matrizes

Métodos de Pesquisa – Pesquisa Linear

```
import javax.swing.*;
public class Exemplo4 //PesquisaLinearSimples
{
    public static void main(String args[])
    {
        int i, flag;
        int numElementos = Integer.parseInt(JOptionPane.showInputDialog("Digite o número de pessoas a
ser cadastrado "));
        String vetorPesquisado[] = new String[numElementos];
        for(i=0; i<numElementos;i++){
            vetorPesquisado[i] = JOptionPane.showInputDialog("Digite o nome para cadastro");
        }
        String elementoProcurado = JOptionPane.showInputDialog("Digite o nome a ser procurado");
        flag = 0;
        for(i=0; i<numElementos;i++){
            if (vetorPesquisado[i].equalsIgnoreCase(elementoProcurado)){
                JOptionPane.showMessageDialog(null,"o valor procurado foi encontrado na posição "+i);
                flag = 1;
            }
        }
        if (flag <= 0){
            JOptionPane.showMessageDialog(null, "o nome não foi encontrado.");
        }
    }
}
```

Algoritmos II

Vetores e Matrizes

Métodos de Ordenação

Os problemas de ordenação são comuns tanto em aplicações comerciais quanto científicas. Entretanto, raro são os problemas que se resumem à pura ordenação de seqüências de elementos. Normalmente, os problemas de ordenação são inseridos em problemas de pesquisa, intercalação e atualização. Isto torna ainda mais importante o projeto e a construção de algoritmos eficientes e confiáveis para tratar o problema.

Algoritmos II

Vetores e Matrizes

Bubble Sort

A estratégia utilizada pelo BubbleSort consiste de comparações e trocas entre elementos consecutivos da seqüência, a fim de "empurrar" o maior elemento para a última posição. Assim, várias iterações são efetuadas e, para cada seqüência considerada, a aplicação da estratégia gera uma nova seqüência pela eliminação do maior elemento da seqüência original. Além disto, uma variável de controle (lógica) é utilizada para registrar a ocorrência ou não de troca entre elementos da seqüência. Quando nenhuma troca é efetuada, tem-se que a seqüência considerada já estava ordenada.

Bubble Sort

Exemplo

Suponha que se deseje classificar em ordem crescente o seguinte vetor:

28	26	30	24	25

Comparamos todos os pares de elementos consecutivos, a partir do par mais à esquerda. Caso os elementos de um certo par se encontrem fora da ordem desejada, efetuamos a troca dos mesmos. O processo de comparação dos n-1 pares de elementos é denominado de varredura. O método efetuará tantas varreduras no vetor quantas forem necessárias para que todos os pares consecutivos de elementos se apresentem na ordem desejada.

Bubble Sort

Primeira Varredura:

28 | 26 | 30 | 24 | 25	compara par (28,26): troca.
26 | 28 | 30 | 24 | 25	compara par (28,30): não troca.
26 | 28 | 30 | 24 | 25	compara par (30,24): troca.
26 | 28 | 24 | 30 | 25	compara par (30,25): troca.
26 | 28 | 24 | 25 | 30	fim da primeira varredura.

Algoritmos II

Vetores e Matrizes

Bubble Sort

Como ainda existem pares não ordenados, reiniciamos o processo de comparações de pares de elementos, executando mais uma varredura. Observe, no entanto, que a primeira varredura sempre irá posicionar o elemento de maior valor na sua posição definitiva correta (no final do vetor). Isto significa que na segunda varredura podemos desconsiderar a última posição do vetor, que portanto fica reduzido de um elemento. Esta situação se repetirá ao final de cada varredura efetuada.

Bubble Sort

Segunda Varredura:

26 | 28 | 24 | 25 | 30	compara par (26,28): não troca.
26 | 28 | 24 | 25 | 30	compara par (28,24): troca.
26 | 24 | 28 | 25 | 30	compara par (28,25): troca.
26 | 24 | 25 | 28 | 30	fim da segunda varredura (o último elemento já se encontra em sua posição definitiva, não precisando ser comparado).

Bubble Sort

Terceira Varredura:

26 | 24 | 25 | 28 | 30	compara par (26,24): troca.
24 | 26 | 25 | 28 | 30	compara par (26,25): troca.
24 | 25 | 26 | 28 | 30	fim da terceira varredura (o dois últimos elemento já se encontram em sua posição definitiva, não precisando ser comparados).

Bubble Sort

Quarta Varredura:

Na quarta varredura será verificado que nenhuma troca deve ser feita, então o algoritmo é finalizado.

24 25 26 28 30

Algoritmos II

Vetores e Matrizes

Bubble Sort

Quinta Varredura:

Só existiria uma quinta varredura se na varredura anterior alguma troca fosse executada.

Bubble Sort

Examinando-se o comportamento do método, vemos que eventualmente, dependendo da disposição dos elementos, uma certa varredura pode colocar mais do que um elemento na sua posição definitiva. Isto ocorrerá sempre que houver blocos de elementos já previamente ordenados, como no exemplo abaixo.

Vetor de Elementos: 13 11 25 10 18 21 23

A primeira varredura produzirá o seguinte resultado: 11 13 10 18 21 23 25

Algoritmos II

Vetores e Matrizes

Exercícios:

Considere o vetor de elementos: 27 35 12 10 82 28 30

1 – Quantas varreduras o Bubble Sort vai executar ?

2 – Ao final da varredura 2, em que posição está o número 35 ?

3 – Ao final da varredura 6, em que posição está o número 30 ?

4 – Ao final da varredura 4, em que posição está o número 27 ?

5 – Altere o Exemplo de Bubble Sort sugerido para mostrar o vetor ao final de cada varredura.

Bubble Sort

```
import javax.swing.*;
public class Exemplo5 //BubbleSort
{
    public static void main(String args[])
    {
        int troca, fim, i, aux;
        int tamanho = Integer.parseInt(JOptionPane.showInputDialog("informe o tamanho do vetor"));
        int vetorOrdenar[] = new int[tamanho];
        // aqui vc pode atribuir valores
        troca = 1;
        fim = tamanho - 1;
        while(troca==1)
        {
            troca = 0;
            for(i=0; i<fim;i++)
            {
                if (vetorOrdenar[i] > vetorOrdenar[i+1])
                {
                    aux = vetorOrdenar[i];
                    vetorOrdenar[i] = vetorOrdenar[i+1];
                    vetorOrdenar[i+1]=aux;
                    troca = 1;
                }
            }
            fim = fim - 1;
        }
        //aqui vc pode mostrar o vetor
    }
}
```

API JAVA

http://download.oracle.com/javase/6/docs/api/

Algoritmos II

FIM

Material Original: **Osmar de Oliveira Braz Junior**