Newton

Quem se prepara, não para.

Estrutura de Dados

Professora: Michelle Hanne Soares de Andrade
michelle.andrade@newtonpaiva.br

Newton
Quem se prepara, não para.

Sumário:

- ✓ Estrutura de Dados
- ✓ Vetores
- ✓ Matriz

Estrutura de Dados

Newton
Quem se prepara, não para.

- Os tipos de dados definidos por uma linguagem de programação não são suficientes para modelar as diversas situações do ambiente dentro do código de programação, nem tudo pode ser representado como um valor inteiro (*int*) ou um valor real (*float*) ou ainda um caracter (*char*), sendo que no caso da linguagem C, não conseguimos representar nem mesmo uma palavra (conjunto de caracteres chamado de *String*) em seus tipos primitivos, sendo necessária a declaração de um vetor para armazenar uma cadeia de caracteres (*char*), formando uma *String*.

Estrutura de Dados

Newton
Quem se prepara, não para.

A criação de novos tipos de dados, denominados tipos construídos, elaborados a partir dos tipos primitivos consegue solucionar essa limitação e destaca que “esses novos tipos têm um formato **denominado estrutura de dados,** que define como os tipos primitivos estão organizados” Forbellone (2005).

Estrutura de Dados - Aplicação

Newton
Quem se prepara, não para.

- **Persistência temporária de dados**

Os programas de forma geral precisam acessar recursos externos ao programa como bancos de dados e informações de computadores e servidores locais ou através da Internet, a recuperação de dados externos à memória RAM é um limitador na velocidade do programa.

Se todas as vezes que uma informação externa for necessária o programa tiver que buscá-la, ele perderá muito tempo. No contexto apresentado as estruturas de dados vêm para possibilitar que seja feita uma persistência temporária de dados.

Estrutura de Dados - Aplicação

Newton
Quem se prepara, não para.

- **Processamento digital de imagens**

A amostragem consiste em discretizar o domínio de definição da imagem nas direções x e y, gerando uma matriz de M x N amostras, respectivamente. (...) Cada elemento f(x, y) dessa matriz de amostras é chamado pixel (acrônimo do inglês Picture element)".

Para as imagens coloridas no padrão RGB, Pedrini (2008, p. 471) destaca que passa a ser necessária uma matriz com 3 dimensões, sendo a terceira dimensão responsável pela cor do pixel que pode variar os valores de R, G e B entre valores de 0 a 255 para cada cor sendo um pixel capaz de formar mais de 16 milhões de cores.

Vetor

Um vetor é um conjunto de variáveis do mesmo tipo que podem ser referenciadas por um único identificador, chamamos de vetores as matrizes de uma dimensão, conforme define Ascencio (2005, p.1) “Um vetor é também chamado de variável composta homogênea unidimensional, ou seja, uma sequência finita de variáveis todas do mesmo tipo, com o mesmo identificador (mesmo nome) e alocadas sequencialmente na memória

Vetor

Figura 2 – Representação de um vetor

Fonte: Elaborada pelo autor.

Vetor Ordenação Simples

Newton
Quem se prepara, não para.

```
public class Ordena_Simples {

public static void main(String[] args) {
int[] vetor = new int[10];
int aux;
Scanner entrada = new Scanner(System.in);

for(int i=0;i<10;i++){
System.out.printf("\nDigite Posição #%d",i+1);
vetor[i]= entrada.nextInt();
}

//ordenação de vetor simples
for(int i=0; i<10; i++){
for(int j=0; j<9; j++){
if(vetor[j]> vetor[j+1]){
aux = vetor[j];
vetor[j] = vetor[j+1];
vetor[j+1] = aux;
}
}
}
```

```
//mostra vetor ordenado
System.out.println("Vetor Ordenado");
for(int i=0;i<10;i++){
System.out.printf("%d\t",vetor[i]);
}
}
}
```

Vetor Maior e Menor

Newton
Quem se prepara, não para.

```
import java.util.Random;

public class Maior_Menor {

public static void main(String[] args) {
int vetor[]; //declaração do vetor
vetor = new int[50];//alocação de memória

Random numrandomico = new Random(); //declara variavel do tipo randomico
int maxValue= Integer.MIN_VALUE; //atribui a variavel maxValue o menor valor de inteiro
int minValue = Integer.MAX_VALUE; //atribui a variavel minValue o maior valor de inteiro

//preenche o vetor de 50 posições com numeros randomicos
for(int i=0;i<50;i++){
vetor[i]= numrandomico.nextInt(999);
}
//descobrindo o menor e maior valor no vetor
for(int i=0;i<50;i++){
if (vetor[i]>maxValue)
maxValue=vetor[i];
if (vetor[i]<minValue)
minValue=vetor[i];
}

//exibe o maior e menor valor
System.out.println("O maior valor é " + maxValue);
System.out.println("O menor valor é " + minValue);

System.out.println("Vetor randômico:");
//exibe todo o vetor
for(int num:vetor)
System.out.println(num);

}

}
```

Busca Binária

Newton
Quem se prepara, não para.

Pesquisa Binária

Este método de pesquisa é muito mais rápido que a pesquisa sequencial, e usa como base que o vetor já está ordenado:
Dado um vetor: **[**0**,** 1**,** 2**,** 4**,** 5**,** 6**,** 7**,** 8**,** 9**]**

Encontra a posição referente ao meio do vetor, e verifica se o valor procurado é o elemento do meio do vetor. Se encontrou o valor:

•imprime uma mensagem de confirmação. Se não encontrou o valor:
•Se o valor procurado é menor que o valor que está no meio do vetor, a posição final do vetor será uma posição antes do meio do vetor.
•Se o valor procurado é maior que o valor que está no meio do vetor, a posição inicial do vetor será uma posição depois do meio do vetor.
•Volta para o passo 1.

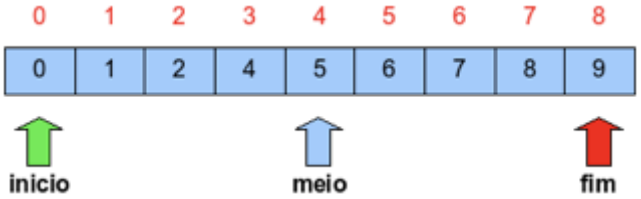
Fonte: http://www.universidadejava.com.br/pesquisa_ordenacao/pesquisa-binaria/

Busca Binária

Newton
Quem se prepara, não para.

Procurando o valor 4 dentro do vetor

- O valor da posição inicial é 0 (zero).
- O valor da posição final é 8.
- O valor da posição meio é 4. (A posição do meio é igual ao (inicio + fim) / 2)
- Verifica se o valor do vetor que está na posição do meio é igual ao valor procurado, neste caso o valor do meio do vetor é 5 e estamos procurando o valor 4.

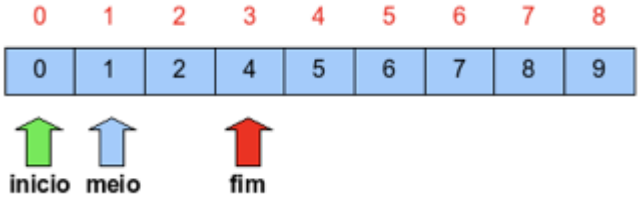


Fonte: http://www.universidadejava.com.br/pesquisa_ordenacao/pesquisa-binaria/

Busca Binária

Como o valor que estamos procurando é menor que o valor que está no meio do vetor, então mudamos o valor da posição final para 3. O valor da posição inicial continua sendo 0 (zero). Agora o valor da posição meio muda 1.

Verifica novamente se o valor do vetor na posição do meio é igual ao valor procurado, neste caso o valor do meio do vetor é 1 e estamos procurando o valor 4.

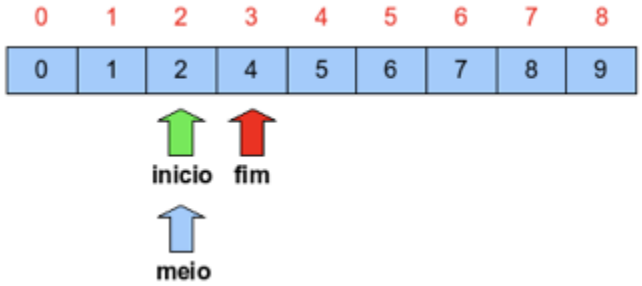


Fonte: http://www.universidadejava.com.br/pesquisa_ordenacao/pesquisa-binaria/

Busca Binária

Como o valor que estamos procurando é maior que o valor que está no meio do vetor, então mudamos o valor da posição inicial para 2. O valor da posição final continua sendo 3. Agora o valor da posição meio muda 2.

Verifica novamente se o valor do vetor na posição do meio é igual ao valor procurado, neste caso o valor do meio do vetor é 2 e estamos procurando o valor 4.

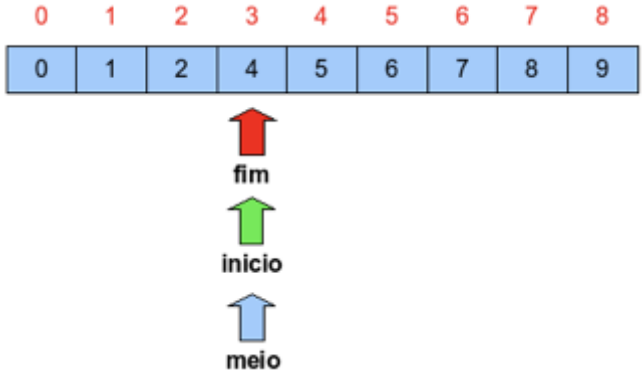


Fonte: http://www.universidadejava.com.br/pesquisa_ordenacao/pesquisa-binaria/

Busca Binária

Como o valor que estamos procurando é maior que o valor que está no meio do vetor, então mudamos o valor da posição inicial para 3. O valor da posição final continua sendo 3. Agora o valor da posição meio muda 3.

Verifica novamente se o valor do vetor na posição do meio é igual ao valor procurado, neste caso o valor do meio do vetor é 4 e encontramos o valor que estávamos procurando.



Fonte: http://www.universidadejava.com.br/pesquisa_ordenacao/pesquisa-binaria/

Busca Binária

Newton
Quem se prepara, não para.

```
public class PesquisaBinaria {

public static void main(String[] args) {
  PesquisaBinaria bin = new PesquisaBinaria();

    int[] numeros = {1, 3, 4, 7, 9, 10, 13, 18, 20, 21, 22};

    bin.pesquisarNumero(20, numeros);
   // bin.pesquisarNumero(3, numeros);
  }

  /**
   * Método que pesquisa um número dentro de um vetor de números.
   * Este método utiliza o algoritmo de pesquisa binaria.
   *
   * @param x    - Número que será pesquisado.
   * @param numeros - Vetor de números.
   */
  public void pesquisarNumero(int x, int[] numeros) {
    int inicio = 0;          //Posição inicial do vetor.
    int meio = 0;           //Posição do meio do vetor.
    int fim = numeros.length - 1;  //Posição final do vetor.
```

Busca Binária

Newton
Quem se prepara, não para.

```
/* Enquanto a posição do inicio for menor ou igual a posição do fim,
      procura o valor de x dentro do vetor. */
    while(inicio <= fim) {
      meio = (fim + inicio) / 2; //Encontra o meio do vetor.

      System.out.println("Inicio: " + numeros[inicio] + " - Meio: " + numeros[meio] + " - Fim: " +
numeros[fim]);

      /* Se o valor que está no meio do vetor é igual ao valor procurado,
        imprime que encontrou o valor e para de pesquisar. */
      if(numeros[meio] == x) {
        System.out.println("Encontrou o número " + x);
        break;
      }

      /* Este if serve para diminuir o tamanho do vetor pela métade. */
      /* Se o valor que está no meio do vetor for menor que o valor de x,
        então o inicio do vetor será igual a posição do meio + 1. */
      if(numeros[meio] < x) {
        inicio = meio + 1;
      } else {
        /* Se o valor que está no meio do vetor for maior que o valor de x,
          então o fim do vetor será igual a posição do meio - 1. */
        fim = meio - 1;
      }
    }
```

Fonte: http://www.universidadejava.com.br/pesquisa_ordenacao/pesquisa-binaria/

Busca Binária

Newton
Quem se prepara, não para.

```
 /* Caso não encontre o valor pesquisado dentro do vetor,
      imprime que não encontrou o valor pesquisado. */
    if(inicio > fim) {
      System.out.println("Não encontrou o número " + x);
    }
  }
}
```

Newton
Quem se prepara, não para.

Armazenamento de dados homogêneos em duas ou mais dimensões

Matriz

Newton
Quem se prepara, não para.

```
public class ExemploMatriz {

public static void main(String[] args) {
int[][] matriz = new int[50][50]; // inicializa a matriz com zeros
//loop para preencher e mostrar a diagonal
int impar=1;
for(int linhas=0;linhas<50;linhas++){
for(int cols=0;cols<50;cols++){
matriz[linhas][cols]=impar;
if (linhas==cols)
System.out.printf("%d\t",matriz[linhas][cols]);
else
System.out.printf("0\t");
impar+=2;
}
System.out.printf("\n");
}

}

}
```

Referências

FORBELLONE, André Luiz Villar; EBERSPACHER, Henri Frederico. **Lógica de programação**: a construção de algoritmos e estruturas de dados. 3. ed. São Paulo: Prentice Hall, 2005.