**1 - Busca Simples**

A busca simples, também conhecida como busca sequencial, é um algoritmo básico para encontrar um elemento específico dentro de um vetor. Na busca simples, cada elemento é verificado sequencialmente até que o elemento desejado seja encontrado ou até que todos os elementos tenham sido examinados.

O algoritmo de busca simples possui uma complexidade, no pior caso, de O(n). Ou seja, se nosso vetor tem n elementos e o item buscado não existe no vetor, esse algoritmo de busca simples irá iterar sobre todos itens do vetor para somente depois concluir que ele não foi encontrado. O mesmo ocorreria se o item buscado fosse o último do vetor.

No bloco 1.1 temos um método para implementação da busca simples, em Java. Este método deve ser implementado dentro da classe MeuVetor, utilizada nas aulas anteriores.

Bloco 1.1

|  |
| --- |
| // Método para realizar busca simples em um vetor de inteiros  public int buscaSimples(int valorPesquisado) {      // Percorre o vetor      for (int i = 0; i < v.length; i++) {          /\* Se o elemento na posição tual é igual à valorPesquisado, retorne a posição do vetor em que ele foi encontrado. \*/          if (v[i] == valorPesquisado) {              return i;          }      }      // Se o valor pesquisado não for encontrado, retorna -1.      return -1;  } |

Para testar o método buscaSimples(), incrementaremos a classe TesteRandomico, criada na aula anterior, com o código destacado em amarelo no bloco 1.2.

Bloco 1.2

|  |
| --- |
| import java.util.Scanner;  public class TesteRandomico {      public static void main(String[] args) {          MeuVetor v1 = new MeuVetor(60);          MeuVetor v2 = new MeuVetor(60);            v1.preencheVetor();          System.out.println(v1);          v2.preencheVetor(200);          System.out.println(v2);          Scanner sc = new Scanner(System.in);          System.out.print("digite um valor para a busca simples: ");          int valorPesquisado = sc.nextInt();          int posicao = v1.buscaSimples(valorPesquisado);          if(posicao != -1){              System.out.println("Valor encontrado na posição de indice " + posicao);          }else{              System.out.println("Valor não encontrado no vetor.");          }      }  } |

**2 - Busca Binária**

Quando estamos lidando com um **vetor ordenado**, podemos implementar um algoritmo de busca mais eficiente do que a busca simples/sequencial. Esse algoritmo é a busca binária, que funciona dividindo repetidamente o espaço de busca pela metade até encontrar o elemento desejado.

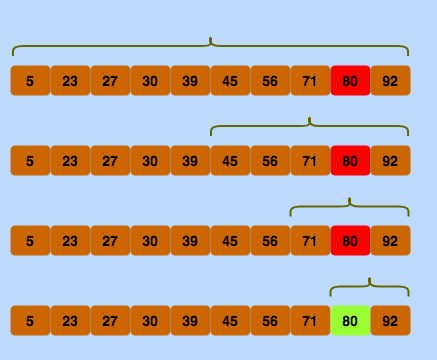
Para entendermos o algoritmo de busca binária podemos realizar uma analogia do problema: imagine que você precisa buscar o significado de uma palavra na língua portuguesa que comece com a letra H. Considere então que você pega um dicionário, que tipicamente estará ordenado pelas letras de nosso alfabeto, e o abra ao meio.

A página aberta está na letra M, por exemplo. Nossa palavra começa com H e H é menor do que M na ordem do alfabeto da língua portuguesa. Logo, sua intuição é abrir novamente ao meio as páginas à esquerda de M. Vamos supor que agora você caiu na letra E. Como H é posterior a E nessa ordenação, você agora divide novamente as páginas à direita de E. Esse processo se repetirá até que seja encontrada a letra H, que é o elemento que buscamos neste cenário.

Essa analogia exemplifica bem o paradigma Dividir para Conquistar (ou *Divide and Conquer)*. Essa técnica consiste em dividir um problema em problemas menores (sub-problemas). No nosso caso, o algoritmo de busca binária irá dividir/reduzir pela metade o espaço de busca à cada iteração. Exatamente como o exemplo do dicionário.

A figura 2.1 exemplifica o funcionamento de uma busca binária. A complexidade do algoritmo de busca binária se da por O(log n).

Figura 2.1



No bloco 2.1, temos a implementação em código do método buscaBinaria(). Também incluiremos este método na classe MeuVetor.

Bloco 2.1

|  |
| --- |
| public int buscaBinaria(int valorPesquisado) {      int inicio = 0, fim = ultimaPos;        // Enquanto o elemento no inicio do vetor for menor ou igual      // ao elemento no final do vetor, faça      while (inicio <= fim) {          int meio = (inicio + fim) / 2;            // Se o valorPesquisado estiver em v[meio], retorna o valor          if (v[meio] == valorPesquisado) {              return valorPesquisado;          }            // Se valorPesquisado > v[meio], descarta a metade esquerda          // atualizando o valor da posição inicial do vetor          if (valorPesquisado > v[meio]) {              inicio = meio + 1;          }          // Se valorPesquisado <= v[meio]r, descarta a metade direita          // atualizando o valor da posição final do vetor          else {              fim = meio - 1;          }      }        // Se o valorPesquisado não for encontrado, retorna -1      return -1;  } |

Para testar o método buscaBinaria(), incrementaremos a classe TesteRandomico com o código destacado em amarelo no bloco 2.2.

Bloco 2.2

|  |
| --- |
| import java.util.Scanner;  public class TesteRandomico {      public static void main(String[] args) {          MeuVetor v1 = new MeuVetor(60);          MeuVetor v2 = new MeuVetor(60);            v1.preencheVetor();          System.out.println(v1);          v2.preencheVetor(200);          System.out.println(v2);          Scanner sc = new Scanner(System.in);          System.out.print("digite um valor para busca: ");          int valorPesquisado = sc.nextInt();          int posicao = v1.buscaSimples(valorPesquisado);          if(posicao != -1){              System.out.println("Valor encontrado na posição de indice " + posicao);          }else{              System.out.println("Valor não encontrado no vetor.");          }  //Para que a busca binária funcione, o vetor deve estar ordenado.          v2.bubbleSort();          System.out.print("digite um valor para busca binária: ");          valorPesquisado = sc.nextInt();          int x = v2.buscaBinaria(valorPesquisado);          if(x != -1){              System.out.println("Valor encontrado: " + x);          }else{              System.out.println("Valor não encontrado no vetor.");          }      }  } |

**3 – Testes com Busca Simples e Busca Binária**

Para testar de forma mais detalhada ambos os algoritmos de busca, utilizaremos uma classe auxiliar chamada **Retorno**. Nesta classe, o atributo “contador” será utilizado para efetuar a contagem de quantas verificações o algoritmo de busca precisou realizar para encontrar o elemento pesquisado. O atributo “achou” retornará um valor booleano verdadeiro se o elemento pesquisado existe dentro do vetor e um valor booleano falso se não existe. O código para criação desta classe está disponível abaixo no bloco 3.1.

Bloco 3.1

|  |
| --- |
| public class Retorno {      private int contador;      private boolean achou;      // getters      public int getContador() {          return contador;      }      public boolean getAchou () {          return achou;      }        // incrementa o valor do atributo contador      // contador = contador + 1      public void incrementaContador() {          contador++;      }      // set para o atributo achou      public void setAchou(boolean achou) {          this.achou = achou;      }  } |

Em seguida, atualizaremos os métodos **buscaSimples()** e **buscaBinaria()**, da classe **MeuVetor**, utilizando um objeto da classe **Retorno**. Os blocos 3.2 e 3.3 mostram os métodos de busca simples e binária atualizados, respectivamente. As linhas destacadas em amarelo mostram o que foi atualizado no código.

Bloco 3.2

|  |
| --- |
| public Retorno buscaSimples(int valorPesquisado) {      Retorno r = new Retorno();      for (int i = 0; i < v.length; i++) {          r.incrementaContador();          // Se o valorPesquisado estiver em v[i], retorna o objeto r  // com o atributo achou = true          if (v[i] == valorPesquisado) {             r.setAchou(true);             return r;          }      }      // Se o valorPesquisado não for encontrado, retorna r sem atualizar  // o atributo achou para true.      return r;  } |

Bloco 3.3

|  |
| --- |
| public Retorno buscaBinaria(int valorPesquisado) {      int inicio = 0, fim = ultimaPos;      Retorno r = new Retorno();      while (inicio <= fim) {          int meio = (inicio + fim) / 2;          r.incrementaContador();          // Se o valorPesquisado estiver em v[meio], retorna o objeto r  // com o atributo achou = true          if (v[meio] == valorPesquisado) {  r.setAchou(true);              return r;          }            if (valorPesquisado > v[meio]) {              inicio = meio + 1;          }  else {              fim = meio - 1;          }      }        // Se o valorPesquisado não for encontrado, retorna r sem atualizar  // o atributo achou para true.      return r;  } |

Para testar os métodos atualizados com o objeto da classe **Retorno**, criaremos uma nova classe de teste chamada **OrdenaBusca**. Nesta classe, programaremos os seguintes itens:

1. Criar um vetor (objeto da classe MeuVetor) e preencher este vetor com valores randômicos;
2. Fazer a leitura de um valor digitado pelo usuário, valor que será posteriormente pesquisado dentro do vetor utilizando os métodos de busca simples e binária;
3. Aplicar o método de busca simples e imprimir quantos testes foram realizados pela busca;
4. Aplicar o método de busca binária e imprimir quantos testes foram realizados pela busca.

O bloco 3.4 mostra todo o código de implementação da classe **OrdenaBusca**.

Bloco 3.4

|  |
| --- |
| import java.util.Scanner;  public class OrdenaBusca {      public static void main(String[] args) {    // Cria, preenche com valores random e imprime o vetor  MeuVetor v1 = new MeuVetor(120000);          v1.preencheVetor();          System.out.println("vetor gerado:\n" + v1);    // Faz a leitura de um valor digitado pelo usuário          Scanner sc = new Scanner(System.in);          System.out.print("digite um valor para busca: ");          int x = sc.nextInt();  // Aplicação da busca simples          Retorno r = v1.buscaSimples(x);          if (r.getAchou()){              System.out.println(x + " encontrado pela busca simples");          }          else {              System.out.println(x + " nao encontrado pela busca simples");          }          System.out.println("foram realizados " + r.getContador() + " testes na busca simples");    // Ordena o vetor utilizando o algoritmo bubble sort          v1.bubbleSort();  // Aplicação da busca binária          r = v1.buscaBinaria(x);          System.out.println();          if (r.getAchou()){              System.out.println(x + " encontrado pela busca binaria");          }          else {              System.out.println(x + " nao encontrado pela busca binaria");          }          System.out.println("foram realizados " + r.getContador() + " testes pela busca binaria");  // Desaloca o espaço de memória utilizado pelo objeto Scanner          sc.close();      }  } |