Algoritmos e Estruturas de Dados

Algoritmos de Pesquisa

Autores:
Carlos Urbano
Catarina Reis
José Magno
Marco Ferreira

Algoritmos de Pesquisa

- Um algoritmo de pesquisa permite procurar um elemento numa sequência (ordenada ou não ordenada), devolvendo:
  - a posição onde se encontra na sequência

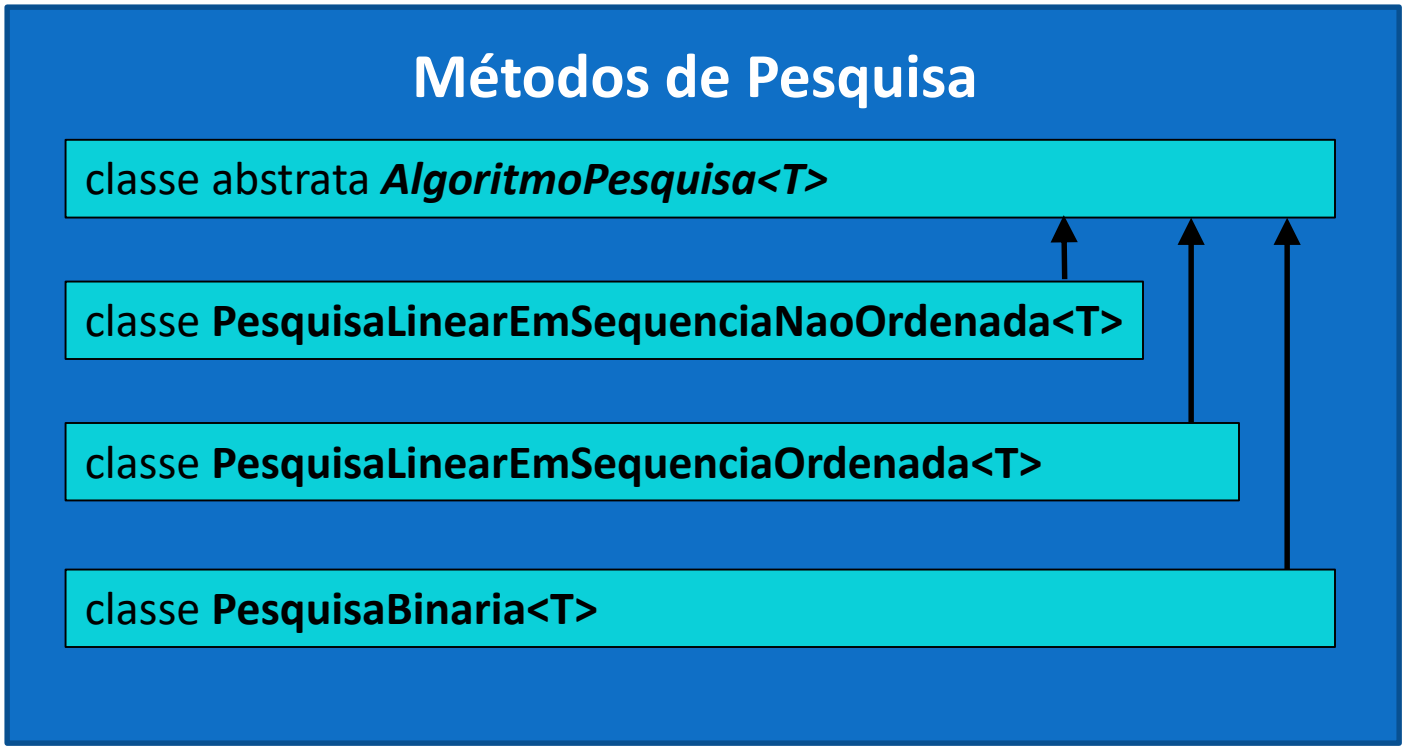
    ou
  - a indicação de que o elemento não se encontra na sequência

Algoritmos de Pesquisa

- Pesquisa sobre sequências não ordenadas
  - Pesquisa Linear
- Pesquisa sobre sequências ordenadas
  - Pesquisa Linear
  - Pesquisa Binária

Algoritmos de Pesquisa

- Classes a construir de seguida



Algoritmos de Pesquisa

- Classes a construir de seguida

Utilização

classe **MainTeoricaPesquisaLinearEmSequenciaNaoOrdenada**

classe **MainTeoricaPesquisaLinearEmSequenciaOrdenada**

classe **MainTeoricaPesquisaBinaria**

classe **MainTeoricaComparacaoAlgoritmosPesquisaEmSequenciaOrdenada**

Algoritmos de Pesquisa

- O código comum entre os vários algoritmos de pesquisa é definido pela classe AlgoritmoPesquisa

```
public abstract class AlgoritmoPesquisa<T> {
    public static final int NAO_ENCONTRADO = -1;

    protected final Comparacao<T> criterio;

    public AlgoritmoPesquisa(Comparacao<T> criterio) {
        this.criterio = criterio;
    }

    public abstract int pesquisar(EstatisticaDeComparacoes estatistica, T elemento,
                                  T... elementos);

    public EstatisticaDeComparacoes getEstatistica(T elemento, T... elementos) {
        EstatisticaDeComparacoes estatistica =
                                      new EstatisticaDeComparacoes(elementos.length);
        int indice = pesquisar(estatistica, elemento, elementos);
        estatistica.parar();
        System.out.print(elemento + " pesquisado por " + getClass().getSimpleName()
                + (indice != NAO_ENCONTRADO ? (" encontrado no índice " + indice) :
                                      " não encontrado") + " : ");
        Vetor.apresentar(10, elementos);
        estatistica.apresentar();
        return estatistica;
    }
}
```

Algoritmos de Pesquisa
Sequências não ordenadas - Pesquisa Linear

- O algoritmo consiste em dividir uma sequência em duas partes:
  - a subsequência pesquisada (parte esquerda, inicialmente vazia)
  - a subsequência por pesquisar (parte direita)
- Assim, em cada iteração, o algoritmo **compara** o elemento a procurar com o 1º elemento da subsequência por pesquisar
  - Caso sejam iguais devolve a posição desse 1º elemento (terminando o algoritmo)
  - Caso contrário, continua a pesquisar a subsequência restante
- O algoritmo termina, igualmente, quando a subsequência por pesquisar for vazia devolvendo, neste caso, uma indicação de que o elemento a procurar não se encontra na sequência

Algoritmos de Pesquisa
Sequências não ordenadas - Pesquisa Linear

```
public class PesquisaLinearEmSequenciaNaoOrdenada<T>
                                                    extends AlgoritmoPesquisa<T> {

    public PesquisaLinearEmSequenciaNaoOrdenada(Comparacao<T> criterio) {
        super(criterio);
    }

    public int pesquisar(EstatisticaDeComparacoes estatistica, T elemento,
                         T... elementos) {
        for (int i = 0; i < elementos.length; i++) {
            estatistica.incrementarComparacoes();
            if (criterio.comparar(elemento, elementos[i]) == 0) {
                return i;
            }
        }
        return NAO_ENCONTRADO;
    }
}
```

Algoritmo de ordem O(N)

Algoritmos de Pesquisa
Sequências não ordenadas - Pesquisa Linear

```
public class MainTeoricaPesquisaLinearEmSequenciaNaoOrdenada {

    public MainTeoricaPesquisaLinearEmSequenciaNaoOrdenada() {
        PesquisaLinearEmSequenciaNaoOrdenada<Integer> pesquisaLinear =
            new PesquisaLinearEmSequenciaNaoOrdenada<>(ComparacaoInteiros.CRITERIO);
        pesquisaLinear.getEstatistica(1, 7, 2, 5, 4, 1, 6, 8, 9);
        pesquisaLinear.getEstatistica(3, 7, 2, 5, 4, 1, 6, 8, 9);
        pesquisaLinear.getEstatistica(3, 1, 2, 4, 5, 6, 7, 8, 9);
    }

    public static void main(String[] args) {
        new MainTeoricaPesquisaLinearEmSequenciaNaoOrdenada();
    }
}
```

Algoritmos de Pesquisa
Sequências não ordenadas - Pesquisa Linear

Algoritmos de Pesquisa
Sequências ordenadas - Pesquisa Linear

- O algoritmo consiste em dividir uma sequência em duas partes:
  - a subsequência pesquisada (parte esquerda, inicialmente vazia)
  - a subsequência por pesquisar (parte direita)
- Assim, em cada iteração, o algoritmo **compara** o elemento a procurar com o 1º elemento da subsequência por pesquisar
  - Caso sejam iguais devolve a posição desse 1º elemento (terminando o algoritmo)
  - Caso seja de menor ordem devolve uma indicação que o elemento a procurar não se encontra na sequência (terminando o algoritmo)
  - Em qualquer outro caso, continua a pesquisar a subsequência restante
- O algoritmo termina, igualmente, quando a subsequência por pesquisar for vazia ou o elemento a procurar for de ordem superior à do último elemento da sequência, devolvendo, neste caso, uma indicação de que o elemento a procurar não se encontra na sequência

Algoritmos de Pesquisa
Sequências ordenadas - Pesquisa Linear

```
public class PesquisaLinearEmSequenciaOrdenada<T> extends AlgoritmoPesquisa<T> {

    public PesquisaLinearEmSequenciaOrdenada(Comparacao<T> criterio) {
        super(criterio);
    }

    public int pesquisar(EstatisticaDeComparacoes estatistica, T elemento,
                         T... elementos) {
        if (elementos.length == 0 ||
            criterio.comparar(elemento, elementos[elementos.length - 1]) > 0) {
                return NAO_ENCONTRADO;
        }

        for (int i = 0; i < elementos.length; i++) {
            estatistica.incrementarComparacoes();
            int cp = criterio.comparar(elemento, elementos[i]);
            if (cp < 0) {
                return NAO_ENCONTRADO;
            }
            estatistica.incrementarComparacoes();
            if (cp == 0) {
                return i;
            }
        }
        return NAO_ENCONTRADO;
    }
}
```

Algoritmo de ordem O(N)

Algoritmos de Pesquisa
Sequências ordenadas - Pesquisa Linear

```
public class MainTeoricaPesquisaLinearEmSequenciaOrdenada {

    public MainTeoricaPesquisaLinearEmSequenciaOrdenada() {
        PesquisaLinearEmSequenciaOrdenada<Integer> pesquisaLinear =
                new PesquisaLinearEmSequenciaOrdenada<>(ComparacaoInteiros.CRITERIO);
        pesquisaLinear.getEstatistica(7, 1, 2, 4, 5, 6, 7, 8, 9);
        pesquisaLinear.getEstatistica(3, 1, 2, 4, 5, 6, 7, 8, 9);
    }

    public static void main(String[] args) {
        new MainTeoricaPesquisaLinearEmSequenciaOrdenada();
    }
}
```

Algoritmos de Pesquisa
Sequências ordenadas - Pesquisa Linear

Algoritmos de Pesquisa
Sequências ordenadas - Pesquisa Binária

- Caso uma sequência esteja ordenada podemos efetuar a pesquisa binária, que consiste em, repetidamente, escolher a posição do meio da (sub)sequência e **comparar** o elemento dessa posição com o elemento a procurar:
  - Se o elemento a procurar tiver ordem inferior, considera na próxima iteração apenas a procura do elemento na subsequência esquerda
  - Se o elemento a procurar tiver ordem superior, considera na próxima iteração apenas a procura do elemento na subsequência direita
  - Se for igual devolve a posição na sequência (terminando o algoritmo)
- O algoritmo termina, igualmente, quando a subsequência por pesquisar for vazia ou o elemento a procurar for de ordem inferior à do primeiro elemento da sequência ou de ordem superior à do último elemento da sequência, devolvendo, neste caso, uma indicação de que o elemento a procurar não pertence à sequência

Algoritmos de Pesquisa
Sequências ordenadas - Pesquisa Binária

```
public class PesquisaBinaria<T> extends AlgoritmoPesquisa<T> {

    public PesquisaBinaria(Comparacao<T> criterio) {
        super(criterio);
    }

    public int pesquisar(EstatisticaDeComparacoes estatistica, T elemento,
                         T... elementos) {
        int indiceUltimoElemento = elementos.length - 1;
        if (elementos.length == 0 ||
                criterio.comparar(elemento, elementos[0]) < 0 ||
                criterio.comparar(elemento, elementos[indiceUltimoElemento]) > 0) {
            return NAO_ENCONTRADO;
        }
        return pesquisarRecursivo(estatistica, elemento, 0, indiceUltimoElemento,
                                  elementos);
    }
```

Algoritmos de Pesquisa
Sequências ordenadas - Pesquisa Binária

```
    public int pesquisarRecursivo(EstatisticaDeComparacoes estatistica, T elemento,
                                  int esq, int dir, T... elementos) {
        if (esq > dir) {
            return NAO_ENCONTRADO;
        }
        int meio = (esq + dir) / 2;
        int cp = criterio.comparar(elemento, elementos[meio]);
        if (cp > 0) {
            return pesquisarRecursivo(estatistica, elemento, meio + 1, dir,
                                      elementos);
        }
        if (cp < 0) {
            return pesquisarRecursivo(estatistica, elemento, esq, meio - 1,
                                      elementos);
        }
        return meio;
    }
}
```

Algoritmo de ordem
$O(\log_2 N)$

Algoritmos de Pesquisa
Sequências ordenadas - Pesquisa Binária

```
public class MainTeoricaPesquisaBinaria {

    public MainTeoricaPesquisaBinaria() {
        PesquisaBinaria<Integer> pesquisaBinaria =
                                    new PesquisaBinaria<>(ComparacaoInteiros.CRITERIO);
        pesquisaBinaria.getEstatistica(7, 7);
        pesquisaBinaria.getEstatistica(7, 1, 2, 4, 5, 6, 7, 8, 9);
        pesquisaBinaria.getEstatistica(3, 1, 2, 4, 5, 6, 7, 8, 9);
    }

    public static void main(String[] args) {
        new MainTeoricaPesquisaBinaria();
    }
}
```

Algoritmos de Pesquisa
Sequências ordenadas - Pesquisa Binária

Algoritmos de Pesquisa

- Análise comparativa dos algoritmos em sequências ordenadas

```
public class MainTeoricaComparacaoAlgoritmosPesquisaEmSequenciaOrdenada {
    private static final int TAMANHO = 1000;
    private static final int NUMERO_EXECUCOES = 20;

    public MainTeoricaComparacaoAlgoritmosPesquisaEmSequenciaOrdenada() {
        VisualizadorEstatisticas v = new VisualizadorEstatisticas();
        v.adicionarEstatisticas("Pesquisa Linear", getEstatisticas(
            new PesquisaLinearEmSequenciaOrdenada<>(ComparacaoInteiros.CRITERIO)));
        v.adicionarEstatisticas("Pesquisa Binária", getEstatisticas(
                                new PesquisaBinaria<>(ComparacaoInteiros.CRITERIO)));
        v.visualizar();
    }

    public static void main(String[] args) {
        new MainTeoricaComparacaoAlgoritmosPesquisaEmSequenciaOrdenada();
    }
```

Algoritmos de Pesquisa

```
    private List<Estatistica> getEstatisticas(AlgoritmoPesquisa<Integer> algoritmo) {
        List<Estatistica> estatisticas = new ArrayList<>();
        Random random = new Random();
        for (int i = 1; i <= NUMERO_EXECUCOES; i++) {
            Integer[] elementos = VetorDeInteiros.criarAleatorioInteger(
                                        TAMANHO * i, -TAMANHO * 10, TAMANHO * 10, true);
            new QuickSort<>(ComparacaoInteiros.CRITERIO).getEstatistica(elementos);
            EstatisticaDeComparacoes estatistica = algoritmo.getEstatistica(
                                random.nextInt(TAMANHO * 30) - TAMANHO * 15, elementos);
            estatisticas.add(estatistica);
        }
        return estatisticas;
    }
}
```

Algoritmos de Pesquisa