Estruturas de Dados em Java

por

Prof. Dr. Paulo Roberto Gomes Luzzardi
e-mail: pluzzardi@gmail.com

Home Page: http://infovis.ucpel.tche.br/luzzardi
http://graphs.ucpel.tche.br/luzzardi

Versão 1.08
13/04/2010

Referências Bibliográficas

CORMEN, et al. **Algoritmos - Teoria e Prática**. Rio de Janeiro: Campus, 2002.
VELOSO, Paulo e SANTOS, Clésio - **Estruturas de Dados** - Editora Campus, 4 ed., Rio de Janeiro, 1986.
WIRTH, Niklaus. **Algoritmos e Estruturas de Dados**. Rio de Janeiro: Prentice-Hall do Brasil, 1989.
PINTO, Wilson - **Introdução ao Desenvolvimento de Algoritmos e Estrutura de Dados**, Editora Érica, 1994.
GOODRICH, Michael T. and **TAMASSIA**, Roberto – Estruturas de Dados e Algoritmos em Java, Bookman, 2002.

Pelotas, 13 de abril de 2010 (18:56:53am)

Sumário

1. Tipos de Dados 1
1.1 Conceitos Básicos 1
1.1.1 Estruturas de Dados 1
1.1.2 Organização de Arquivos 1
1.1.3 Conceito 1
1.1.4 Dados 1
1.1.5 Tipos de Dados 1
1.2 Tipos Primitivos 2
1.2.1 Tipos primitivos da Linguagem de Programação Java 2
1.3 Construção de Tipos (Estruturados ou Complexos) 3
1.3.1 String (Cadeia de Caracteres) 3
1.3.2 Vetor e Matriz (Agregados Homogêneos) 4
1.3.3 Classe (class) 5
1.4 Operadores (Aritméticos, Relacionais e Lógicos) 6
1.4.1 Aritméticos 6
1.4.2 Relacionais 6
1.4.3 Lógicos 6
2. Vetores e Matrizes 7
2.1 Conceitos Básicos 7
2.2 Exercícios com Vetores e Matrizes 8
2.3 Alocação de Memória (RAM - Random Access Memory) 9
2.3.1 Alocação Estática de Memória 9
2.3.2 Alocação Dinâmica de Memória 9
2.3.3 Célula, Nodo ou Nó 9
3. Listas Lineares 10
3.1 Listas Genéricas 10
3.2 Tipos de Representações 12
3.2.1 Lista Representada por Contiguidade Física 12
3.2.2 Lista representada por Encadeamento 29
3.2.3 Lista Encadeada com Descritor 37
3.2.4 Lista Duplamente Encadeada 43
3.2.5 Listas Lineares com Disciplina de Acesso 49
3.2.5.1 Filas 49
3.2.5.1.1 Fila com Vetor 50
3.2.5.1.2 Fila Circular 53
3.2.5.1.3 Fila com Alocação Dinâmica 56
3.2.5.2 Pilhas 58
3.2.5.2.1 Pilha com Vetor 59
3.2.5.2.2 Pilha com Alocação Dinâmica 62
3.2.5.3 Deque ("*Double-Ended QUEue*") 64
4. Pesquisa de Dados 69
4.1 Pesquisa Seqüencial 69
4.2 Pesquisa Binária 73
4.3 Cálculo de Endereço (*Hashing*) 76
5. Classificação de Dados (Ordenação) 80
5.1 Classificação por Força Bruta 80
5.2 Vetor Indireto de Ordenação (Tabela de Índices) 81
5.3 Classificação por Encadeamento 82
5.4 Métodos de Classificação Interna 85
5.4.1 Método por Inserção Direta 85
5.4.2 Método por Troca 88
5.4.2.1 Método da Bolha (*Bubble Sort*) 88
5.4.3 Método por Seleção 89
5.4.3.1 Método por Seleção Direta 89

6. Árvores 92
6.1 Conceitos Básicos 92
6.2 Árvores Binárias 95
6.3 Representações 97
6.3.1 Representação por Contigüidade Física (Adjacência) 97
6.3.2 Representação por Encadeamento 99
6.4 Caminhamento em Árvores 99
6.4.1 Caminhamento Pré-Fixado (Pré-Ordem) 99
6.4.2 Caminhamento In-Fixado (Central) 100
6.4.3 Caminhamento Pós-Fixado 100
6.4.4 Algoritmos recursivos para percorrer Árvores Binárias 101
6.4.4.1 Caminhamento Pré-Fixado (Pré-Ordem) 102
6.4.4.2 Caminhamento In-Fixado (Central) 102
6.4.4.3 Caminhamento Pós-Fixado 102
6.5 Árvore de Busca Binária 104
6.6 Árvore AVL 106
6.6.1 Inserção em uma árvore AVL 108
6.6.2 Remoção em uma árvore AVL 114
7. Grafos 117
7.1 Conceitos 117
7.2 Representação por Lista e Matriz de Adjacências 124
7.2.1 Lista de Adjacências 124
7.2.2 Matriz de Adjacências 125
7.3 Percurso em Amplitude e Percurso em Profundidade 126
7.4 Determinação do Caminho Mínimo 127
7.4.1 Algoritmo de Dijkstra 127
8. Programas exemplos 134
8.1 Torre de Hanoi (Pilha - *Stack*) 134
8.2 Analisador de Expressões (Pilha - *Stack*) 139

1. Tipos de Dados

1.1 Conceitos Básicos

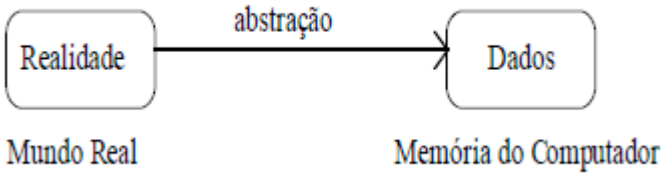
1.1.1 Estruturas de Dados

Estuda as principais técnicas de representação e manipulação de dados na **memória principal** (Memória de Acesso Randômico, RAM – ***R****andom* ***A****ccess* ***M****emory*).

1.1.2 Organização de Arquivos

Estuda as principais técnicas de representação e manipulação de dados na **memória secundária** (Disco).

1.1.3 Conceito



1.1.4 Dados

São as **informações** a serem representadas, armazenadas ou manipuladas.

1.1.5 Tipos de Dados

É o conjunto de valores que uma constante, ou variável, ou expressão pode assumir, ou então a um conjunto de valores que possam ser gerados por uma função.

Na definição de uma variável, constante, expressão ou método deve-se definir o **Tipo de Dado**, por algumas razões:

1) Representar um tipo abstrato de dado (**Realidade**);

2) Delimitar a faixa de abrangência (**Limites**);

3) Definir a quantidade de *bytes* para armazenamento;

4) E as operações que podem ser efetuadas.

Os tipos de dados podem ser: **Primitivos** ou **Estruturados**, sendo que os estruturados, são chamados de **Complexos**.

1.2 Tipos Primitivos

São os tipos de dados que, além de depender das características do sistema, dependem do processador e do co-processador.

1.2.1 Tipos primitivos da Linguagem de Programação Java

CARACTER: *char ch;*
INTEIRO: *int i; byte i; short i; long i;*
REAL: *float f; double d;*
LÓGICO: *boolean flag;*

Tipo	Bytes	Bits	Faixa de valores
Boolean	*1*	*8*	*true ou false*
Char	*2*	*16*	*0 à 65.535*
Byte	*1*	*8*	*-128 à 127*
Short	*2*	*16*	*-32.768 à 32.767*
Int	*4*	*32*	*-2.147.483.648 à 2.147.483.647*
Long	*8*	*64*	*-2E63 à 2E63-1*
float	*4*	*32*	*1.40239846e-46 à 3.40282347e+38*
double	*8*	*64*	*4.94065645841246544e-324 à 1.7976931348623157e+308*

Programa exemplo (1): Programa mostrando a declaração de variáveis de vários tipos.

```
// ------------------------------------------------- Fonte: Dados1.java

package dados1;
public class Dados1 {
   public static void main(String[] args) {
        boolean flag = true;
        char ch = 'a';
        byte b = 10;
        short s = 155;
        int i = 100;
        long l = 123456;
        float f = 1.23f;             // f significa float
        double d = 12345.678;
        String nome = "Paulo Roberto Gomes Luzzardi";

        System.out.println("boolean: " + flag);
        System.out.println("char: " + ch);
        System.out.println("byte: " + b);
        System.out.println("short: " + s);
        System.out.println("int: " + i);
        System.out.println("long: " + l);
        System.out.println("float: " + f);
        System.out.println("double: " + d);
        System.out.println("String: " + nome);
    }
}
```

Programa exemplo (2): Exemplo de programa para entrada de dados do tipo inteiro via teclado e saída na tela usando métodos da classe "**swing**".

```
// ------------------------------------------------------ Fonte: Dados2.java

package dados2;

import javax.swing.*;

public class Dados2 {
    public static void main(String[] args) {
     String s;

// painel de entrada de dados
     s = JOptionPane.showInputDialog("String: ");

// converte string em inteiro
     int x = Integer.parseInt(s);

     JOptionPane.showMessageDialog(null, "Inteiro: " + x, "Resultado",
                    JOptionPane.PLAIN_MESSAGE);
    }
}
```

Tipo de Ícone	Tipo de Mensagem
Ícone de erro	*JOptionPane.ERROR_MESSAGE*
Ícone de informação "i"	*JOptionPane.INFORMATION_MESSAGE*
Ícone de advertência "!"	*JOptionPane.WARNING_MESSAGE*
Ícone de pergunta "?"	*JOptionPane.QUESTION_MESSAGE*
Sem ícone	*JOptionPane.PLAIN_MESSAGE*

1.3 Construção de Tipos (Estruturados ou Complexos)

Tipos obtidos através de tipos primitivos, podem ser:

// Cadeia de Caracteres
STRING: *String s;*

// Agregados Homogêneos
VETOR E MATRIZ: *int [] v; int [][] m;*

// Classe
CLASSE: *public class **Classe** {*

1.3.1 String (Cadeia de Caracteres)

Tipo de dado que permite que uma variável possua um conjunto de caracteres.

Exemplos de declaração e inicialização:

```
String s = "";

String s = new String(20);

String s = new String ("Pelotas");
```

1.3.2 Vetor e Matriz (Agregados Homogêneos)

Tipo de dado que permite que uma variável possua um conjunto de elementos do mesmo tipo, todos alocados na memória ao mesmo tipo.

Exemplo:

```
// constante tipada inteira
final int numElementos = 10;
float [] vetor = new float [numElementos];
// vetor[0] até vetor[9]

final int numLin = 3;
final int numCol = 2;
double [][] matriz = new double[numLin][numCol];
```

Exemplos de inicialização:

```
int [] vetor = {10, 20,30, 40, 50};

int [][] matriz = {{1,2},{3,4},{5,6}};
```

Programa exemplo (3): Programa demonstra a utilização de um vetor e uma matriz 2D.

```
// ---------------------------------------------------- Fonte: Dados3.java

package dados3;

public class Dados3 {

    public static void main(String[] args) {
        int [] vetor = {10, 20,30, 40, 50};
        int [][] matriz = {{1,2},{3,4},{5,6}};
        int numLin = 3, numCol = 2;

        System.out.println("Vetor Unidimensional");
        for (int i = 0;i < vetor.length;i++) {
            System.out.println("Vetor: " + vetor[i]);
        }

        System.out.println("Matriz Bidimensional de ordem: "
```

```
                        + numLin + " x " + numCol);

        for (int i = 0;i < numLin;i++) {
            for (int j = 0;j < numCol;j++) {
            System.out.println("Matriz: " + matriz[i][j]);
            }
        }
    }
}
```

Programa exemplo (4): Uso de um vetor para ordenar valores inteiros positivos. O método utilizado é o da força bruta.

```
// ------------------------------------------------------- Fonte: Dados4.java

package dados4;

public class Dados4 {

    public static void main(String[] args) {
      int [] vetor = {5, 40, 20, 10, 50, 30};

      System.out.print("Vetor de Entrada: ");
      for (int i = 0;i < vetor.length; i++)
          System.out.print(vetor[i] + " ");
      System.out.println();

      for (int i = 0;i < vetor[0] - 1;i++)
          for (int j = i + 1; j < vetor[0]; j++)
              if (vetor[i] > vetor[j])
              {
                  int temp = vetor[i];                    // sort
                  vetor[i] = vetor[j];
                  vetor[j] = temp;
              }

      System.out.print("Vetor Ordenado: ");
      for (int i = 1;i < vetor.length; i++)
          System.out.print(vetor[i] + " ");
      System.out.println();
      System.out.println("Número de Elementos: " + vetor[0]);
    }
}
```

1.3.3 Classe (class)

Tipo de dado que permite instanciar (criar) **objetos**. Uma classe possui **atributos** (conjuntos de variáveis que o objeto possui) e **métodos** (funções que manipulam os atributos).

Programa exemplo (5): Programa mostra a declaração e uso de uma classe chamada **Aluno**.

```
// ---------------------------------------------- Fonte: Dados5.java

package dados5;

public class Dados5 {
    public static void main(String[] args) {
        Aluno aluno = new Aluno("Paulo", 10);
```

```
        aluno.exibeAtributos();
        System.out.println(aluno);
    }
}

// ---------------------------------------------- Fonte: Aluno.java

package dados5;

public class Aluno {          // ..................... classe Aluno
        private String nome;  // atributo privado a esta classe
        private int matricula;       // atributo privado a esta classe

        public Aluno(String s, int mat) {    // construtor
            nome = s;
            matricula = mat;
        }

        public void exibeAtributos()         // método
        {
            System.out.println("Nome: " + nome);
            System.out.println("matrícula: " + matricula);
        }

        public String toString() {
            return(nome + " ... " + matricula);
        }
}
```

1.4 Operadores (Aritméticos, Relacionais e Lógicos)

1.4.1 Aritméticos

+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto Inteiro da Divisão
++	Incremento
--	Decremento

1.4.2 Relacionais

>	Maior
<	Menor
>=	Maior ou Igual
<=	Menor ou Igual
==	Igual
!=	Diferente

1.4.3 Lógicos

&&	e
||	ou
!	não

2. Vetores e Matrizes

Permitem armazenamento de vários dados na memória RAM ao mesmo instante de tempo e com contigüidade física, ou seja, uma variável com possui vários elementos, igualmente distanciados, ou seja, um ao lado do outro.

2.1 Conceitos Básicos

Vetor: (É uma matriz unidimensional)

```
final int numElementos = 7;

int [] vetor = new int [numElementos];
```

Matriz: (Possui mais de uma dimensão)

```
final int numLin = 3;
final int numCol = 4;

double [][] matriz = new double [numLin][numCol];
```

Índice: Constante numérica inteira que referencia cada elemento

Exemplos: Dada a definição acima:

```
vetor[0]........................... primeiro elemento
vetor[numElementos-1].............. último elemento
m[0][0]............................ primeiro elemento
m[numLin-1][numCol-1].............. último elemento
```

Entrada de um Vetor Unidimensional:

```
final int numElementos = 7;

int [] vetor = new int [numElementos];

for (int i = 0;i < numElementos;i++)
     {
      s = JOptionPane.showInputDialog("Valor: );
      vetor[i] = Integer.parseInt(s);
    }
```

Entrada de uma Matriz Bidimensional:

```
final int numLin = 3;
```

```
final int numCol = 4;

double [][] matriz = new double [numLin][numCol];

for (int i = 0;i < numLin;i++) {
     for (int j = 0;j < numCol;j++)
         {
         s = JOptionPane.showInputDialog("Valor: );
         vetor[i][j] = Integer.parseInt(s);
         }
}
```

2.2 Exercícios com Vetores e Matrizes

1) Escreva um programa em Java que lê uma matriz A (6 x 6) e cria 2 vetores SL e SC de 6 elementos que contenham respectivamente a soma das linhas (SL) e a soma das colunas (SC). Imprimir os vetores SL e SC.

2) Escreva um programa em Java que lê uma matriz A (12 x 13) e divide todos os elementos de cada uma das 12 linhas de A pelo valor do maior elemento daquela linha. Imprimir a matriz A modificada.

Observação: Considere que a matriz armazena apenas elementos inteiros

3) Escreva um programa em Java que insere números inteiros (máximo 10 elementos) em um vetor mantendo-o ordenado. Quando o usuário digitar um ZERO o programa deve imprimir na tela o vetor ordenado.

Operações sobre os Dados

- Criação dos Dados
- Manutenção dos Dados
  - Inserção de um Componente
  - Remoção de um Componente
  - Alteração de um Componente
- Consulta aos Dados
- Destruição dos Dados
- Pesquisa e Classificação

2.3 Alocação de Memória (RAM - Random Access Memory)

2.3.1 Alocação Estática de Memória

É a forma mais simples de alocação, na qual cada dado tem sua área reservada, não variando em tamanho ou localização ao longo da execução do programa.

float f; // variável "f" ocupa 4 bytes na memória RAM

2.3.2 Alocação Dinâmica de Memória

Nesta forma de alocação, são feitas requisições e liberações de porções da memória ao longo da execução do programa.

Observação: Em Java a liberação é feita automaticamente pelo coletor de lixo (*Garbage Collector*).

2.3.3 Célula, Nodo ou Nó

Espaço reservado (alocado) na memória RAM para uma variável (tipo primitivo ou complexo), ou seja, número de bytes "*gastos*" para o armazenamento de um dado.

3. Listas Lineares

3.1 Listas Genéricas

Conceito

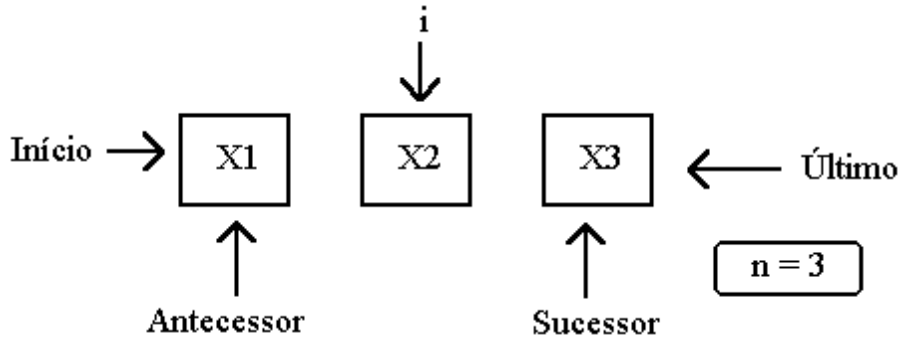
Conjunto de dados que mantém a relação de ordem ***Linear*** entre os componentes. É composta de elementos (componentes ou nós), os quais podem conter um dado ***primitivo*** ou ***estruturado***.

Lista Linear

É uma estrutura que permite representar um conjunto de dados de forma a preservar a relação de ordem entre eles.

Uma lista linear X é um conjunto de nodos (nós) X1, X2, ... Xn, Tais que:

1) Existem "n" nodos na lista (n >= 0)
2) X1 é o primeiro nodo da lista
3) Xn é o último nodo da lista
4) Para todo i,j entre 1 e n, se i<j, então o elemento Xi antecede o elemento Xj
5) Caso i = j-1, Xi é o antecessor de Xj e Xj é o sucessor de Xi



Observação: Quando n = 0, diz-se que a ***Lista*** é ***Vazia***

Exemplos de Listas

· Lista de clientes de um Banco
· Lista de Chamada
· Fichário

Operações sobre Listas

1) **Percurso**

Permite utilizar cada um dos elementos de uma lista, de tal forma que:

· 0 primeiro nodo utilizado é o primeiro da lista;
· Para utilizar o nodo Xj, todos os nodos de X1 até X(j-1) já foram utilizados;
· último nodo utilizado é o último nodo da lista.

2) **Busca**

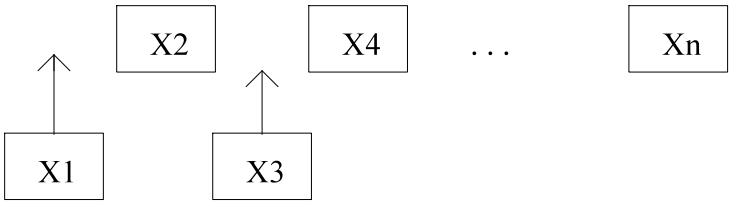
Procura um nodo específico da lista linear, de tal forma que:

· nodo é identificado por sua posição na lista;
· nodo é identificado pelo seu conteúdo.

3) **Inserção**

Acrescenta um nodo X a uma lista linear, de tal forma que:

· nodo X terá um sucessor e/ou um antecessor;
· Após inserir o nodo X na posição i (i >= 1 e i <= n+1), ele passará a ser i-ésimo nodo da lista;
· número de elementos (n) é acrescido de uma unidade.



4) **Retirada** (Exclusão)

Retira um nodo X da lista, de tal forma que:

· Se Xi é o elemento retirado, o seu sucessor passa a ser o sucessor de seu antecessor. X(i+1) passa a ser o sucessor de X(i-1). Se Xi é o primeiro nodo, o seu sucessor passa a ser o primeiro, se Xi é o último, o seu antecessor passa a ser o último;

· número de elementos (n) é decrescido de uma unidade.

Operações Válidas sobre Listas

· Acessar um elemento qualquer da lista;
· Inserir um novo elemento à lista;
· Concatenar duas listas;
· Determinar o número de elementos da lista;
· Localizar um elemento da lista com um determinado valor;
· Excluir um elemento da lista;
· Alterar um elemento da lista;
· Criar uma lista;
· Destruir a lista.

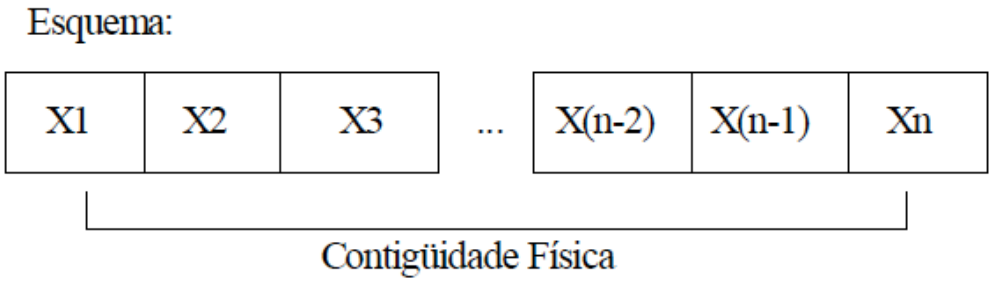
3.2 Tipos de Representações

3.2.1 Lista Representada por Contiguidade Física

Os nodos são armazenados em endereços contíguos, ou igualmente distanciados um do outro.

Os elementos são armazenados na memória um ao lado do outro, levando-se em consideração o tipo de dado, ou seja, a quantidade de bytes.

Se o endereço do nodo Xi é conhecido, então o endereço do nodo X(i+1) pode ser determinado.

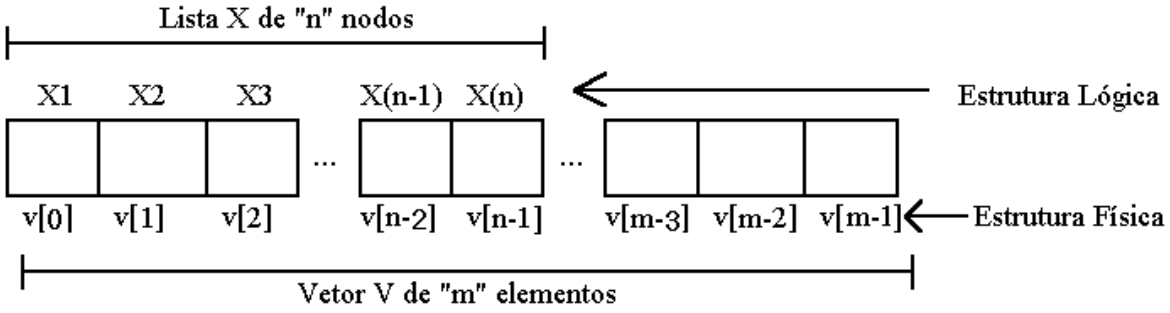


· Os relacionamentos são representados pela disposição física dos componentes na memória;
· A posição na estrutura lógica determina a posição na estrutura física.

Observação: Uma lista pode ser implementada através de um vetor de "m" elementos.

Atenção: Se "n" = "m" a ***Lista*** é chamada ***Cheia***

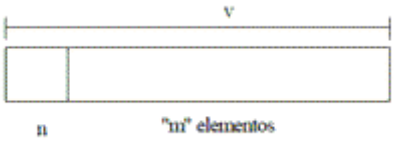
Observação: Como o número de nodos armazenados na lista pode ser modificado durante a execução do programa, deve-se representar como parte de um vetor de "m" elementos com "n <= m".

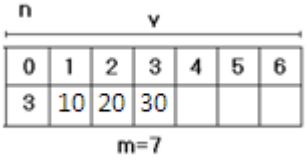


Representação: A lista X está representada por um vetor V de "m" elementos.

Componentes de uma Lista

- Número de nodos da lista (n); `// armazenado na posição 0 do vetor`
- Vetor de nodos (v);
- Tamanho total da lista (m).






```
final const m = 7;
int [] v = new int [m];
```

Observação: O número de elementos da lista será armazenado na posição 0 do vetor, ou seja: n = v[0].

Programa exemplo (6): O programa abaixo demonstra a construção de uma lista de m elementos, onde o número de

elementos é armazenado na posição zero do vetor, ou seja, n = v[0].

```
// ---------------------------------------------------------- Fonte: Dados6.java

package dados6;

import javax.swing.*;

public class Dados6 {

    final static int SUCESSO = 0;
    final static int LISTA_CHEIA = 1;
    final static int LISTA_VAZIA = 2;

    public static void main(String[] args) {
        int m = 7;
        int [] v = new int [m];
        String s;
        int valor;
        char ch;
        int erro = 0;

        criaLista(v);
        do {
            s = JOptionPane.showInputDialog("Elemento: ");
            valor = Integer.parseInt(s);
            if (valor != 0)
               erro = incluiFim(m, v, valor);
               if (erro == LISTA_CHEIA)
               {
                   JOptionPane.showMessageDialog(null,
               "Erro: Lista Cheia", "Atenção", JOptionPane.PLAIN_MESSAGE);
                   exibeLista(v);
                   return;
               }

        } while (valor != 0);
        exibeLista(v);
    }

    // ------------------------------------------------- criaLista

    static void criaLista(int [] v) {
        v[0] = 0;
    }

    // ------------------------------------------------- incluiFim

    static int incluiFim(int m, int [] v, int valor) {
        String s;
        int n = v[0];

        if (n < m-1)
        {
            v[0]++;
            v[v[0]] = valor;
            return(SUCESSO);
        }
        else
            return(LISTA_CHEIA);
    }

    // ------------------------------------------------- exibeLista

    static void exibeLista(int [] v) {
        String s = "";
```

```
        int n = v[0];

        if (n == 0)
             JOptionPane.showMessageDialog(null, "Erro: Lista Vazia", "Atenção",
JOptionPane.PLAIN_MESSAGE);
        else
        {
         for (int i = 1; i <= n; i++)
             s = s + v[i] + " ";
         JOptionPane.showMessageDialog(null, "Lista: " + s, "Lista",
JOptionPane.PLAIN_MESSAGE);
        }
    }
}
```

Problema: Incluir dados em uma **lista** de números inteiros (máximo 10 elementos), mantendo-a **ordenada.**

Programa exemplo (7): O programa abaixo demonstra a construção de uma lista de m elementos ordenados diretamente na entrada dos dados.

```
// ---------------------------------------------------------- Fonte: Dados7.java

package dados7;

import javax.swing.*;

public class Dados7 {

    final static int SUCESSO = 0;
    final static int LISTA_CHEIA = 1;
    final static int LISTA_VAZIA = 2;

    public static void main(String[] args) {
        int m = 7;
        int [] v = new int [m];
        String s;
        int valor, pos, n;
        char ch, op;
        int erro = 0;

        criaLista(v);
        s = JOptionPane.showInputDialog("Elemento: ");
        valor = Integer.parseInt(s);
        if (valor != 0)
           {
            erro = incluiInic(m, v, valor);
            do {
                   s = JOptionPane.showInputDialog("Elemento: ");
                   valor = Integer.parseInt(s);
                   if (valor != 0)
                   {
                    erro = verificaPos(m, v, valor);
                    if (erro == LISTA_CHEIA)
                       {
                        JOptionPane.showMessageDialog(null,
              "Erro: Lista Cheia", "Atenção", JOptionPane.PLAIN_MESSAGE);
                        exibeLista(v);
                        return;
                       }
                   }
             } while (valor != 0);
        }
```

```
    exibeLista(v);
}

// ------------------------------------------------- criaLista

static void criaLista(int [] v) {
    v[0] = 0;
}

// ------------------------------------------------- incluiFim

static int incluiFim(int m, int [] v, int valor) {
    String s;
    int n = v[0];

    if (n < m - 1)
    {
        v[0]++;
        v[v[0]] = valor;
        return(SUCESSO);
    }
    else
        return(LISTA_CHEIA);
}

// ------------------------------------------------- incluiInic

static int incluiInic(int m, int [] v, int valor) {
    String s;
    int n = v[0];

    if (n < m - 1)
    {
        v[0]++;
        for (int i = n + 1;i >= 2;i--)
            v[i] = v[i-1];
        v[1] = valor;
        return(SUCESSO);
    }
    else
        return(LISTA_CHEIA);
}

// ------------------------------------------------- incluiPos

static int incluiPos(int m, int [] v, int valor, int pos) {
    String s;
    int n = v[0];

    if (n < m - 1)
    {
        v[0]++;
        if (pos == n)
        {
            v[v[0]] = v[n];
            v[n] = valor;
        }
        else
            {
             for (int i = n + 1;i >= pos;i--)
                 {
                  v[i] = v[i-1];
                  System.out.println("Valor: " + v[i]);
                 }
             v[pos] = valor;
            }
        return(SUCESSO);
    }
```

```
        else
            return(LISTA_CHEIA);
    }

    // ------------------------------------------------ tamanhoLista

    static int tamanhoLista(int [] v) {
        return(v[0]);
    }

    // ------------------------------------------------ verificaPos

    static int verificaPos(int m, int [] v, int valor) {
        int i = 1;

        do {
            if (valor < v[i])
              return(incluiPos(m, v, valor, i));
            i++;
           } while (i <= v[0]);
        return(incluiFim(m, v, valor));
     }

    // ------------------------------------------------ exibeLista

    static void exibeLista(int [] v) {
        String s = "";
        int n = v[0];

        if (n == 0)
             JOptionPane.showMessageDialog(null, "Erro: Lista Vazia",
                      "Atenção", JOptionPane.PLAIN_MESSAGE);
        else
        {
         for (int i = 1; i <= n; i++)
             s = s + v[i] + " ";
         JOptionPane.showMessageDialog(null, "Lista: " + s, "Lista",
                      JOptionPane.PLAIN_MESSAGE);
        }
    }
}
```

Problema: Incluir dados em uma **lista linear** de números inteiros (máximo 50) sem **repetição.** O programa termina quando o dado lido for zero, então o programa deve imprimir a lista na tela sem repetição.

Programa exemplo (8): O programa abaixo demonstra a construção de uma lista de m elementos ordenados e SEM REPETIÇÃO diretamente na entrada dos dados.

```
// ------------------------------------------------------ Fonte: Dados8.java

package dados8;

import javax.swing.*;

public class Dados8 {

    final static int SUCESSO = 0;
    final static int LISTA_CHEIA = 1;
    final static int LISTA_VAZIA = 2;
    final static int REPETIDO = 3;
```

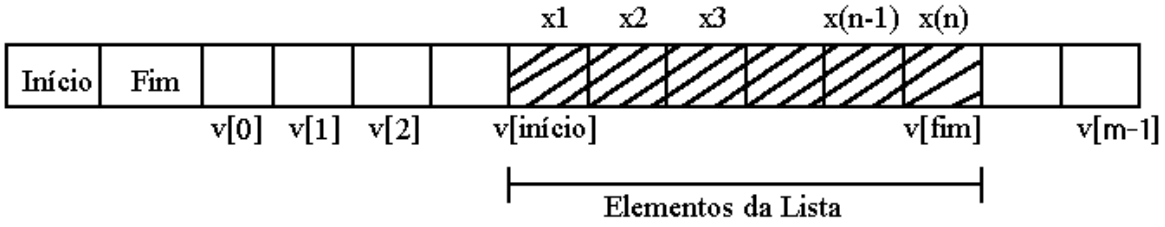
```
public static void main(String[] args) {
    int m = 7;
    int [] v = new int [m];
    String s;
    int valor, pos, n;
    char ch, op;
    int erro = 0;

    criaLista(v);
    s = JOptionPane.showInputDialog("Elemento: ");
    valor = Integer.parseInt(s);
    if (valor != 0)
       {
        erro = incluiInic(m, v, valor);
        do {
               s = JOptionPane.showInputDialog("Elemento: ");
               valor = Integer.parseInt(s);
               if (valor != 0)
               {
                erro = verificaPos(m, v, valor);
                switch (erro)
                {
                    case REPETIDO:
                        JOptionPane.showMessageDialog(null,
                                        "Erro: Elemento REPETIDO",
                                "Atenção", JOptionPane.PLAIN_MESSAGE);
                        break;
                    case LISTA_CHEIA:
                        JOptionPane.showMessageDialog(null,
                                        "Erro: Lista Cheia",
                                "Atenção", JOptionPane.PLAIN_MESSAGE);
                        exibeLista(v);
                        return;
                }
               }
          } while (valor != 0);
    }
    exibeLista(v);
}

// ------------------------------------------------- criaLista

static void criaLista(int [] v) {
    v[0] = 0;
}

// ------------------------------------------------- incluiFim

static int incluiFim(int m, int [] v, int valor) {
    String s;
    int n = v[0];

    if (n < m - 1)
    {
        v[0]++;
        v[v[0]] = valor;
        return(SUCESSO);
    }
    else
        return(LISTA_CHEIA);
}

// ------------------------------------------------- incluiInic

static int incluiInic(int m, int [] v, int valor) {
    String s;
    int n = v[0];
```

```
    if (n < m - 1)
    {
        v[0]++;
        for (int i = n + 1;i >= 2;i--)
            v[i] = v[i-1];
        v[1] = valor;
        return(SUCESSO);
    }
    else
        return(LISTA_CHEIA);
}

// ------------------------------------------------- incluiPos

static int incluiPos(int m, int [] v, int valor, int pos) {
    String s;
    int n = v[0];

    if (n < m - 1)
    {
        v[0]++;
        if (pos == n)
        {
            v[v[0]] = v[n];
            v[n] = valor;
        }
        else
            {
             for (int i = n + 1;i >= pos;i--)
                  v[i] = v[i-1];
             v[pos] = valor;
            }
        return(SUCESSO);
    }
    else
        return(LISTA_CHEIA);
}

// ------------------------------------------------- tamanhoLista

static int tamanhoLista(int [] v) {
    return(v[0]);
}

// ------------------------------------------------- verificaPos

static int verificaPos(int m, int [] v, int valor) {
    int i = 1;

    do {
        if (valor == v[i])
            return(REPETIDO);
        if (valor < v[i])
          return(incluiPos(m, v, valor, i));
        i++;
       } while (i <= v[0]);
    return(incluiFim(m, v, valor));
 }

// ------------------------------------------------- exibeLista

static void exibeLista(int [] v) {
    String s = "";
    int n = v[0];

    if (n == 0)
         JOptionPane.showMessageDialog(null, "Erro: Lista Vazia",
```

```
                                "Atenção", JOptionPane.PLAIN_MESSAGE);
        else
        {
         for (int i = 1; i <= n; i++)
             s = s + v[i] + " ";
         JOptionPane.showMessageDialog(null, "Lista: " + s, "Lista",
                        JOptionPane.PLAIN_MESSAGE);
        }
    }
}
```

Contiguidade Física

Uma alternativa para representação por contigüidade física é não iniciar no início do vetor, isto facilita as inserções.



Observação: As operações de inclusão e exclusão de nodos podem optar pela extremidade da lista que irá diminuir (no caso de exclusão) ou aumentar (no caso de inserção) de comprimento. "A escolha deverá considerar o caso que produz menor movimentação de elementos".

```
final int m = 7;
int inicio;
int fim;
int []v = new int[m];

ou

final int m = 7;
int []v = new int[m + 2];

// onde v[0] = inic e v[m] = fim
```

Lista vazia

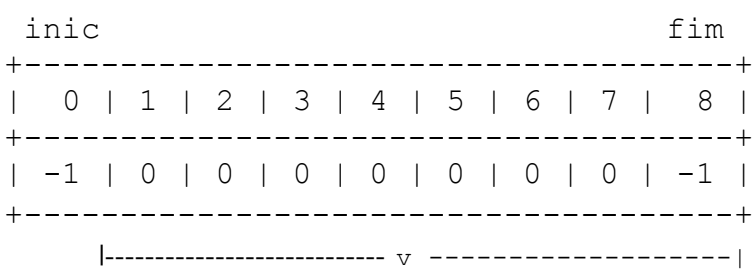
```
início  = -1
fim = -1
```

Lista cheia

```
início = 0
fim = m - 1
```

Problema: Escreva um programa em Java que permite a inclusão de números inteiros no ***início*** ou no ***fim*** da **lista linear** (máximo 7 elementos), partindo do meio do vetor:

metade =(m / 2) + 1



Programa exemplo (9): O programa abaixo demonstra a inclusão de números inteiros no ***início*** ou no ***fim*** de uma **lista linear.**

```
// ------------------------------------------------------ Fonte: Dados9.java

package dados9;

import java.util.Scanner;

public class Dados9 {

    final static int SUCESSO = 0;
    final static int LISTA_CHEIA = 1;
    final static int LISTA_VAZIA = 2;

    public static void main(String[] args) {
        Scanner entrada = new Scanner (System.in);
        String s;
        final int m = 7;
        int [] v = new int[m + 2];      // v[0] = inic e v[m] = fim
        int valor;
        int erro = SUCESSO;
        char ch = 'A';

        criaLista(v);
        do {
            exibeLista(v);
            System.out.printf("\nValor: ");
            s = entrada.nextLine();
            valor = Integer.parseInt(s);
            if (valor != 0) {
                System.out.printf("[I]nício ou [F]im ?");
               do {
                    s = entrada.nextLine();
                    ch = s.charAt(0);
               } while (ch != 'I' && ch != 'i' && ch != 'F' && ch != 'f');
            }
           switch (ch) {
                case 'I':
                case 'i': erro = incluiInicio(v, valor);
                   break;
               case 'F':
                case 'f': erro = incluiFim(v, valor);
                   break;
           }
```

```
        if (erro != SUCESSO) {
            imprimeErro(erro);
         }
    } while (valor != 0 && erro != LISTA_CHEIA);
    exibeLista(v);
    System.exit(0);
 }

// ---------------------------------------------- criaLista

static void criaLista(int []v) {
    int inic = 0;
    int fim = v.length - 1;

    v[inic] = -1;
    v[fim] = -1;
}

// ---------------------------------------------- incluiInicio

static int incluiInicio(int [] v, int dado) {
    int inic = v[0];
    int fim = v[v.length - 1];
    int m = v.length - 2;
    int metade = (m / 2) + 1;

    if (inic == -1) {
        inic = metade;
        fim = inic;
        v[inic] = dado;
        v[0] = inic;
        v[v.length - 1] = fim;
        return(SUCESSO);
    }
    else {
        if (inic == 1) {
            return(LISTA_CHEIA);
        }
        else {
            inic--;
           v[inic] = dado;
            v[0] = inic;
           return(SUCESSO);
       }
    }
}

// ---------------------------------------------- incluiFim

static int incluiFim(int [] v, int dado) {
    int inic = v[0];
    int fim = v[v.length - 1];
    int m = v.length - 2;
    int metade = (m / 2) + 1;

    if (fim == -1) {
        inic = metade;
        fim = inic;
        v[fim] = dado;
        v[0] = inic;
        v[v.length - 1] = fim;
        return(SUCESSO);
    }
    else {
        if (fim == m) {
            return(LISTA_CHEIA);
        }
        else {
```

```
                fim++;
                v[fim] = dado;
                v[v.length - 1] = fim;
                return(SUCESSO);
            }
        }
    }

    // ---------------------------------------------- imprimeErro

    static void imprimeErro(int erro) {
        switch (erro) {
            case LISTA_VAZIA: System.out.println("ERRO: Lista Vazia");
                  break;
            case LISTA_CHEIA: System.out.println("ERRO: Lista Cheia");
                  break;
        }
    }

    // ---------------------------------------------- exibeLista

    static void exibeLista(int [] v) {
        int inic = v[0];
        int fim = v[v.length - 1];

        System.out.println("inic: " + inic);
        System.out.println(" fim: " + fim);
        System.out.printf("indices: ");
        for (int i = 0;i < v.length;i++) {
             System.out.printf("%2d ",i);
        }
        System.out.println();
        System.out.printf("  Lista: ");
        for (int i = 0;i < v.length;i++) {
             System.out.printf("%2d ",v[i]);
        }
    }
}
```

Problema: Escreva um programa em Java que permite a inclusão de números inteiros no **início** ou no **fim** da **lista linear** (máximo 7 elementos) **avisando qual lado está cheio.**

Observação: Note que a lista pode estar cheia num lado e não estar cheia no outro lado. A próxima solução (*10*) avisa ao usuário qual lado da **lista linear** está cheia.

Programa exemplo (10): O programa abaixo demonstra a inclusão de números inteiros no **início** ou no **fim** de uma **lista linear** avisando qual lado está cheio.

```
// ------------------------------------------------------- Fonte: Dados10.java

package dados10;

import java.util.Scanner;

public class Dados10 {

    final static int SUCESSO = 0;
    final static int LISTA_CHEIA_DIREITA = 1;
    final static int LISTA_CHEIA_ESQUERDA = 2;
```

```
public static void main(String[] args) {
    Scanner entrada = new Scanner (System.in);
    String s;
    final int m = 7;
    int [] v = new int[m + 2];      // v[0] = inic e v[m] = fim
    int valor;
    int erro = SUCESSO;
    char ch = 'A';

    criaLista(v);
    do {
        exibeLista(v);
        System.out.printf("\nValor: ");
        s = entrada.nextLine();
        valor = Integer.parseInt(s);
        if (valor != 0) {
            System.out.printf("[I]nício ou [F]im ?");
           do {
                s = entrada.nextLine();
                ch = s.charAt(0);
           } while (ch != 'I' && ch != 'i' && ch != 'F' && ch != 'f');
        }
       switch (ch) {
            case 'I':
            case 'i': erro = incluiInicio(v, valor);
              break;
           case 'F':
            case 'f': erro = incluiFim(v, valor);
              break;
       }
       if (erro != SUCESSO) {
            imprimeErro(erro);
        }
    } while (valor != 0);
    exibeLista(v);
    System.exit(0);
 }

// ---------------------------------------------- criaLista

static void criaLista(int []v) {
    int inic = 0;
    int fim = v.length - 1;

    v[inic] = -1;
    v[fim] = -1;
}

// ---------------------------------------------- incluiInicio

static int incluiInicio(int [] v, int dado) {
    int inic = v[0];
    int fim = v[v.length - 1];
    int m = v.length - 2;
    int metade = (m / 2) + 1;

    if (inic == -1) {
        inic = metade;
        fim = inic;
        v[inic] = dado;
        v[0] = inic;
        v[v.length - 1] = fim;
        return(SUCESSO);
    }
    else {
        if (inic == 1) {
            return(LISTA_CHEIA_ESQUERDA);
```

```
        }
        else {
            inic--;
           v[inic] = dado;
            v[0] = inic;
           return(SUCESSO);
        }
    }
}

// ---------------------------------------------- incluiFim

static int incluiFim(int [] v, int dado) {
    int inic = v[0];
    int fim = v[v.length - 1];
    int m = v.length - 2;
    int metade = (m / 2) + 1;

    if (fim == -1) {
        inic = metade;
        fim = inic;
        v[fim] = dado;
        v[0] = inic;
        v[v.length - 1] = fim;
        return(SUCESSO);
    }
    else {
        if (fim == m) {
            return(LISTA_CHEIA_DIREITA);
        }
        else {
            fim++;
           v[fim] = dado;
            v[v.length - 1] = fim;
           return(SUCESSO);
        }
    }
}

// ---------------------------------------------- imprimeErro

static void imprimeErro(int erro) {
    switch (erro) {
        case LISTA_CHEIA_ESQUERDA: System.out.println("ERRO: Lista Cheia à Esquerda");
              break;
        case LISTA_CHEIA_DIREITA: System.out.println("ERRO: Lista Cheia à Direita");
              break;
    }
}

// ---------------------------------------------- exibeLista

static void exibeLista(int [] v) {
    int inic = v[0];
    int fim = v[v.length - 1];

    System.out.println("inic: " + inic);
    System.out.println(" fim: " + fim);
    System.out.printf("indices: ");
    for (int i = 0;i < v.length;i++) {
         System.out.printf("%2d ",i);
    }
    System.out.println();
    System.out.printf("  Lista: ");
    for (int i = 0;i < v.length;i++) {
         System.out.printf("%2d ",v[i]);
    }
}
```

```
}
```

Problema: Escreva um programa em Java que permite a inclusão de números inteiros em uma **lista linear** no ***início***, ***fim*** e na ***posição*** escolhida pelo usuário

Programa exemplo (11): O programa abaixo demonstra a inclusão de números inteiros no **início**, na posição ou no **fim** de uma **lista linear.**

```
// ---------------------------------------------------- Fonte: Dados11.java

package dados11;

import java.util.Scanner;

public class Dados11 {

    final static int SUCESSO = 0;
    final static int LISTA_CHEIA = 1;
    final static int LISTA_VAZIA = 2;
    final static int POSICAO_INVALIDA = 3;

    public static void main(String[] args) {
        Scanner entrada = new Scanner (System.in);
        String s;
        final int m = 7;
        int [] v = new int[m + 2];      // v[0] = inic e v[m] = fim
        int valor, pos;
        int erro = SUCESSO;
        char ch = 'A';

        criaLista(v);
        do {
            exibeLista(v);
            System.out.printf("\nValor: ");
            s = entrada.nextLine();
            valor = Integer.parseInt(s);
            if (valor != 0) {
                System.out.printf("[I]nício, [P]osição ou [F]im ?");
                do {
                    s = entrada.nextLine();
                    ch = s.charAt(0);
                } while (ch != 'I' && ch != 'i' &&
                       ch != 'P' && ch != 'p' && ch != 'F' && ch != 'f');
            }
            switch (ch) {
                case 'I':
                case 'i': erro = incluiInicio(v, valor);
                          break;
                case 'F':
                case 'f': erro = incluiFim(v, valor);
                          break;
                case 'P':
                case 'p': System.out.printf("Posição: ");
                          s = entrada.nextLine();
                          pos = Integer.parseInt(s);
                          erro = incluiPosicao(v, valor, pos);
                   break;
            }
            if (erro != SUCESSO) {
                imprimeErro(erro);
            }
        } while (valor != 0);
```

```
    exibeLista(v);
    System.exit(0);
 }

// ---------------------------------------------- criaLista

static void criaLista(int []v) {
    int inic = 0;
    int fim = v.length - 1;

    v[inic] = -1;
    v[fim] = -1;
}

// ---------------------------------------------- incluiInicio

static int incluiInicio(int [] v, int dado) {
    int inic = v[0];
    int fim = v[v.length - 1];
    int m = v.length - 2;
    int metade = (m / 2) + 1;

    if (inic == -1) {
        inic = metade;
        fim = inic;
        v[inic] = dado;
        v[0] = inic;
        v[v.length - 1] = fim;
        return(SUCESSO);
    }
    else {
        if (inic == 1) {
            return(LISTA_CHEIA);
        }
        else {
            inic--;
           v[inic] = dado;
            v[0] = inic;
           return(SUCESSO);
       }
    }
}

// ---------------------------------------------- incluiFim

static int incluiFim(int [] v, int dado) {
    int inic = v[0];
    int fim = v[v.length - 1];
    int m = v.length - 2;
    int metade = (m / 2) + 1;

    if (fim == -1) {
        inic = metade;
        fim = inic;
        v[fim] = dado;
        v[0] = inic;
        v[v.length - 1] = fim;
        return(SUCESSO);
    }
    else {
        if (fim == m) {
            return(LISTA_CHEIA);
        }
        else {
            fim++;
           v[fim] = dado;
            v[v.length - 1] = fim;
           return(SUCESSO);
```

```
        }
    }
}

// ---------------------------------------------- incluiPosicao

static int incluiPosicao(int [] v, int dado, int pos) {
    int erro;
    int inic = v[0];
    int fim = v[v.length - 1];
    int m = v.length - 2;

    if (pos < inic || pos > fim) {
        return(POSICAO_INVALIDA);
    }
    if (inic == -1) {
        inic = pos;
       fim = pos;
       v[inic] = dado;
        v[0] = inic;
        v[v.length - 1] = fim;
       return(SUCESSO);
   }
    else {
        if (inic == 0 && fim == m - 1) {
           return(LISTA_CHEIA);
        }
       else {
            if (pos == inic - 1) {
                erro = incluiInicio(v, dado);
              return(erro);
           }
           else {
               if (pos == fim + 1) {
                     erro = incluiFim(v, dado);
                     return(erro);
                }
               else {
                      for (int i = fim;i >= pos;i--) {
                            v[i+1] = v[i];
                        }
                      v[pos] = dado;
                      fim++;
                        v[0] = inic;
                        v[v.length - 1] = fim;
                      return(SUCESSO);
                }
            }
        }
    }
}

// ---------------------------------------------- imprimeErro

static void imprimeErro(int erro) {
    switch (erro) {
        case LISTA_CHEIA: System.out.println("ERRO: Lista Cheia");
               break;
        case LISTA_VAZIA: System.out.println("ERRO: Lista Vazia");
               break;
        case POSICAO_INVALIDA: System.out.println("ERRO: Posição Inválida");
               break;
    }
}

// ---------------------------------------------- exibeLista

static void exibeLista(int [] v) {
```

```
        int inic = v[0];
        int fim = v[v.length - 1];

        System.out.println("inic: " + inic);
        System.out.println(" fim: " + fim);
        System.out.printf("indices: ");
        for (int i = 0;i < v.length;i++) {
             System.out.printf("%2d ",i);
        }
        System.out.println();
        System.out.printf("  Lista: ");
        for (int i = 0;i < v.length;i++) {
             System.out.printf("%2d ",v[i]);
        }
    }
}
```

Vantagens e Desvantagens da Representação por Contigüidade Física

Vantagens

- A consulta pode ser calculada (acesso randômico aos dados);
- Facilita a transferência de dados (área de memória contígua);
- Adequada para o armazenamento de estruturas simples.

Desvantagens

- O tamanho máximo da lista precisa ser conhecido e alocado antecipadamente, pois a lista é alocada estaticamente na memória;
- Inserções e remoções podem exigir considerável movimentação de dados;
- Inadequada para o armazenamento de estruturas complexas;
- Mantém um espaço de memória ocioso (não ocupado);
- Como a lista é limitada, devem ser testados os limites.

3.2.2 Lista representada por Encadeamento

Permite **Alocação Dinâmica de Memória**, ou seja, a lista cresce com a execução do programa. Operações como inserção e remoção são mais simples. Este tipo de estrutura se chama **Lista Encadeada Simples** ou **Lista Encadeada**.

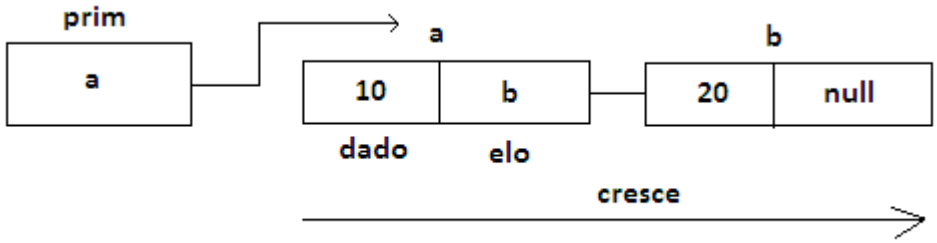
A seguir será visto duas formas de implementar uma lista encadeada simples:

a) Usando uma classe do Java ***LinkedList***;
b) Usando classes do programador: **Nodo** e **ListaEncadeada**.

Problema: Escrever um programa em Java que permite incluir números inteiros em uma **lista encadeada.**

Programa exemplo (13): O programa abaixo demonstra a inclusão de números inteiros em uma lista encadeada simples usando a classe ***LinkedList***.

```
// ---------------------------------------------------- Fonte: Dados13.java

package dados13;

import java.util.*;

public class Dados13 {

    public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);
        String s;
        LinkedList lista = new LinkedList();

        System.out.print("Nodo: ");
        s = entrada.nextLine();
        int nodo = Integer.parseInt(s);
        Object primeiro = lista.add(nodo);
        do {
            System.out.print("Nodo: ");
            s = entrada.nextLine();
            nodo = Integer.parseInt(s);
            if (nodo != 0) {
                lista.add(nodo);
            }
        } while (nodo != 0);

        System.out.println("Lista: " + lista);
        for (int i = 0;i < lista.size();i++) {
            primeiro = lista.removeFirst();
            System.out.println("Nodo: " + primeiro);
        }
        System.exit(0);
    }
}
```



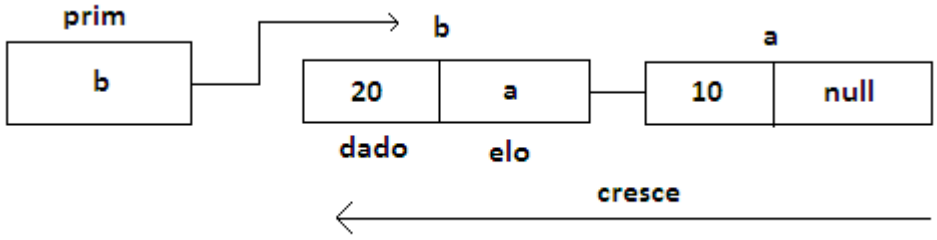
Problema: Escrever um programa em Java que insere dados em uma **lista encadeada,** permitindo obter o conteúdo do último elemento, imprimindo também, toda a lista.

Programa exemplo (14): O programa abaixo demonstra a inclusão de números inteiros em uma lista encadeada simples usando as classes: **Nodo** e **ListaEncadeada** escritas pelo programador.

```
// ------------------------------------------------ Fonte: Nodo.java

package dados14;

public class Nodo {
    public int dado;
    public Nodo elo = null;
}

// ------------------------------------------------ Fonte: ListaEncadeada.java

package dados14;

public class ListaEncadeada {

    private Nodo prim = null;
    private Nodo ult = null;

    // ------------------------------- inserirLista

    public void inserirLista(int chave)
    {
       Nodo aux = new Nodo();

       aux.dado = chave;
       if (prim == null) {
          prim = aux;
       }
       else {
          ult.elo = aux;
       }
       ult = aux;
    }

    // ------------------------------- removerLista

    public int removerLista() {

        Nodo aux, tmp;

        aux = tmp = prim;

        if (aux != null) {
            prim = aux.elo;
            if (prim == null) {
                ult = null;
            }
        return(tmp.dado);
        }
        else
            return(0);
     }

    // ------------------------------- imprimirLista

    public void imprimirLista() {
        Nodo aux;

        aux = prim;

        System.out.print("Lista Encadeada: ");
        if (aux != null) {
            while(aux != null) {
                System.out.print(aux.dado + " ");
                aux = aux.elo;
            }
```

```
        }
        else {
              System.out.print("Vazia");
        }
        System.out.println();
    }

 }
```

```
// ---------------------------------------------------- Fonte: Dados14.java

package dados14;

import java.util.Scanner;

public class Dados14 {

    public static void main(String[] args) {
        ListaEncadeada lista = new ListaEncadeada();
        Scanner entrada = new Scanner(System.in);
        String s;
        int dado;

        do {
            System.out.print("Dado: ");
            s = entrada.nextLine();
            dado = Integer.parseInt(s);
            if (dado != 0) {
                lista.inserirLista(dado);
            }
        } while (dado != 0);

        lista.imprimirLista();

        dado = lista.removerLista();
        while (dado != 0) {
            System.out.println("Dado Removido: " + dado);
            dado = lista.removerLista();
        }
        System.exit(0);
    }
}
```



Problema: Escrever um programa em Java que insere dados em uma **lista encadeada simples**, permitindo obter o conteúdo do último elemento, imprimindo também, toda a lista.

Programa exemplo (15): O programa abaixo demonstra a inclusão de números inteiros em uma lista encadeada simples usando as classes: **Nodo** e **ListaEncadeadaSimples.**

```
// ---------------------------------------- Fonte: Nodo.java

package dados15;

public class Nodo {
    public int dado;
    public Nodo elo = null;
}

// ------------------------------------------------ ListaEncadeadaSimples.java

package dados15;

public class ListaEncadeadaSimples {

    private Nodo primeiro = null;

    // ------------------------------- inserirLista

    public void inserirLista(int valor)
    {
       Nodo aux = new Nodo();
       Nodo tmp;

       aux.dado = valor;
       if (primeiro == null) {
          primeiro = aux;
       }
       else {
           tmp = primeiro;
           aux.elo = tmp;
           primeiro = aux;
       }
    }

    // ------------------------------- removerLista

    public int removerLista() {

        Nodo aux;
        int dado;

        aux = primeiro;

        if (aux != null) {
            dado = primeiro.dado;
            primeiro = primeiro.elo;
            return(dado);
        }
        else
            return(0);
     }

    // ------------------------------- imprimirLista

    public void imprimirLista() {
        Nodo aux;

        aux = primeiro;

        System.out.print("Lista Encadeada Simples: ");
        if (aux != null) {
            while(aux != null) {
                System.out.print(aux.dado + " ");
                aux = aux.elo;
            }
        }
        else {
```

```
            System.out.print("Vazia");
        }
        System.out.println();
    }

}

// ---------------------------------------- Fonte: Dados15.java

package dados15;

import java.util.Scanner;

public class Dados15 {

    public static void main(String[] args) {
        ListaEncadeadaSimples lista = new ListaEncadeadaSimples();
        Scanner entrada = new Scanner(System.in);
        String s;
        int dado;

        do {
            System.out.print("Dado: ");
            s = entrada.nextLine();
            dado = Integer.parseInt(s);
            if (dado != 0) {
                lista.inserirLista(dado);
            }
        } while (dado != 0);

        lista.imprimirLista();

        dado = lista.removerLista();
        while (dado != 0) {
            System.out.println("Dado Removido: " + dado);
            dado = lista.removerLista();
        }
        System.exit(0);
    }
}
```

Problema: Escrever um programa em Java que permite **incluir, excluir** e **consultar** (no *início* ou *fim*) dados inteiros em uma **lista encadeada.** Em cada operação imprimir a lista.

Programa exemplo (16): O programa abaixo demonstra a inclusão, exclusão e consulta de números inteiros em uma lista encadeada simples usando as classes: **Nodo** e **ListaEncadeadaSimples.**

```
// ---------------------------------------- Fonte: Nodo.java

package dados16;

public class Nodo {
    public int dado;
    public Nodo elo = null;
}

// ---------------------------------------- Fonte: ListaEncadeadaSimples.java

package dados16;
```

```
public class ListaEncadeada {

    private Nodo primeiro = null;

    // ------------------------------- inserirLista

    public void inserirLista(int valor)
    {
       Nodo aux = new Nodo();
       Nodo tmp;

       aux.dado = valor;
       if (primeiro == null) {
          primeiro = aux;
       }
       else {
           tmp = primeiro;
           aux.elo = tmp;
           primeiro = aux;
       }
    }

    // ------------------------------- removerLista

    public int removerLista() {

        Nodo aux;
        int dado;

        aux = primeiro;

        if (aux != null) {
            dado = primeiro.dado;
            primeiro = primeiro.elo;
            return(dado);
        }
        else
            return(0);
     }

    // ------------------------------- consultarLista

    public int consultarLista() {

        Nodo aux;
        int dado;

        aux = primeiro;

        if (aux != null) {
            return(primeiro.dado);
        }
        else
            return(0);
     }

    // ------------------------------- imprimirLista

    public void imprimirLista() {
        Nodo aux;

        aux = primeiro;

        System.out.print("Lista Encadeada Simples: ");
        if (aux != null) {
            while(aux != null) {
                System.out.print(aux.dado + " ");
                aux = aux.elo;
```

```
            }
        }
        else {
              System.out.print("Vazia");
        }
        System.out.println();
    }

}

// ---------------------------------------- Fonte: Dados16.java

package dados16;

import java.util.Scanner;

public class Dados16 {

    public static void main(String[] args) {
        ListaEncadeada lista = new ListaEncadeada();
        Scanner entrada = new Scanner(System.in);
        String s;
        char op;
        int dado = 0;

        do {
            System.out.print("[I]ncluir, [C]onsultar, [R]emover ou [F]im: ");
            do {
                s = entrada.nextLine();
                op = s.charAt(0);
                op = Character.toLowerCase(op);
            } while (op != 'i' && op != 'c' && op != 'r' && op != 'f');
            switch (op) {
                case 'i': System.out.print("Dado: ");
                          s = entrada.nextLine();
                          dado = Integer.parseInt(s);
                          if (dado != 0) {
                              lista.inserirLista(dado);
                          }
                          break;
                case 'c': dado = lista.consultarLista();
                          System.out.println("Dado Consultado: " + dado);
                          break;
                case 'r': dado = lista.removerLista();
                          System.out.println("Dado Removido: " + dado);
                          break;
            }
        lista.imprimirLista();
        } while (op != 'f');
        System.exit(0);
    }
}
```

Vantagens das Listas representadas por Encadeamento

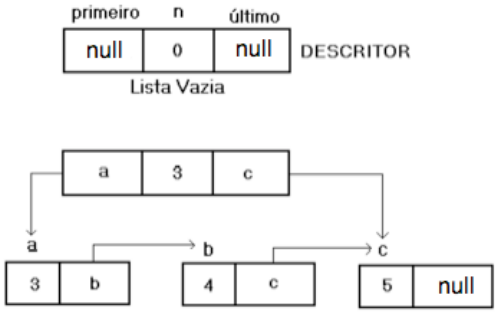
- Lista cresce indeterminadamente, enquanto houver memória livre (Alocação Dinâmica de Memória);
- As operações de inserção e remoção de elementos não exige a movimentação dos demais elementos.

Desvantagens das Listas representadas por Encadeamento

- Determinar o número de elementos da lista, pois para tanto, deve-se percorrer toda a lista;
- Acessar diretamente um determinado elemento pela sua posição, pois só é conhecido o primeiro elemento da lista;
- Acessar o último elemento da lista, pois para acessá-lo, deve-se "**visitar**" todos os intermediários.

3.2.3 Lista Encadeada com Descritor

Como foi visto anteriormente, as dificuldades da lista encadeada, é descobrir o **número de elementos** e ainda, acessar o **último elemento**. Estas dificuldades podem ser resolvidas utilizando-se um **descritor**, da seguinte forma:



Problema: Escrever o mesmo programa em Java que insere dados em uma **lista encadeada com descritor**, permitindo obter o conteúdo do último elemento diretamente, imprimindo também, toda a lista.

Programa exemplo (17): O programa abaixo demonstra a inclusão e consulta de números inteiros (à esquerda e à direita) em uma lista encadeada simples com **descritor** usando as classes: **Nodo** e **ListaEncadeadaDescritor.**

```
// ---------------------------------------------- Fonte: Nodo.java

package dados17;

public class Nodo {
    public int dado;
    public Nodo elo = null;
}

// ---------------------------------------------- Fonte: ListaEncadeadaDescritor.java

package dados17;

public class ListaEncadeadaDescritor {
```

```
private Nodo primeiro = null;
private int n = 0;
private Nodo ultimo = null;

// ------------------------------ inserirListaEsquerda

public void inserirListaEsquerda(int valor)
{
   Nodo aux = new Nodo();

   aux.dado = valor;
   if (n == 0) {
      primeiro = aux;
      ultimo = aux;
      aux.elo = null;
   }
   else {
       aux.elo = primeiro;
       primeiro = aux;
   }
   n++;
}

// ------------------------------ inserirListaDireita

public void inserirListaDireita(int valor)
{
   Nodo aux = new Nodo();
   Nodo tmp;

   aux.dado = valor;
   if (n == 0) {
      primeiro = aux;
      ultimo = aux;
   }
   else {
       tmp = ultimo;
       tmp.elo = aux;
       aux.elo = null;
       ultimo = aux;
   }
   n++;
}

// ------------------------------ consultarListaEsquerda

public int consultarListaEsquerda() {

    Nodo aux;
    int dado;

    aux = primeiro;

    if (aux != null) {
        return(primeiro.dado);
    }
    else
        return(0);
 }

// ------------------------------ consultarListaDireita

public int consultarListaDireita() {

    Nodo aux;
    int dado;

    aux = ultimo;
```

```
        if (aux != null) {
            return(ultimo.dado);
        }
        else
            return(0);
     }

    // ------------------------------ imprimirLista

    public void imprimirLista() {
        Nodo aux;

        aux = primeiro;

        System.out.print("Lista Encadeada Simples: ");
        if (aux != null) {
            while(aux != null) {
                System.out.print(aux.dado + " ");
                aux = aux.elo;
            }
        }
        else {
              System.out.print("Vazia");
        }
        System.out.println();
    }
}

// ------------------------------------------------ Fonte: Dados17.java

package dados17;

import java.util.Scanner;

public class Dados17 {

    public static void main(String[] args) {
        ListaEncadeadaDescritor lista = new ListaEncadeadaDescritor();
        Scanner entrada = new Scanner(System.in);
        String s;
        char op, ch = 'e';
        int dado = 0;

        do {
            System.out.print("[I]ncluir, [C]onsultar ou [F]im: ");
            do {
                s = entrada.nextLine();
                op = s.charAt(0);
                op = Character.toLowerCase(op);
            } while (op != 'i' && op != 'c' && op != 'f');
            if (op == 'i' || op == 'c') {
                do {
                    System.out.print("[E]squerda ou [D]ireita: ");
                    s = entrada.nextLine();
                    ch = s.charAt(0);
                    ch = Character.toLowerCase(ch);
                } while (ch != 'e' && ch != 'd');
            }

            switch (op) {
                case 'i': System.out.print("Dado: ");
                          s = entrada.nextLine();
                          dado = Integer.parseInt(s);
                          if (dado != 0) {
                              if (ch == 'e') {
                                 lista.inserirListaEsquerda(dado);
                              }
```

```
                            else {
                               lista.inserirListaDireita(dado);
                            }
                        }
                        break;
                case 'c': if (ch == 'e') {
                            dado = lista.consultarListaEsquerda();
                        }
                        else {
                            dado = lista.consultarListaDireita();
                        }
                        System.out.println("Dado Consultado: " + dado);
                        break;
            }
        lista.imprimirLista();
        } while (op != 'f');
        System.exit(0);
    }
}
```

Problema: Escrever um programa em Java que permite **incluir, excluir** e **consultar** (no *início* ou *fim*) dados inteiros em uma **lista encadeada.** Em cada operação imprimir a lista.

Programa exemplo (18): O programa abaixo demonstra a inclusão, remoção e consulta de números inteiros (à esquerda e à direita) em uma lista encadeada simples com **descritor** usando as classes: **Nodo** e **ListaEncadeadaDescritor.**

```
// ------------------------------------------------------ Fonte: Nodo.java

package dados18;

public class Nodo {
    public int dado;
    public Nodo elo = null;
}
```

```
// ------------------------------------------------------ Fonte: Dados18.java

package dados18;

public class ListaEncadeadaDescritor {

    private Nodo primeiro = null;
    private int n = 0;
    private Nodo ultimo = null;

    // ------------------------------- inserirListaEsquerda

    public void inserirListaEsquerda(int valor)
    {
       Nodo aux = new Nodo();

       aux.dado = valor;
       if (n == 0) {
          primeiro = aux;
          ultimo = aux;
          aux.elo = null;
       }
       else {
           aux.elo = primeiro;
           primeiro = aux;
```

```
   }
   n++;
}

// ------------------------------ inserirListaDireita

public void inserirListaDireita(int valor)
{
   Nodo aux = new Nodo();
   Nodo tmp;

   aux.dado = valor;
   if (n == 0) {
      primeiro = aux;
      ultimo = aux;
   }
   else {
       tmp = ultimo;
       tmp.elo = aux;
       aux.elo = null;
       ultimo = aux;
   }
   n++;
}

// ------------------------------ removerListaEsquerda

public int removerListaEsquerda() {
    Nodo aux = primeiro;
    int dado;

    if (n == 0) {
        return(0);
    }
    else {
        dado = aux.dado;
        primeiro = aux.elo;
        n--;
        return(dado);
    }
}

// ------------------------------ removerListaDireita

public int removerListaDireita() {
    Nodo tmp, ant = primeiro, aux = ultimo;
    int dado;

    if (n == 0) {
        return(0);
    }
    else {
        dado = aux.dado;
        n--;
        tmp = primeiro;
        while (tmp.elo != null) {
            ant = tmp;
            tmp = tmp.elo;
        }
        ant.elo = null;
        ultimo = ant;
        return(dado);
    }
}

// ------------------------------ consultarListaEsquerda

public int consultarListaEsquerda() {
```

```
        Nodo aux;
        int dado;

        aux = primeiro;

        if (aux != null) {
            return(primeiro.dado);
        }
        else
            return(0);
     }

    // ------------------------------ consultarListaDireita

    public int consultarListaDireita() {

        Nodo aux;
        int dado;

        aux = ultimo;

        if (aux != null) {
            return(ultimo.dado);
        }
        else
            return(0);
     }

    // ------------------------------ imprimirLista

    public void imprimirLista() {
        Nodo aux;

        aux = primeiro;

        System.out.print("Lista Encadeada Simples: ");
        if (aux != null) {
            while(aux != null) {
                System.out.print(aux.dado + " ");
                aux = aux.elo;
            }
        }
        else {
              System.out.print("Vazia");
        }
        System.out.println();
    }
}

// ---------------------------------------------------------- Fonte: Dados18.java

package dados18;

import java.util.Scanner;

public class Dados18 {

    public static void main(String[] args) {
        ListaEncadeadaDescritor lista = new ListaEncadeadaDescritor();
        Scanner entrada = new Scanner(System.in);
        String s;
        char op, ch = 'e';
        int dado = 0;

        do {
            System.out.print("[I]ncluir, [C]onsultar, [R]emover ou [F]im: ");
            do {
```

```
            s = entrada.nextLine();
            op = s.charAt(0);
            op = Character.toLowerCase(op);
        } while (op != 'i' && op != 'c' && op != 'r' && op != 'f');
        if (op == 'i' || op == 'r' || op == 'c') {
            do {
                System.out.print("[E]squerda ou [D]ireita: ");
                s = entrada.nextLine();
                ch = s.charAt(0);
                ch = Character.toLowerCase(ch);
            } while (ch != 'e' && ch != 'd');
        }

        switch (op) {
            case 'i': System.out.print("Dado: ");
                      s = entrada.nextLine();
                      dado = Integer.parseInt(s);
                      if (dado != 0) {
                          if (ch == 'e') {
                             lista.inserirListaEsquerda(dado);
                          }
                          else {
                             lista.inserirListaDireita(dado);
                          }
                      }
                      break;
            case 'r': if (ch == 'e') {
                             dado = lista.removerListaEsquerda();
                          }
                          else {
                             dado = lista.removerListaDireita();
                          }
                      if (dado != 0) {
                          System.out.println("Nodo Removido: " + dado);
                      }
                      else {
                          System.out.println("Status: Lista Vazia");
                      }
                      break;
            case 'c': if (ch == 'e') {
                          dado = lista.consultarListaEsquerda();
                      }
                      else {
                          dado = lista.consultarListaDireita();
                      }
                      System.out.println("Dado Consultado: " + dado);
                      break;
            }
        lista.imprimirLista();
        } while (op != 'f');
        System.exit(0);
    }
}
```

3.2.4 Lista Duplamente Encadeada

Na lista duplamente encadeada, cada elemento possui um elo para o **anterior** e o **posterior**, sendo que a lista pode ter ou não **descritor**.

Problema: Escrever um programa em Java que insere dados em uma **lista duplamente encadeada com descritor**.

Programa exemplo (19): O programa abaixo demonstra a inclusão e exibição de números inteiros (à esquerda e à direita) em uma lista duplamente encadeada com **descritor** usando as classes: **Nodo** e **ListaDuplaEncadeadaDescritor.**

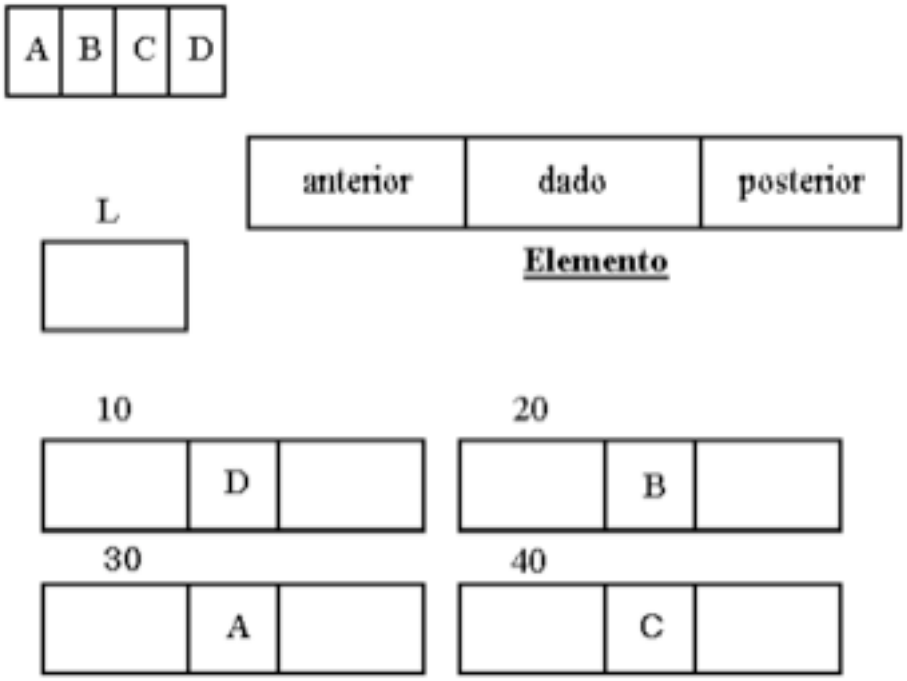
```
// ---------------------------------------------------------- Fonte: Nodo.java

package dados19;

public class Nodo {
    public Nodo anterior = null;
    public int dado;
    public Nodo posterior = null;
}

// ------------------------------------------------ Fonte: ListaDuplaEncadeadaDescritor.java

package dados19;

public class ListaDuplaEncadeadaDescritor {
    private Nodo primeiro = null;
    private int n = 0;
    private Nodo ultimo = null;

    // ------------------------------ inserirListaEsquerda

    public void inserirListaEsquerda(int valor)
    {
       Nodo aux = new Nodo();

       aux.dado = valor;
       if (n == 0) {
          primeiro = aux;
          ultimo = aux;
          aux.anterior = null;
       }
       else {
           aux.anterior = primeiro;
           primeiro = aux;
       }
       n++;
    }

    // ------------------------------ inserirListaDireita

    public void inserirListaDireita(int valor)
    {
       Nodo aux = new Nodo();
       Nodo tmp;

       aux.dado = valor;
       if (n == 0) {
          primeiro = aux;
          ultimo = aux;
       }
       else {
           tmp = ultimo;
           tmp.posterior = aux;
           aux.anterior = tmp;
           aux.posterior = null;
           ultimo = aux;
       }
       n++;
    }

    // ------------------------------ removerListaEsquerda
```

```
public int removerListaEsquerda() {
    Nodo aux = primeiro;
    int dado;

    if (n == 0) {
        return(0);
    }
    else {
        dado = aux.dado;
        n--;
        if (n == 0) {
            primeiro = null;
            ultimo = null;
        }
        else {
           primeiro = aux.posterior;
           primeiro.anterior = null;
        }
        return(dado);
    }
}

// ------------------------------- removerListaDireita

public int removerListaDireita() {
    Nodo ant, aux = ultimo;
    int dado;

    if (n == 0) {
        return(0);
    }
    else {
        dado = aux.dado;
        n--;
        if (n == 0) {
            primeiro = null;
            ultimo = null;
        }
        else {
           ant = aux.anterior;
           ultimo = ant;
           ant.posterior = null;
        }
        return(dado);
    }
}

// ------------------------------- consultarListaEsquerda

public int consultarListaEsquerda() {

    Nodo aux;
    int dado;

    aux = primeiro;

    if (aux != null) {
        return(primeiro.dado);
    }
    else
        return(0);
 }

// ------------------------------- consultarListaDireita

public int consultarListaDireita() {
```

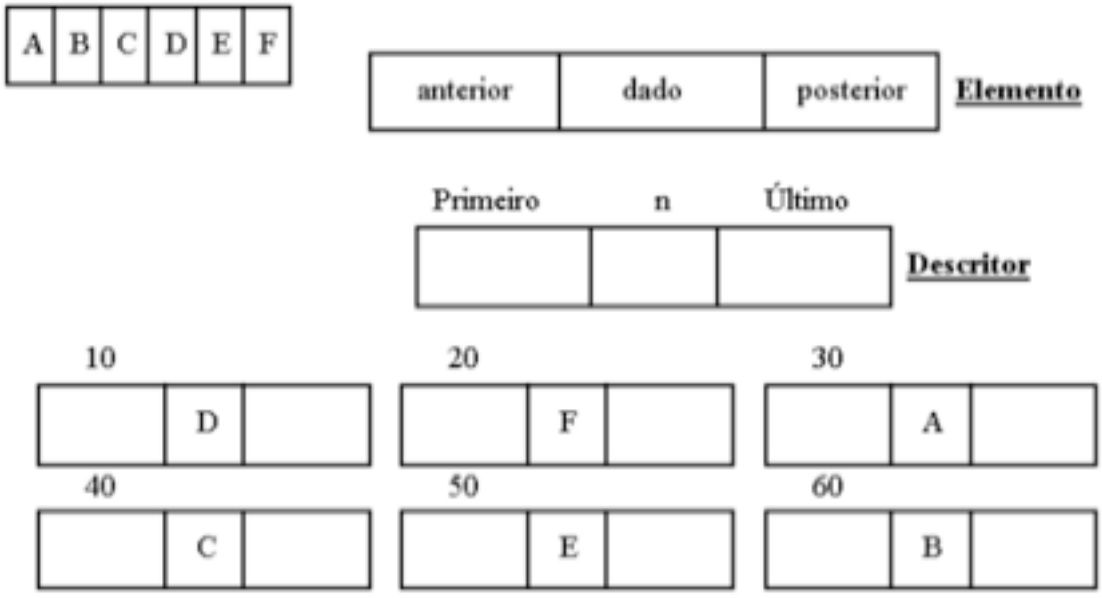
```
        Nodo aux;
        int dado;

        aux = ultimo;

        if (aux != null) {
            return(ultimo.dado);
        }
        else
            return(0);
     }

    // ------------------------------ imprimirLista

    public void imprimirLista() {
        Nodo aux = primeiro;

        System.out.print("Lista Duplamente Encadeada: ");
        if (aux != null) {
            while(aux != null) {
                System.out.print(aux.dado + " ");
                aux = aux.posterior;
            }
        }
        else {
              System.out.print("Vazia");
        }
        System.out.println();
    }
}

// ------------------------------------------------------------ Fonte: Dados19.java

package dados19;

import java.util.Scanner;

public class Dados19 {

    public static void main(String[] args) {
        ListaDuplaEncadeadaDescritor lista = new ListaDuplaEncadeadaDescritor();
        Scanner entrada = new Scanner(System.in);
        String s;
        char op, ch = 'e';
        int dado = 0;

        do {
            System.out.print("[I]ncluir, [C]onsultar, [R]emover ou [F]im: ");
            do {
                s = entrada.nextLine();
                op = s.charAt(0);
                op = Character.toLowerCase(op);
            } while (op != 'i' && op != 'c' && op != 'r' && op != 'f');
            if (op == 'i' || op == 'r' || op == 'c') {
                do {
                    System.out.print("[E]squerda ou [D]ireita: ");
                    s = entrada.nextLine();
                    ch = s.charAt(0);
                    ch = Character.toLowerCase(ch);
                } while (ch != 'e' && ch != 'd');
            }

            switch (op) {
                case 'i': System.out.print("Dado: ");
                          s = entrada.nextLine();
                          dado = Integer.parseInt(s);
                          if (dado != 0) {
                              if (ch == 'e') {
```

```
                                lista.inserirListaEsquerda(dado);
                            }
                            else {
                                lista.inserirListaDireita(dado);
                            }
                        }
                        break;
                case 'r': if (ch == 'e') {
                                dado = lista.removerListaEsquerda();
                            }
                            else {
                                dado = lista.removerListaDireita();
                            }
                        if (dado != 0) {
                            System.out.println("Nodo Removido: " + dado);
                        }
                        else {
                            System.out.println("Status: Lista Vazia");
                        }
                        break;
                case 'c': if (ch == 'e') {
                            dado = lista.consultarListaEsquerda();
                        }
                        else {
                            dado = lista.consultarListaDireita();
                        }
                        System.out.println("Dado Consultado: " + dado);
                        break;
            }
        lista.imprimirLista();
        } while (op != 'f');
        System.exit(0);
    }
}
```

a) Preencha os campos de elo com os valores adequados para que seja representada a seqüência (A, B, C, D): (sem descritor)



Preencha os campos de elo com os valores adequados para que seja representada a seqüência (A, B, C, D, E, F): (com descritor)



Vantagens e Desvantagens das Listas Duplamente Encadeadas

Vantagens

- Inserção e remoção de componentes sem movimentar os demais;
- Pode-se ter qualquer quantidade de elementos, limitado pela memória livre, pois cada elemento é alocado dinamicamente.

Desvantagens

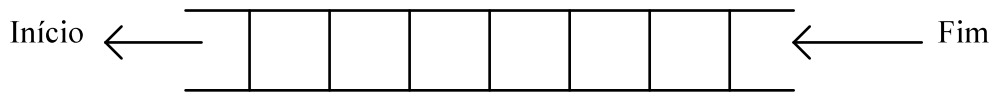
- Gerência de memória mais onerosa, **alocação** / **desalocação** para cada elemento;
- Procedimentos mais complexos;
- Processamento serial (Acesso Seqüencial).

3.2.5 Listas Lineares com Disciplina de Acesso

São tipos especiais de **Listas Lineares**, onde inserção, consulta e exclusão são feitas somente nos extremos. Estas listas lineares com disciplina de acesso são: **Filas**, **Pilhas** e **Deques.**

3.2.5.1 Filas

É uma **Lista Linear** na qual as inserções são feitas no fim e as exclusões e consultas no início da fila.



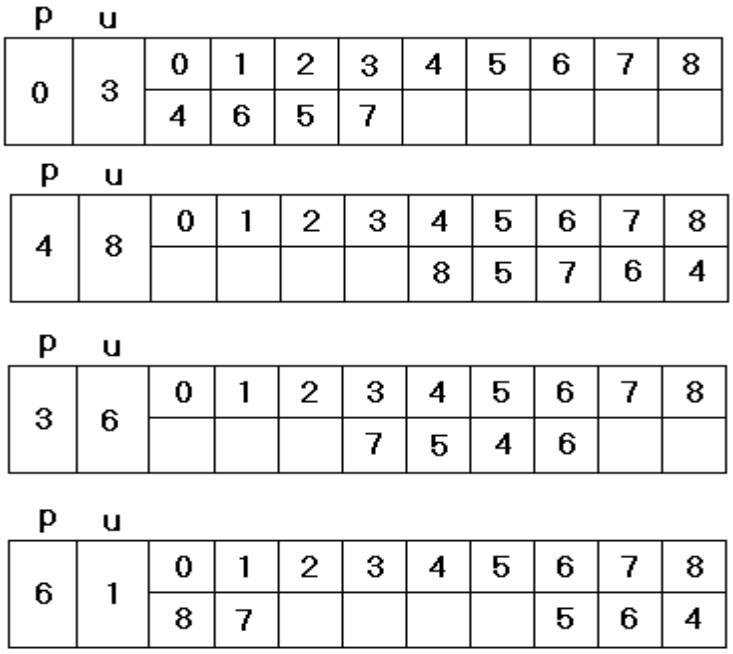
Critério de Utilização

FIFO - "**F**irst **I**n **F**irst **O**ut" (primeiro a entrar é o primeiro a sair)

Operações sobre Filas

```
criaFila();              Cria FILA Vazia
insereFila(i);           Insere o dado "i" no fim da FILA
erro = excluiFila ();    Exclui da FILA o primeiro elemento
dado = consultaFila ();  Copia em "j" primeiro elemento FILA
```

Erros nas operações sobre Filas: **FILA_CHEIA** ou **FILA_VAZIA**



Fila Circular

Problema: Escrever um programa em Java que **insere, exclui** e **consulta** dados (números inteiros) em uma **fila.**

Programa exemplo (20): O programa abaixo demonstra a inclusão, exclusão e consulta de números inteiros em uma **Fila** usando as classes: **Fila.**

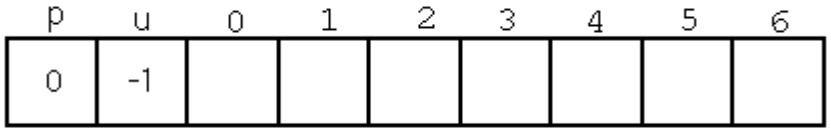
3.2.5.1.1 Fila com Vetor

```
// ------------------------------------------------------------ Fonte: Dados20.java

// Dados20.java

package dados20;

import java.util.Scanner;

public class Dados20 {

    public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);
        String s;
        Fila fila = new Fila();
        int valor;
        int erro = 0;
        char ch;

        fila.criaFila();
        do {
```

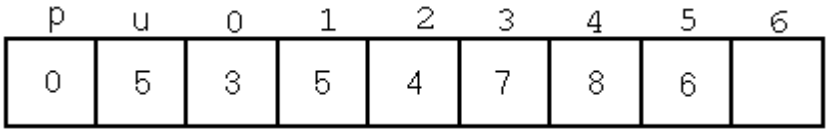
```
            System.out.printf("[I]ncluir, [E]xcluir, [C]onsultar ou [F]im? ");
            do {
                s = entrada.nextLine();
                ch = s.charAt(0);
                ch = Character.toLowerCase(ch);
           } while (ch != 'i' && ch != 'e' && ch != 'c' && ch != 'f');
     switch (ch)
     {
      case 'i': System.out.printf("Valor: ");
                s = entrada.nextLine();
               valor = Integer.parseInt(s);
               erro = fila.insereFila(valor);
               break;
      case 'e': valor = fila.excluiFila();
               break;
      case 'c': valor = fila.consultaFila();
               if (valor != 0) {
                   System.out.println("Primeiro da Fila: " + valor);
                }
                else {
                    System.out.println("ERRO: Fila Vazia");
                }
               break;
     }
     fila.exibeFila();
     if (erro != 0)
        fila.imprimeErroCircular(erro);
     } while (ch != 'f');
     System.exit(0);
    }
}

// ---------------------------------------------------------------- Fonte: Fila.java

package dados20;

public class Fila {
    final int SUCESSO = 0;
    final int FILA_CHEIA = 1;
    final int FILA_VAZIA = 2;
    private final int m = 7;
    private int primeiro;
    private int ultimo;
    private int []elem = new int[m];

    // ------------------------------------ criaFila

    public void criaFila() {
        primeiro = 0;
        ultimo = -1;
    }

    // ------------------------------------ insereFila

    public int insereFila(int dado) {
        if (ultimo == m - 1) {
            return(FILA_CHEIA);
        }
        else {
            ultimo++;
            elem[ultimo] = dado;
            return(SUCESSO);
        }
    }

    // ------------------------------- excluiFila

    public int excluiFila() {
```

```
        if (ultimo == -1) {
          return(FILA_VAZIA);
        }
        else {
            System.out.println("Dado Excluído: " + elem[primeiro]);
            primeiro++;
            if (primeiro > ultimo) {
                primeiro = 0;
               ultimo = -1;
            }
            return(SUCESSO);
       }
    }

    // ------------------------------ consultaFila

    public int consultaFila() {
        if (ultimo == -1) {
           return(0);
        }
        else {
           return(elem[primeiro]);
        }
    }

    // ------------------------------ exibeFila

    public void exibeFila() {
        System.out.print("Fila: ");
        if (ultimo != -1) {
            for (int i = primeiro;i <= ultimo; i++) {
              System.out.print(elem[i] + " ");
            }
         System.out.println();
        }
    }

    // ---------------------------------------- imprimeErro

    public void imprimeErro(int erro) {
        switch (erro) {
            case FILA_CHEIA: System.out.println("ERRO: Fila Cheia");
                  break;
            case FILA_VAZIA: System.out.println("ERRO: Fila Vazia");
                  break;
        }
    }
}
```

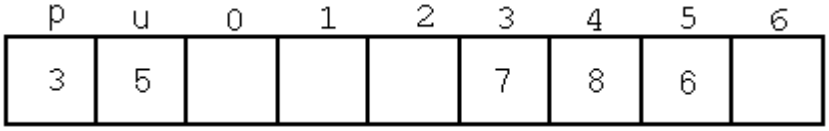
Exemplo: FILA VAZIA



Inclusão de: 3, 5, 4, 7, 8 e 6



Exclusão dos três primeiros elementos: 3, 5 e 4



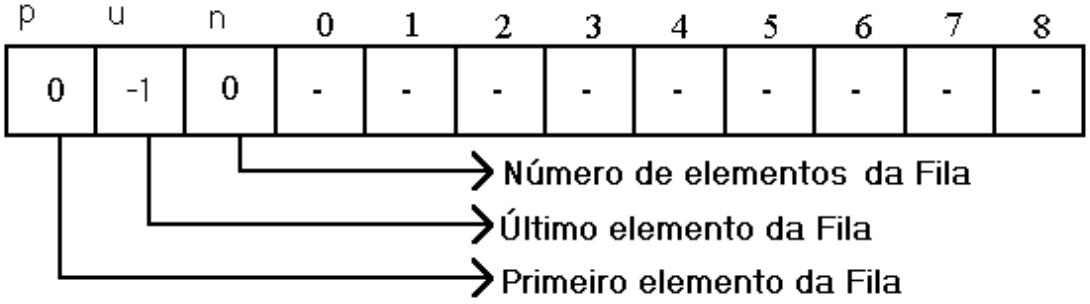
Problema com as Filas

A **reutilização** da fila depois que alguns elementos foram extraídos.

Solução deste Problema

Para reutilizar "as vagas" dos elementos já extraídos, deve-se usar uma **Fila Circular.**

3.2.5.1.2 Fila Circular



Problema: Escrever um programa em Java que **insere**, **exclui** e **consulta** dados (números inteiros) em uma **fila circular.**

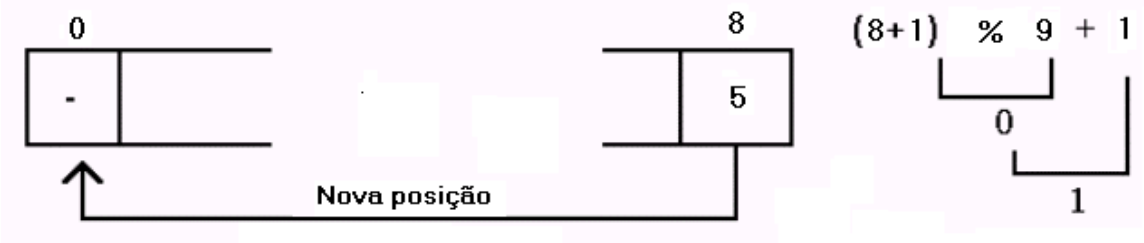
Programa exemplo (21): O programa abaixo demonstra a inclusão, exclusão e consulta de números inteiros em uma **Fila Circular** usando a classe: **FilaCircular.**

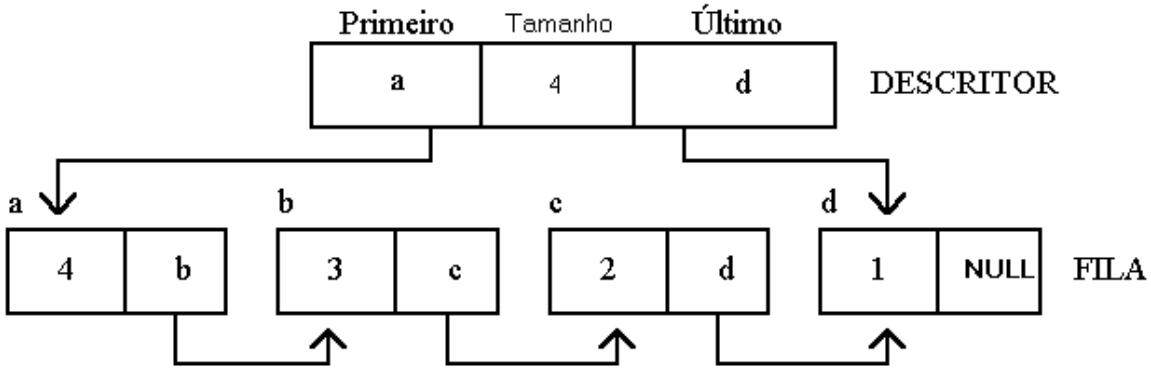
```
// ------------------------------------------------ Fonte: Dados21.java

package dados21;
```

```
import java.util.Scanner;

public class Dados21 {

    public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);
        String s;
        FilaCircular fila = new FilaCircular();
        int valor;
        int erro = 0;
        char ch;

        fila.criaFilaCircular();
        do {
            System.out.printf("[I]ncluir, [E]xcluir, [C]onsultar ou [F]im? ");
            do {
                s = entrada.nextLine();
                ch = s.charAt(0);
                ch = Character.toLowerCase(ch);
            } while (ch != 'i' && ch != 'e' && ch != 'c' && ch != 'f');
     switch (ch)
     {
      case 'i': System.out.printf("Valor: ");
                s = entrada.nextLine();
               valor = Integer.parseInt(s);
               erro = fila.insereFilaCircular(valor);
               break;
      case 'e': valor = fila.excluiFilaCircular();
               break;
      case 'c': valor = fila.consultaFilaCircular();
               if (valor != 0) {
                   System.out.println("Primeiro da Fila: " + valor);
                }
                else {
                    System.out.println("ERRO: Fila Vazia");
                }
               break;
     }
     fila.exibeFilaCircular();
     if (erro != 0)
        fila.imprimeErroCircular(erro);
     } while (ch != 'f');
     System.exit(0);
    }
}

// ---------------------------------------------------- Fonte: FilaCircular.java

package dados21;

public class FilaCircular {
    final int SUCESSO = 0;
    final int FILA_CHEIA = 1;
    final int FILA_VAZIA = 2;
    private final int m = 5;
    private int primeiro;
    private int tamanho;
    private int ultimo;
    private int []elem = new int[m];

    // ------------------------------------ Cria_FilaCircular

    public void criaFilaCircular() {
        primeiro = 0;
        tamanho = 0;
        ultimo = -1;
    }
```

```
    // ---------------------------------- insereFilaCircular

    public int insereFilaCircular(int dado) {
        if (tamanho == m) {
     return(FILA_CHEIA);
        }
        else {
            tamanho++;
            ultimo = (ultimo + 1) % m;
            elem[ultimo] = dado;
            return(SUCESSO);
        }
    }

    // ------------------------------- excluiFilaCircular

    public int excluiFilaCircular() {
        if (tamanho == 0) {
     return(FILA_VAZIA);
        }
        else {
            System.out.println("Dado Excluído: " + elem[primeiro]);
            tamanho--;
            primeiro = (primeiro + 1) % m;
            return(SUCESSO);
        }
    }

    // ------------------------------- consultaFilaCircular

    public int consultaFilaCircular() {
        if (ultimo == -1) {
           return(0);
        }
        else {
           return(elem[primeiro]);
        }
    }

    // ------------------------------- exibeFilaCircular

    public void exibeFilaCircular() {
        int t = primeiro;

        System.out.print("Fila Circular: ");
        if (tamanho != 0) {
            for (int i = 1;i <= tamanho; i++) {
              System.out.print(elem[t] + " ");
              t = (t + 1) % m;
            }
         System.out.println();
        }
    }

    // ---------------------------------------- imprimeErroCircular

    public void imprimeErroCircular(int erro) {
        switch (erro) {
            case FILA_CHEIA: System.out.println("ERRO: Fila Circular Cheia");
                  break;
            case FILA_VAZIA: System.out.println("ERRO: Fila Circular Vazia");
                  break;
        }
    }
}
```

Como calcular a nova posição:





3.2.5.1.3 Fila com Alocação Dinâmica

Problema: Escrever um programa em Java que insere, exclui e consulta dados (números inteiros) em uma **fila alocada dinamicamente.**

Programa exemplo (22): O programa abaixo demonstra a inclusão, exclusão e consulta de números inteiros em uma **Fila** alocada dinamicamente usando as classes: **Nodo** e **FilaDinamica.**

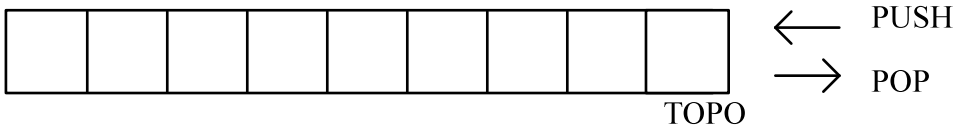
```
// ---------------------------------------------- Fonte: Nodo.java

package dados22;

public class Nodo {
    public int info;
    public Nodo seg;
}

// ---------------------------------------------- Fonte: Dados22.java

package dados22;

public class FilaDinamica {
    final int SUCESSO = 0;
    final int FILA_VAZIA = 1;
    private Nodo primeiro;
    private int tamanho;
    private Nodo ultimo;

    // ---------------------------------------------- Cria_FilaDinamica
```

```
    public void criaFilaDinamica() {
        primeiro = null;
        tamanho = 0;
        ultimo = null;
    }

    // ---------------------------------------------- inserirFilaDinamica

    public int inserirFilaDinamica(int dado) {
        Nodo t = new Nodo();

        t.info = dado;
        t.seg = null;
        tamanho = tamanho + 1;
        if (ultimo != null) {
            ultimo.seg = t;
        }
        ultimo = t;
        if (primeiro == null) {
            primeiro = t;
        }
//        System.out.println("FILA: " + primeiro + " " + tamanho + " " + ultimo);
        return(SUCESSO);
    }

    // ---------------------------------------------- excluirFilaDinamica

    public int excluirFilaDinamica() {
        Nodo t = new Nodo();

        if (primeiro == null) {
           return(FILA_VAZIA);
        }
        else {
            t = primeiro;
            tamanho = tamanho - 1;
            primeiro = t.seg;
            if (primeiro == null) {
                ultimo = null;
              return(FILA_VAZIA);
            }
        return(SUCESSO);
        }
    }

    // ---------------------------------------------- consultarFilaDinalica

    public int consultarFilaDinamica() {
        if (primeiro == null) {
            return(FILA_VAZIA);
        }
        else {
            return(primeiro.info);
        }
    }

    // ---------------------------------------------- exibirFilaDinalica

    public void exibirFilaDinamica() {
        Nodo t = primeiro;

        System.out.print("Fila Dinâmica: ");
        if (primeiro == null) {
            System.out.print("Vazia");
        }
        else {
          while (t != null) {
```

```
            System.out.print(t.info + " ");
            t = t.seg;
          }
        }
        System.out.println();
    }

    // ------------------------------------------- imprimeErroDinamica

    public void imprimeErroDinamica(int erro) {
        switch (erro) {
            case FILA_VAZIA: System.out.println("ERRO: Fila Vazia");
                     break;
        }
    }
}

// ------------------------------------------------------ Fonte: Dados22.java

package dados22;

import java.util.Scanner;

public class Dados22 {

    public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);
        String s;
        FilaDinamica fila = new FilaDinamica();
        int valor, erro = 0;
        char tecla;

        fila.criaFilaDinamica();
        do {
            System.out.print("[I]ncluir, [E]xcluir, [C]onsultar ou [F]im? ");
            do {
                s = entrada.nextLine();
                tecla = s.charAt(0);
               tecla = Character.toLowerCase(tecla);
            } while (tecla != 'i' && tecla != 'e' && tecla != 'c' && tecla != 'f');
            switch (tecla) {
                case 'i': System.out.print("Valor: ");
                    s = entrada.nextLine();
                   valor = Integer.parseInt(s);
                   erro = fila.inserirFilaDinamica(valor);
                   break;
                case 'e': erro = fila.excluirFilaDinamica();
                   break;
                case 'c': valor = fila.consultarFilaDinamica();
                    System.out.println("Valor Consultado: " + valor);
                    break;
              }
         if (erro != 0) {
             fila.imprimeErroDinamica(erro);
         }
         fila.exibirFilaDinamica();
    } while (tecla != 'f');
    }
}
```

3.2.5.2 Pilhas

É uma **Lista Linear** na qual as inserções, exclusões e consultas são feitas em um mesmo extremo (**Topo**).



Critério de Utilização

LIFO - "**L**ast **I**n **F**irst **O**ut" (último a entrar é o primeiro a sair)

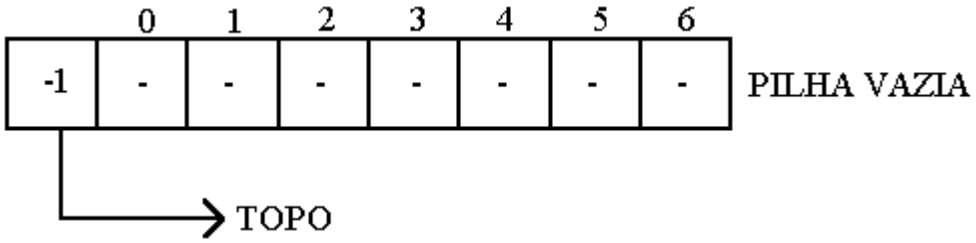
Operações sobre Pilhas

```
criaPilha();              Cria pilha Vazia
erro = push(i);           Empilha o dado "i" no topo da Pilha
erro = pop();             Desempilha o primeiro elemento
erro = consultaPilha();   Exibe o primeiro elemento
```

Identificadores da Pilha

```
B(p)      Base da Pilha
T(p)      Topo da Pilha
L(p)      Limite da Pilha
```

3.2.5.2.1 Pilha com Vetor



Problema: Escreva um programa em Java que insere (***push***), exclui (***pop***) e consulta dados (números inteiros) em uma **Pilha alocada estaticamente.**

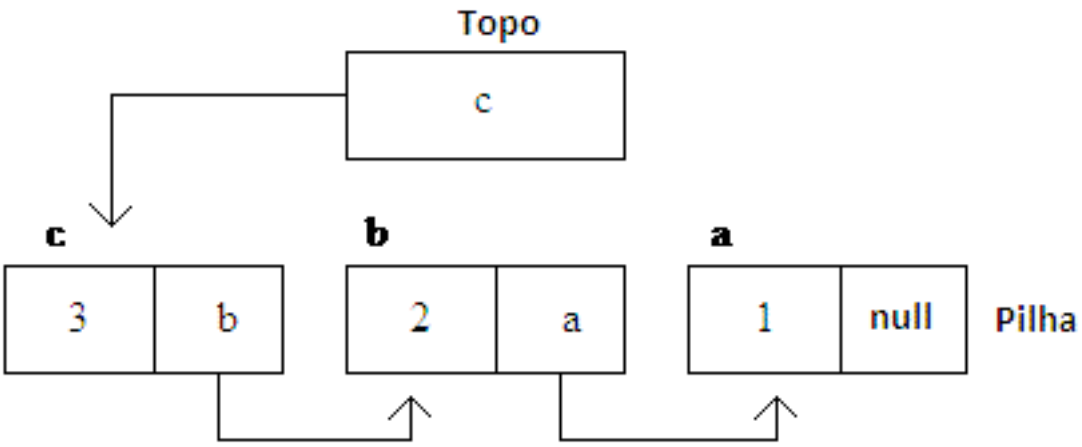
Programa exemplo (23): O programa abaixo demonstra a inclusão, exclusão e consulta de números inteiros em uma **Pilha** alocada estaticamente usando as classes: **PilhaVetor.**

```
// ------------------------------------------------- Fonte: Dados23.java

package dados23;

import java.util.Scanner;

public class Dados23 {

    public static void main(String[] args) {
```

```
        Scanner entrada = new Scanner(System.in);
        String s;
        PilhaVetor pilha = new PilhaVetor();
        int valor;
        int erro = 0;
        char ch;

        pilha.criaPilha();
        do {
            System.out.printf("[P]ush, P[o]p, [C]onsultar ou [F]im? ");
            do {
                s = entrada.nextLine();
                ch = s.charAt(0);
                ch = Character.toLowerCase(ch);
            } while (ch != 'p' && ch != 'o' && ch != 'c' && ch != 'f');
     switch (ch)
     {
      case 'p': System.out.printf("Valor: ");
                s = entrada.nextLine();
               valor = Integer.parseInt(s);
               erro = pilha.push(valor);
               break;
      case 'o': erro = pilha.pop();
               break;
      case 'c': erro = pilha.consultaPilha();
               break;
     }
     pilha.exibePilha();
     if (erro != 0)
       pilha.imprimeErro(erro);
     } while (ch != 'f');
     System.exit(0);
    }
}

// ------------------------------------------------------ Fonte: Dados23.java

package dados23;

public class PilhaVetor {
    final int SUCESSO = 0;
    final int PILHA_CHEIA = 1;
    final int PILHA_VAZIA = 2;
    private final int m = 7;
    private int topo;
    private int []elem = new int[m];

    // ------------------------------------ criaPilha

    public void criaPilha() {
        topo = -1;
    }

    // ------------------------------------ push

    public int push(int dado) {
        if (topo == m - 1) {
            return(PILHA_CHEIA);
        }
        else {
            topo = topo + 1;
            elem[topo] = dado;
            return(SUCESSO);
        }
    }

    // ------------------------------- pop
```

```
    public int pop() {
        if (topo == -1) {
            return(PILHA_VAZIA);
        }
        else {
            System.out.println("Pop: " + elem[topo]);
            topo = topo - 1;
            return(SUCESSO);
        }
    }

    // ------------------------------ consultaPilha

    public int consultaPilha() {
        if (topo == -1) {
            return(PILHA_VAZIA);
        }
        else {
            System.out.println("Pop: " + elem[topo]);
            return(SUCESSO);
        }
    }

    // ------------------------------ exibePilha

    public void exibePilha() {
        System.out.print("Pilha: ");
        if (topo != -1) {
            for (int i = topo;i >= 0; i--) {
              System.out.print(elem[i] + " ");
            }
         System.out.println();
        }
    }

    // ---------------------------------------- imprimeErro

    public void imprimeErro(int erro) {
        switch (erro) {
            case PILHA_CHEIA: System.out.println("ERRO: Pilha Cheia");
                  break;
            case PILHA_VAZIA: System.out.println("ERRO: Pilha Vazia");
                  break;
        }
    }
}
```

3.2.5.2.2 Pilha com Alocação Dinâmica



Problema: Escreva um programa em Java que inclui, exclui e consulta dados em uma **Pilha Encadeada.**

Programa exemplo (24): O programa abaixo demonstra a inclusão, exclusão e consulta em uma Pilha alocada dinamicamente.

```
// ------------------------------------------------------------ Fonte: Nodo.java

package dados24;

public class Nodo {
    public int info;
    public Nodo seg = null;
}

// ------------------------------------------------------------ Fonte: PilhaDinamica.java

package dados24;

public class PilhaDinamica {
    final int SUCESSO = 0;
    final int PILHA_VAZIA = 2;
    private Nodo topo;

    // ---------------------------------------------- criaPilha

    public void criaPilha() {
        topo = null;
    }

    // ---------------------------------------------- push

    public int push(int dado) {
        Nodo t = new Nodo();

        t.info = dado;
        t.seg = topo;
        topo = t;
        return(SUCESSO);
    }
```

```
    // ---------------------------------------------- pop

    public int pop() {
        Nodo t = new Nodo();

        if (topo == null) {
            return(PILHA_VAZIA);
        }
        else {
            t = topo;
            System.out.println("Dado no Topo: " + t.info);
            topo = t.seg;
            return(SUCESSO);
        }
    }

    // ---------------------------------------------- consultaPilha

    public int consultaPilha() {
        Nodo t = new Nodo();

        if (topo == null) {
           return(PILHA_VAZIA);
        }
        else {
            t = topo;
            System.out.println("Topo: " + t.info);
            return(SUCESSO);
        }
    }

    // ---------------------------------------------- imprimeErro

    public void imprimeErro(int erro) {
        switch (erro) {
            case PILHA_VAZIA: System.out.println("ERRO: Pilha Vazia");
                       break;
        }
    }

    // ---------------------------------------------- exibePilha

    public void exibePilha() {
        Nodo t = new Nodo();

        System.out.print("Pilha: ");
        if (topo == null) {
            System.out.print("Vazia");
        }
        else {
            t = topo;
            while (t != null) {
                System.out.print(t.info + " ");
               t = t.seg;
           }
        }
        System.out.println();
    }
}

// ---------------------------------------------------------------- Fonte: Dados24.java

package dados24;

import java.util.Scanner;

public class Dados24 {
```

```
    public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);
        String s;
        PilhaDinamica pilha = new PilhaDinamica();
        int valor, erro = 0;
        char tecla;

        pilha.criaPilha();
        do {
            pilha.exibePilha();
            System.out.print("[P]ush, p[O]p, [C]onsultar ou [F]im?");
            do {
                s = entrada.nextLine();
               tecla = s.charAt(0);
                tecla = Character.toLowerCase(tecla);
            } while (tecla != 'p' && tecla != 'o' && tecla != 'c' && tecla != 'f');
            switch (tecla) {
                case 'p': System.out.print("Valor a ser Empilhado: ");
                          s = entrada.nextLine();
                          valor = Integer.parseInt(s);
                          erro = pilha.push(valor);
                          break;
                case 'o': erro = pilha.pop();
                          break;
                case 'c': erro = pilha.consultaPilha();
                          break;
            }
            if (erro != 0) {
                pilha.imprimeErro(erro);
            }
        } while (tecla != 'f');
        System.exit(0);
    }
}
```

Problema: Escreva um programa em Java que é um Analisador de Expressões, ou seja, resolve operações do tipo: (3*(4+5))

Programa exemplo (25): O programa se encontra no capítulo 8.2.

3.2.5.3 Deque ("*Double-Ended QUEue*")

É uma fila de duas extremidades. As inserções, consultas e retiradas são permitidas nas duas extremidades.

Deque de Entrada Restrita

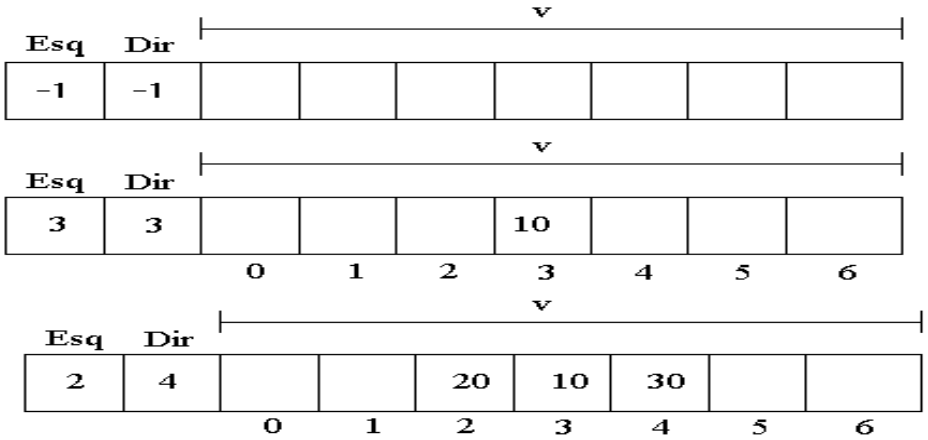
A inserção só pode ser efetuada ou no início ou no final da lista.

Deque de Saída Restrita

A retirada só pode ser efetuada ou no início ou no final da lista.

Condição Inicial: Esquerda (Esq) e Direita (Dir) no meio do vetor

```
Esq = (m DIV 2 + 1
Dir = (m DIV 2) + 1
```



Deque Vazio

Esq = -1
Dir = -1

Deque Cheio

Esq = 0
Dir = m-1

Cálculo do número de elementos do Deque

```
número_de_elementos = Dir - Esq + 1;
```

Problema: Escreva um programa em Java que inclui, exclui e consulta dados em um **Deque.**

Programa exemplo (26): O programa abaixo demonstra um **Deque** usando um vetor.

```
// ---------------------------------------------------------------- Fonte: Deque.java

package dados26;

public class Deque {
    final int m = 9;
    final int SUCESSO = 0;
```

```
final int DEQUE_ESQUERDO_CHEIO = 1;
final int DEQUE_DIREITO_CHEIO = 2;
final int DEQUE_VAZIO = 3;
int esq;
int dir;
int []v = new int[m];

// ------------------------------------ criaDeque

public void criaDeque() {
    esq = -1;
    dir = -1;
}

// ------------------------------------ incluiDequeEsquerda

public int incluiDequeEsquerda(int dado) {
    if (esq == 0) {
        return(DEQUE_ESQUERDO_CHEIO);
    }
    else {
        if (esq == -1) {
            esq = m / 2;
           dir = esq;
        }
    else {
     esq = esq - 1;
    }
  v[esq] = dado;
  return(SUCESSO);
  }
}

// ------------------------------------ incluiDequeDireita

public int incluiDequeDireita(int dado) {
    if (dir == m - 1) {
        return(DEQUE_DIREITO_CHEIO);
    }
    else {
        if (dir == -1) {
            dir = m / 2;
           esq = dir;
       }
        else {
            dir = dir + 1;
        }
  v[dir] = dado;
  return(SUCESSO);
  }
}

// ------------------------------------ excluiDequeEsquerda

 public int excluiDequeEsquerda() {
     if (esq == -1) {
         return(DEQUE_VAZIO);
     }
     else {
         System.out.println("Dado Removido à Esquerda: " + v[esq]);
         esq = esq + 1;
         if (esq > dir) {
           criaDeque();
         }
     return(SUCESSO);
     }
}
```

```
    // ------------------------------------ excluiDequeDireita

    public int excluiDequeDireita() {
        if (dir == -1) {
           return(DEQUE_VAZIO);
        }
        else {
            System.out.println("Dado excluído Direita do Deque:" + v[dir]);
            dir = dir - 1;
            if (dir < esq) {
               criaDeque();
            }
      return(SUCESSO);
     }
    }

    // ------------------------------------ consultaDequeEsquerda

    public int consultaDequeEsquerda() {
        if (esq == -1) {
           return(DEQUE_VAZIO);
        }
        else {
          System.out.println("Valor Consultado Esquerda Deque: " + v[esq]);
      return(SUCESSO);
        }
    }

    // ------------------------------------ ConsultaDequeDireita

    public int consultaDequeDireita() {
        if (dir == -1) {
           return(DEQUE_VAZIO);
        }
        else {
            System.out.println("Valor Consultado Direita Deque: " + v[dir]);
            return(SUCESSO);
        }
    }

    // ------------------------------------ exibeDeque

    public void exibeDeque() {
        System.out.print("Deque: ");
        if (esq == -1) {
           System.out.print("Vazio");
        }
        else {
           for (int i = esq;i <= dir;i++) {
             System.out.print(v[i] + " ");
           }
        }
        System.out.println();
    }

    // ------------------------------------ imprimeErro

    public void imprimeErro(int erro) {
        switch (erro) {
            case DEQUE_ESQUERDO_CHEIO: System.out.println("ERRO: Deque Cheio à Esquerda");
                                       break;
            case DEQUE_DIREITO_CHEIO: System.out.println("ERRO: Deque Cheio à Direita");
                                      break;
           case DEQUE_VAZIO: System.out.println("ERRO: Deque Vazio");
                             break;
      }
    }
}
```

```
// ------------------------------------------------------------ Fonte: Dados26.java

package dados26;

import java.util.Scanner;

public class Dados26 {

    public static void main(String[] args) {
        Deque deque = new Deque();
        Scanner entrada = new Scanner(System.in);
        String s;
        int valor;
        char ch, op;
        int erro = 0;

        deque.criaDeque();
        do {
            deque.exibeDeque();
            System.out.print("[I]nclui, [E]xclui, [C]onsulta ou [F]im: ");
            do {
                s = entrada.nextLine();
                op = s.charAt(0);
               op = Character.toLowerCase(op);
           } while (op != 'i' && op != 'e' && op != 'c' && op != 'f');
            if (op == 'i' || op == 'e' || op == 'c') {
                System.out.print("[E]squerda ou [D]ireita: ");
               do {
                    s = entrada.nextLine();
                    ch = s.charAt(0);
                  ch = Character.toLowerCase(ch);
               } while (ch != 'e' && ch != 'd');
               switch (op) {
                    case 'i': System.out.print("Valor: ");
                              s = entrada.nextLine();
                             valor = Integer.parseInt(s);
                             if (valor != 0) {
                                  switch (ch) {
                                      case 'e': erro = deque.incluiDequeEsquerda(valor);
                                                break;
                                        case 'd': erro = deque.incluiDequeDireita(valor);
                                                break;
                                    }
                                }
                             break;
                  case 'e': switch (ch) {
                                case 'e': erro = deque.excluiDequeEsquerda();
                                          break;
                                  case 'd': erro = deque.excluiDequeDireita();
                                          break;
                              }
                             break;
                  case 'c': switch (ch) {
                                case 'e': erro = deque.consultaDequeEsquerda();
                                          break;
                                case 'd': erro = deque.consultaDequeDireita();
                                          break;
                              }
                             break;
              }
          if (erro != 0) {
            deque.imprimeErro(erro);
          }
        }
    } while (op != 'f');
    }
}
```

4. Pesquisa de Dados

Uma operação complexa e trabalhosa é a consulta em tabelas. Normalmente uma aplicação envolve grande quantidade de dados que são armazenadas em **Tabelas.**

As tabelas são compostas de registros (normalmente possui uma chave), e os registros de campos.

Tabela

Chave	Nome	Altura	Peso
0	Francisco	1.75	80.4
1	Renato	1.80	76.9
2	Paulo	1.67	87.5
3	Ricardo	1.86	57.8

Registro

Campos

4.1 Pesquisa Seqüencial

Método mais simples de pesquisa em tabela, consiste em uma varredura seqüencial, sendo que cada campo é comparado com o valor que está sendo procurado. Esta pesquisa termina quando for achado o valor desejado ou quando chegar o final da tabela.

$$Número\_Médio\_de\_Comparações = (n + 1) / 2$$

$n \rightarrow$ Número de Elementos

Problema: Escreva um programa em Java que faz uma **Busca Seqüencial** em uma estrutura que possui os campos: **chave, nome, altura** e **peso.**

Programa exemplo (27): O programa demonstra uma **busca sequencial** em uma **tabela.**

```
// ------------------------------------------------------------- Fonte: Tabela.java

package dados27;

import java.util.Scanner;

public class Tabela {
        private final int max = 10;
        private int [] chave = new int[max];;
        private String [] nome = new String[max];
       private float [] peso = new float[max];;
       private float [] altura = new float[max];
        private int n = -1;

        // ---------------------------------------------- pesquisaSequencial

        public int pesquisaSequencial(int ch) {
            int key = -1;

            for (int i = 0;i <= n;i++) {
                if (chave[i] == ch) {
                    key = i;
                   break;
                }
           }
            return(key);
        }

        // ---------------------------------------------- exibeTabela

        public void exibeTabela(int key) {
            if (key != -1) {
                System.out.println("Chave: " + chave[key]);
               System.out.println("Nome: " + nome[key]);
               System.out.println("Peso: " + peso[key]);
               System.out.println("Altura: " + altura[key]);
           }
            else {
                System.out.println("Erro: Chave Inexistente");
            }
        }

        // ---------------------------------------------- entradaTabela

        public void entradaTabela() {
            Scanner entrada = new Scanner(System.in);
            String s;
            char tecla;

            do {
                n++;
                System.out.printf("Chave: ");
                s = entrada.nextLine();
                chave[n] = Integer.parseInt(s);
                System.out.printf("Nome: ");
                nome[n] = entrada.nextLine();
                System.out.printf("Peso: ");
                s = entrada.nextLine();
                peso[n] = Integer.parseInt(s);
                System.out.printf("Altura: ");
                s = entrada.nextLine();
                altura[n] = Float.parseFloat(s);
                System.out.printf("Continua [S/N] ?");
                do {
```

```
                    s = entrada.nextLine();
                    tecla = s.charAt(0);
                   tecla = Character.toLowerCase(tecla);
                } while (tecla != 's' && tecla != 'n');
            } while (tecla == 's' && n < max);
        }
}
```

```
// ------------------------------------------------------------ Fonte: Dados27.java

package dados27;

import java.util.Scanner;

public class Dados27 {

    public static void main(String[] args) {
        Tabela tabela = new Tabela();
        Scanner entrada = new Scanner(System.in);
        String s;
        int key, chave;

        tabela.entradaTabela();
        do {
            System.out.print("Chave para consulta [0 - Sair]: ");
            s = entrada.nextLine();
            key = Integer.parseInt(s);
            chave = tabela.pesquisaSequencial(key);
            tabela.exibeTabela(chave);
        } while (key != 0);
    }
}
```

Observação: A **Pesquisa Sequencial** apresenta desempenho melhor se a tabela estiver ordenada pela chave de acesso:

Problema: Escreva um programa em Java que faz uma **Busca Seqüencial**, em uma estrutura ordenada por chave, que possui os campos: **chave**, **nome**, **altura** e **peso**.

Programa exemplo (28): O programa demonstra uma **busca sequencial** em uma **tabela ordenada por chave.**

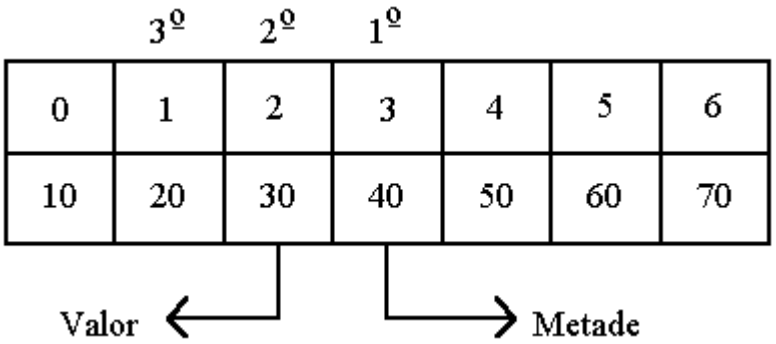
```
// ------------------------------------------------- Fonte: Tabela.java

package dados28;

import java.util.Scanner;

public class Tabela {
        private final int max = 10;
        private int [] chave = new int[max];;
        private String [] nome = new String[max];
       private float [] peso = new float[max];;
       private float [] altura = new float[max];
        private int n = -1;

        // ---------------------------------------------- pesquisaSequencial

        public int pesquisaSequencial(int ch) {
            int key = -1;

            for (int i = 0;i <= n;i++) {
```

```
        if (chave[i] == ch) {
            key = i;
           break;
        }
    }
    return(key);
}

// ------------------------------------------- ordenaTabela

public void ordenaTabela() {
    for (int i = 0;i < n-1;i++) {
        for (int j = i+1;j < n;j++) {
            if (chave[i] > chave[j]) {
                int temp = chave[i];
                chave[i] = chave[j];
                chave[j] = temp;
                String s = nome[i];
                nome[i] = nome[j];
                nome[j] = s;
                float f = peso[i];
                peso[i] = peso[j];
                peso[j] = f;
                f = altura[i];
                altura[i] = altura[j];
                altura[j] = f;
            }
        }
    }
}

// ------------------------------------------- exibeTabela

public void exibeTabela(int key) {
    if (key != -1) {
        System.out.println("Chave: " + chave[key]);
       System.out.println("Nome: " + nome[key]);
       System.out.println("Peso: " + peso[key]);
       System.out.println("Altura: " + altura[key]);
    }
    else {
        System.out.println("Erro: Chave Inexistente");
    }
}

// ------------------------------------------- entradaTabela

public void entradaTabela() {
    Scanner entrada = new Scanner(System.in);
    String s;
    char tecla;

    do {
        n++;
        System.out.printf("Chave: ");
        s = entrada.nextLine();
        chave[n] = Integer.parseInt(s);
        System.out.printf("Nome: ");
        nome[n] = entrada.nextLine();
        System.out.printf("Peso: ");
        s = entrada.nextLine();
        peso[n] = Integer.parseInt(s);
        System.out.printf("Altura: ");
        s = entrada.nextLine();
        altura[n] = Float.parseFloat(s);
        System.out.printf("Continua [S/N] ?");
        do {
            s = entrada.nextLine();
```

```
                    tecla = s.charAt(0);
                    tecla = Character.toLowerCase(tecla);
                } while (tecla != 's' && tecla != 'n');
            } while (tecla == 's' && n < max);
        }
}

// ---------------------------------------------------- Fonte: Dados28.java

package dados28;

import java.util.Scanner;

public class Dados28 {

    public static void main(String[] args) {
        Tabela tabela = new Tabela();
        Scanner entrada = new Scanner(System.in);
        String s;
        int key, chave;

        tabela.entradaTabela();
        tabela.ordenaTabela();
        do {
            System.out.print("Chave para consulta [0 - Sair]: ");
            s = entrada.nextLine();
            key = Integer.parseInt(s);
            chave = tabela.pesquisaSequencial(key);
            tabela.exibeTabela(chave);
        } while (key != 0);
        System.exit(0);
    }
}
```

4.2 Pesquisa Binária

Método de Pesquisa que só pode ser aplicada em tabelas ordenadas.



O método consiste na comparação do "valor" com a chave localizada na metade da tabela, pode ocorrer:

valor = chave..... chave localizada
valor < chave..... chave está na primeira metade (esquerda)
valor > chave..... chave está na segunda metade (direita)

```
metade = n / 2;
```

A cada comparação, a área de pesquisa é reduzida a metade do número de elementos.

O número máximo de comparações será:

$$nc = \log_2 n + 1$$

Problema: Escreva um programa em Java que faz uma **Pesquisa Binária** em uma tabela de números inteiros.

Programa exemplo (29): O programa demonstra uma **busca binária** em uma **tabela ordenada por chave.**

```
// ---------------------------------------- Fonte: Tabela.java

package dados29;

import java.util.Scanner;

public class Tabela {
    private final int max = 10;
    private int [] chave = new int[max];;
    private String [] nome = new String[max];
    private float [] peso = new float[max];;
    private float [] altura = new float[max];
    private int n = -1;

    // ---------------------------------------------- pesquisaBinaria

    public int pesquisaBinaria(int valor) {
        int indice = -1;
        int inic, fim, metade;

        inic = 0;
        fim = (n - 1) + 1;
        metade = n / 2;
        do {
            if (valor == chave[metade]) {
                indice = metade;
               break;
            }
            else {
                if (valor < chave[metade]) {
                    fim = metade - 1;
                   metade = (fim + inic) / 2;
                }
                else {
                    inic = metade + 1;
                   metade = (fim + inic) / 2;
                }
            }
        } while (indice == -1 && inic <= fim);
        return(indice);
    }
```

```
    // ---------------------------------------------- ordenaTabela

    public void ordenaTabela() {
            for (int i = 0;i < n;i++) {
                for (int j = i+1;j <= n;j++) {
                    if (chave[i] > chave[j]) {
                        int temp = chave[i];
                        chave[i] = chave[j];
                        chave[j] = temp;
                        String s = nome[i];
                        nome[i] = nome[j];
                        nome[j] = s;
                        float f = peso[i];
                        peso[i] = peso[j];
                        peso[j] = f;
                        f = altura[i];
                        altura[i] = altura[j];
                        altura[j] = f;
                    }
                }
            }
    }

        // ---------------------------------------------- exibeTabela

        public void exibeTabela(int key) {
            if (key != -1) {
                System.out.println("Chave: " + chave[key]);
               System.out.println("Nome: " + nome[key]);
               System.out.println("Peso: " + peso[key]);
               System.out.println("Altura: " + altura[key]);
            }
            else {
                System.out.println("Erro: Chave Inexistente");
            }
        }

        // ---------------------------------------------- entradaTabela

        public void entradaTabela() {
            Scanner entrada = new Scanner(System.in);
            String s;
            char tecla;

            do {
                n++;
                System.out.printf("Chave: ");
                s = entrada.nextLine();
                chave[n] = Integer.parseInt(s);
                System.out.printf("Nome: ");
                nome[n] = entrada.nextLine();
                System.out.printf("Peso: ");
                s = entrada.nextLine();
                peso[n] = Integer.parseInt(s);
                System.out.printf("Altura: ");
                s = entrada.nextLine();
                altura[n] = Float.parseFloat(s);
                System.out.printf("Continua [S/N] ?");
                do {
                    s = entrada.nextLine();
                    tecla = s.charAt(0);
                   tecla = Character.toLowerCase(tecla);
               } while (tecla != 's' && tecla != 'n');
           } while (tecla == 's' && n < max);
        }
}

// ------------------------------------------------------------ Fonte: Dados29.java
```

```
package dados29;

import java.util.Scanner;

public class Dados29 {

    public static void main(String[] args) {
        Tabela tabela = new Tabela();
        Scanner entrada = new Scanner(System.in);
        String s;
        int key, chave;

        tabela.entradaTabela();
        tabela.ordenaTabela();
        do {
            System.out.print("Chave para consulta [0 - Sair]: ");
            s = entrada.nextLine();
            key = Integer.parseInt(s);
            chave = tabela.pesquisaBinaria(key);
            tabela.exibeTabela(chave);
        } while (key != 0);
        System.exit(0);
    }
}
```

4.3 Cálculo de Endereço (*Hashing*)

Além de um método de pesquisa, este método é também um método de organização física de tabelas (***Classificação***). Onde cada dado de entrada é armazenado em um endereço previamente calculado (através de uma função), desta forma, o processo de busca é igual ao processo de entrada, ou seja, eficiente.

Um dos problemas é definir bem o tipo de função a ser usada, pois normalmente as funções geram endereços repetidos. A eficiência deste método depende do tipo de função.

Numa tabela com ***n*** elementos com valores na faixa de [0..max] pode ser utilizada a seguinte a função:

```
endereço = entrada % n;
```

Onde: n é o número de elementos.

Exemplo:

n = 53
entrada: [0..1000]

Entrada	383	487	235	527	510	320	203	108	563	500	646	103
endereço	12	10	23	50	33	2	44	2	33	23	10	50

Note que no cálculo dos endereços houve repetições, tais como: 2, 33, 23, 10 e 50, por causa disto, é necessário verificar se o endereço está ocupado ou não. Finalmente os endereços calculados são:

Entrada	383	487	235	527	510	320	203	108	563	500	646	103
endereço	12	10	23	50	33	2	44	3	34	24	11	51

Problema: Escreva um programa em Java que faz um **Cálculo de Endereço (*Hashing*)** em uma tabela de números inteiros.

Programa exemplo (30): O programa demonstra operações em uma Tabela ***Hashing***.

```
// ------------------------------------------------------------- Fonte: Dados30.java

package dados30;

import java.util.Scanner;

public class Dados30 {

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        String s;
        final int max = 5;
        boolean [] situacao = new boolean[max];
        int [] valor = new int[max];
        int n = 0, entrada, endereco;
        char tecla;

        inicializaTabela(situacao, valor);
        do {
            exibeTabela(situacao, valor);
            n++;
            System.out.print("Número [0 para Sair]: ");
            s = input.nextLine();
            entrada = Integer.parseInt(s);
            if (entrada != 0) {
                insereTabela(situacao, valor, entrada);
                }
       } while (entrada != 0 && n <= max);
       exibeTabela(situacao, valor);
       do {
           System.out.print("Valor a ser CONSULTADO: ");
           s = input.nextLine();
           entrada = Integer.parseInt(s);
           if (entrada != 0) {
               endereco = hashing(situacao, valor, entrada);
```

```
                if (endereco == -1) {
                    System.out.println("Status: Valor Inválido");
                }
                else {
                    System.out.println("Endereco: %" + endereco);
                }
            }
        } while (entrada != 0);
        System.exit(0);
    }

    // ---------------------------------------------- inicializaTabela

    static void inicializaTabela(boolean [] situacao, int [] valor) {
        for (int i = 0;i < situacao.length;i++) {
            situacao[i] = false;
            valor[i] = 0;
        }
    }

    // ---------------------------------------------- testaTabela

    static boolean testaTabela(boolean [] situacao) {
           for (int i = 0;i < situacao.length; i++) {
               if (situacao[i] == false) {
                   return(false);
               }
           }
           return(true);
        }

    // ---------------------------------------------- insereTabela

    static void insereTabela(boolean [] situacao, int [] valor, int entrada) {
        final int max = 5;
        int endereco, n = -1;
        boolean lotado = testaTabela(situacao);

        if (!lotado) {
            endereco = entrada % max;
            while (situacao[endereco] == true) {
                endereco++;
                n++;
            }
            if (n < max) {
                valor[endereco] = entrada;
                situacao[endereco] = true;
            }
        }
        else {
            System.out.println("Status: TABELA CHEIA ...");
        }
    }

    // ---------------------------------------------- hashing

    static int hashing(boolean [] situacao, int [] valor, int entrada) {
        final int max = 5;
        int endereco;

        endereco = entrada % max;
        while (valor[endereco] != entrada && endereco != max) {
            endereco++;
            if (endereco == max) {
                return(-1);
            }
        }
        if (endereco != max) {
```

```
            return(endereco);
        }
        else {
            return(-1);
        }
    }

    // ---------------------------------------------- exibeTabela

    static void exibeTabela(boolean [] situacao, int [] valor) {
        final int max = 5;

        for (int i = 0;i < max;i++) {
            System.out.printf("%3d ",i);
        }
        System.out.println();
        for (int i = 0;i < max;i++) {
            System.out.printf("%03d ",valor[i]);
        }
        System.out.println();
    }
}
```

5. Classificação de Dados (Ordenação)

É o processo pelo qual é determinada a ordem em que devem ser apresentados os elementos de uma tabela de modo a obedecer à seqüência de um ou mais campos (chaves de classificação).

```
Classificação Interna..................... Memória Principal
Classificação Externa..................... Memória Secundária
```

5.1 Classificação por Força Bruta

0	Carla
1	Beatriz
2	Débora
3	Ana

Tabela Original

0	Ana
1	Beatriz
2	Carla
3	Débora

Tabela Ordenada

Logo, os elementos são fisicamente reorganizados.

Problema: Escreva um programa em Java que ordena (classifica em ordem crescente) um vetor de números inteiros.

Programa exemplo (31): O programa demonstra um programa de classificação por **força bruta.**

```
// ---------------------------------------------------------- Fonte: Dados31.java

package dados31;

import java.util.Scanner;

public class Dados31 {

    public static void main(String[] args) {
        final int QUANT = 10;
        int [] v = new int[QUANT];
        Scanner entrada = new Scanner(System.in);
        String s;
        int n = -1;
        char tecla;

        do {
            n++;
            System.out.print("Valor: ");
```

```
            s = entrada.nextLine();
            v[n] = Integer.parseInt(s);
        } while (v[n] != 0 && n < QUANT);
        if (v[n] == 0) {
            n--;
        }
        sortForcaBruta(v, n);
        exibeLista(v, n);
        System.exit(0);
    }

    // ------------------------------------------------------ sortForcaBruta

    static void sortForcaBruta(int [] v, int n) {
        for (int i = 0;i < n;i++) {
            for (int j = i+1;j <= n;j++) {
                if (v[i] > v[j]) {
                    int temp = v[i];
                   v[i] = v[j];
                   v[j] = temp;
                }
            }
        }
    }

// ------------------------------------------------------ exibe

    static void exibeLista(int [] v, int n) {
        System.out.print("Lista: ");
        if (n == -1) {
           System.out.println("Vazia");
        }
        else {
              for (int i = 0;i <= n;i++) {
                 System.out.printf("%2d ", v[i]);
              }
        }
        System.out.println();
    }
}
```

5.2 Vetor Indireto de Ordenação (Tabela de Índices)

Os elementos **não** são reorganizados fisicamente, apenas é criado um outro vetor (**tabela de índices**) que controla a ordem do primeiro.

Exemplo:

n

0	Carla
1	Beatriz
2	Débora
3	Ana

Tabela Original

vio

0	3
1	1
2	0
3	2

Tabela de Índices

Problema: Escreva um programa em Java que ordena (classifica em ordem crescente) um vetor de números inteiros utilizando um **Vetor Indireto de Ordenação** (VIO).

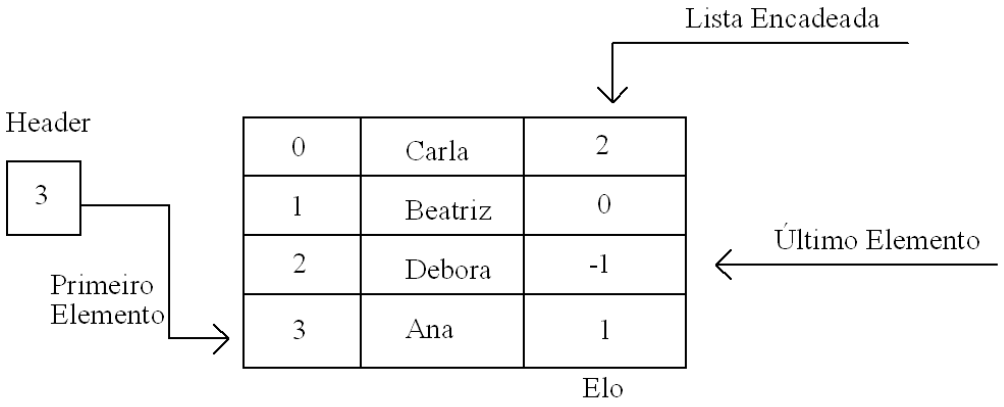
Programa exemplo (32): O programa demonstra a utilização de um vetor indireto de ordenação.

```
// ------------------------------------------------------------ Fonte: Dados32.java

package dados32;

import java.util.Scanner;

public class Dados32 {

    public static void main(String[] args) {
        final int QUANT = 10;
        String [] v = new String[QUANT];
        int [] vio = new int[QUANT];
        Scanner entrada = new Scanner(System.in);
        String s;
        int u = -1;
        char tecla;
        boolean troca, ok = true;

        do {
            u++;
            System.out.print("Nome: ");
            v[u] = entrada.nextLine();
            System.out.print("Continua [S/N] ?");
            vio[u] = 0;
            do {
                s = entrada.nextLine();
               tecla = s.charAt(0);
                tecla = Character.toLowerCase(tecla);
           } while (tecla != 's' && tecla != 'n');
        } while (tecla != 'n' && u <= QUANT);
        for (int i = 0;i <= u;i++) {
           vio[i] = i;
        }
        for (int i = 0;i < u;i++) {
            for (int j = i+1;j <= u;j++) {
                if (v[vio[i]].compareTo(v[vio[j]]) > 0) {
                    int temp = vio[i];
                   vio[i] = vio[j];
                   vio[j] = temp;
               }
            }
        }

        System.out.println("Lista de Nomes Ordenados");
        for (int i = 0;i <= u;i++) {
            System.out.printf("Nome: %s\n",v[vio[i]]);
        }
    }
}
```

5.3 Classificação por Encadeamento

Os elementos permanecem em seus lugares. É criado então uma lista encadeada ordenada. Esta lista possui um ***Header*** (Cabeça) o qual indica o primeiro elemento da lista.

Exemplo:



Problema: Escreva um programa em Java que ordena (classifica em ordem crescente) um vetor de números inteiros utilizando **encadeamento**.

Programa exemplo (33): O programa demonstra a ordenação utilizando encadeamento.

```
// ------------------------------------------------------------ Fonte: Tabela.java

package dados33;

public class Tabela {
    public String nome;
    public int prox = -1;
}

// ---------------------------------------------------------------- Fonte: Dados33.java

package dados33;

import java.util.Scanner;

public class Dados33 {

    public static void main(String[] args) {
        final int QUANT = 10;
        Tabela [] t = new Tabela[QUANT];
        Tabela [] ta = new Tabela[QUANT];
        Scanner entrada = new Scanner(System.in);
        String s;
        int j = 0, k, m, u = -1;
        int anterior, primeiro, sai;
        char tecla;

        do {
            u++;
            System.out.print("Nome: ");
            s = entrada.nextLine();
            t[u] = new Tabela();
            t[u].nome = s;
            t[u].prox = -1;
            System.out.print("Continua [S/N]? ");
            do {
```

```
            s = entrada.nextLine();
            tecla = s.charAt(0);
            tecla = Character.toLowerCase(tecla);
        } while (tecla != 's' && tecla != 'n');
    } while (tecla != 'n' && u < QUANT);
    primeiro = 0;
    for (int i = 1;i <= u;i++) {
        if (t[i].nome.compareTo(t[primeiro].nome) < 0) {
            primeiro = i;
        }
    }
    t[primeiro].prox = 0;
    anterior = primeiro;
    do {
        m = copia(t, ta, u);
        if (m != 0) {
            if (m >= 1) {
                int i = 1;
                j = 0;
                do {
                    if (ta[i].nome.compareTo(ta[j].nome) < 0) {
                        j = i;
                     }
                    i++;
                } while (i <= m);
            }
        }
        else {
          j = 0;
        }
  k = verifica(t, ta, j);
  t[anterior].prox = k;
  t[k].prox = 0;
  anterior = k;
} while (m != 0);

j = primeiro;
System.out.println("Lista de Nomes Ordenados por Encadeamento");
for (int i = 0;i <= u;i++) {
  System.out.printf("%s\n",t[j].nome);
  j = t[j].prox;
}
}

// ---------------------------------------------------- verifica

static int verifica(Tabela [] t, Tabela [] ta, int j) {
    boolean sai;
    int i = 0;

    do {
        sai = false;
        if (t[i].nome.compareTo(ta[j].nome) == 0) {
            j = i;
           sai = true;
        }
        i++;
    } while (!sai);
    return(j);
}

// ------------------------------------------------ Copia

static int copia(Tabela [] t, Tabela [] ta, int n) {
    int m = -1;

    for (int i = 0;i <= n;i++) {
        ta[i] = new Tabela();
```

```
            if (t[i].prox == -1) {
                ta[m].nome = t[i].nome;
                ta[m].prox = -1;
            }
        }
        return(m);
    }
}
```

5.4 Métodos de Classificação Interna

Os métodos de Classificação Interna podem ser:

- Por **Inserção**
- Por **Troca**
- Por **Seleção**

Observação: Para os seguintes métodos considere que as entradas são feitas no vetor ***v***, logo após é criado o vetor ***c*** (chaves) e o vetor ***e*** (endereços). A ordenação é feita no vetor ***c*** e o vetor ***e*** é o vetor indireto de ordenação, ou seja, será mantido o vetor de entrada intacto.

v

0	50
1	30
2	20
3	10
4	40

Vetor Original

c

0	50
1	30
2	20
3	10
4	40

Chaves

e

0	0
1	1
2	2
3	3
4	4

Vetor Indireto de Ordenação

5.4.1 Método por Inserção Direta

Neste método ocorre a inserção de cada elemento em outro vetor ordenado.

v

0	50
1	30
2	20
3	10
4	40

Vetor Original

c

0	10
1	20
2	30
3	40
4	50

Chaves

e

0	3
1	2
2	1
3	4
4	0

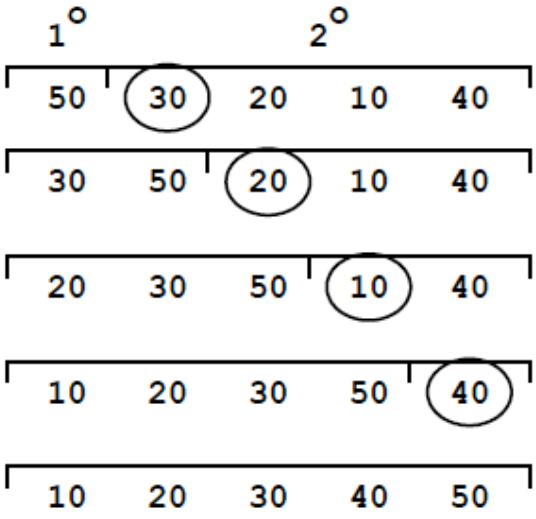
Vetor Indireto de Ordenação

Utilização: Pequena quantidade de dados, pois é pouco eficiente.

O vetor é dividido em dois segmentos. Inicialmente:

c[0] e c[1], c[2], ... c[n]

A classificação acontece por meio de interações, cada elemento do segundo segmento é inserido ordenadamente no primeiro até que o segundo segmento acabe. Por exemplo:



Problema: Escreva um programa em Java que ordena (classifica em ordem crescente) um vetor de números inteiros utilizando **Método por Inserção Direta**.

Programa exemplo (34): O programa demonstra a ordenação utilizando o método de inserção direta.

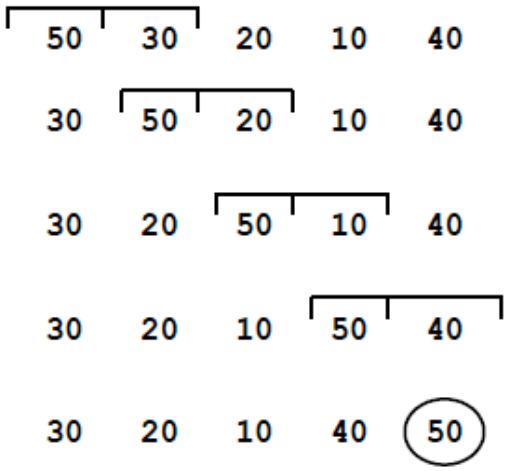
```
// ------------------------------------------------------------------ Fonte: Dados34.java

package dados34;

import java.util.Scanner;

public class Dados34 {

    public static void main(String[] args) {
        final int QUANT = 10;
        int [] v = new int[QUANT];
        int [] c = new int[QUANT];
        int [] e = new int[QUANT];
        int j, k, u = -1, temp;
        int chave, endereco;
        Scanner entrada = new Scanner(System.in);
        String s;
        char tecla;

        do {
            u++;
            System.out.print("Número: ");
            s = entrada.nextLine();
            temp = Integer.parseInt(s);
            if (temp != 0) {
                v[u] = temp;
            }
            else {
                u--;
            }
        } while (temp != 0 && u < QUANT);
        for (int i = 0;i <= u;i++) {
            c[i] = v[i];
            e[i] = i;
        }
        for (int i = 1;i <= u;i++) {
            k = 0;
            j = i - 1;
            chave = c[i];
            endereco = e[i];
            while (j >= 0 && k == 0) {
                if (chave < c[j]) {
                 c[j+1] = c[j];
                 e[j+1] = e[j];
                 j--;
               }
               else {
                 k = j + 1;
                }
            }
            c[k] = chave;
            e[k] = endereco;
        }
        System.out.printf("  Valores: ");
        for (int i = 0;i <= u;i++) {
            System.out.printf("%2d ", c[i]);
        }
        System.out.println();

        System.out.print("Endereços: ");
        for (int i = 0;i <= u;i++) {
            System.out.printf("%2d ", e[i]);
        }
        System.out.println();
        System.exit(0);
    }
}
```

5.4.2 Método por Troca

Neste método, compara-se pares de elementos, trocando-os de posição caso estejam desordenados.

5.4.2.1 Método da Bolha (*Bubble Sort*)

Cada elemento do vetor é testado com o seguinte, se estiverem fora de ordem ocorre a troca, isto é repetido até não ocorrer mais trocas. Por exemplo:



Observação: Na primeira passagem completa o último elemento esta ordenado, logo na segunda passagem não é necessário ir até o fim.

Problema: Escreva um programa em Java que ordena (classifica em ordem crescente) um vetor de números inteiros utilizando o **Método da Bolha**.

Programa exemplo (35): Programa abaixo demonstra a ordenação utilizando o ***Bubble Sort***.

```
// ---------------------------------------------------------------- Fonte: Dados35.java

package dados35;

import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        final int QUANT = 10;
        int [] v = new int[QUANT];
```

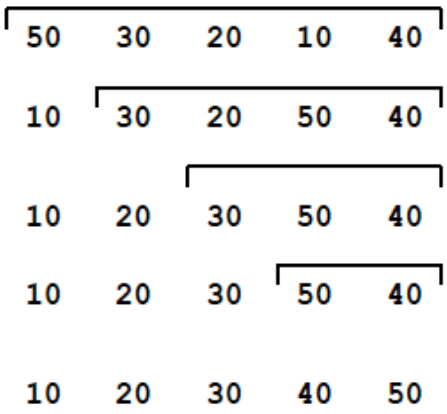
```
        int [] c = new int[QUANT];
        int [] e = new int[QUANT];
        int j = 0, n = -1, m, temp;
        int chave, endereco;
        boolean troca;
        Scanner entrada = new Scanner(System.in);
        String s;
        char tecla;

        do {
            n++;
            System.out.print("Número: ");
            temp = entrada.nextInt();
            if (temp != 0) {
               v[n] = temp;
            }
            else {
                n--;
            }
        } while (temp != 0 && n < QUANT);
        for (int i = 0;i <= n;i++) {
            c[i] = v[i];
            e[i] = i;
        }
        m = n - 1;
        do {
            troca = true;
            for (int i = 0;i <= m;i++) {
                if (c[i] > c[i+1]) {
                    chave = c[i];
                   c[i] = c[i+1];
                   c[i+1] = chave;
                   endereco = e[i];
                   e[i] = e[i+1];
                   e[i+1] = endereco;
                   j = i;
                   troca = false;
                }
            }
           m = j;
        } while (!troca);
        System.out.print("Lista Ordenada: ");
        for (int i = 0;i <= n;i++) {
            System.out.print(c[i] + " ");
        }
        System.out.println();
        System.exit(0);
    }
}
```

5.4.3 Método por Seleção

Seleção sucessiva do menor valor da tabela. A cada passo o menor elemento é colocado em sua posição definitiva.

5.4.3.1 Método por Seleção Direta

A cada passo do método é feita uma varredura do segmento que corresponde os elementos, ainda não selecionados, e determinado o menor elemento o qual é colocado na primeira posição do elemento por troca. Por exemplo:



Problema: Escreva um programa em Java que ordena (classifica em ordem crescente) um vetor de números inteiros utilizando o **Método por Seleção Direta**.

Programa exemplo (36): Programa abaixo demonstra a ordenação utilizando o **método da seleção direta.**

```
// ------------------------------------------------------------ Fonte: Dados36.java

package dados36;

import java.util.Scanner;

public class Dados36 {

    public static void main(String[] args) {
        final int QUANT = 10;
        int [] v = new int[QUANT];
        int [] c = new int [QUANT];
        int [] e = new int [QUANT];
        Scanner entrada = new Scanner(System.in);
        int j, n = -1, min, temp;
        int chave, endereco, troca;
        char tecla;

        do {
            n++;
            System.out.print("Número: ");
            temp = entrada.nextInt();
            if (temp != 0) {
                v[n] = temp;
            }
            else {
                n--;
            }
        } while (temp != 0 && n < QUANT);
        for (int i = 0;i <= n;i++) {
            c[i] = v[i];
            e[i] = i;
        }
        for (int i = 0;i <= n-1;i++) {
            min = i;
            for (j = i+1;j <= n;j++) {
                if (c[j] < c[min]) {
```

```
                    min = j;
                }
            }
            chave = c[i];
            c[i] = c[min];
            c[min] = chave;
            endereco = e[i];
            e[i] = e[min];
            e[min] = endereco;
        }
        System.out.print("Lista Ordenada: ");
        for (int i = 0;i <= n;i++) {
            System.out.printf("%d ", c[i]);
        }
        System.out.println();
        System.exit(0);
    }
}
```

6. Árvores

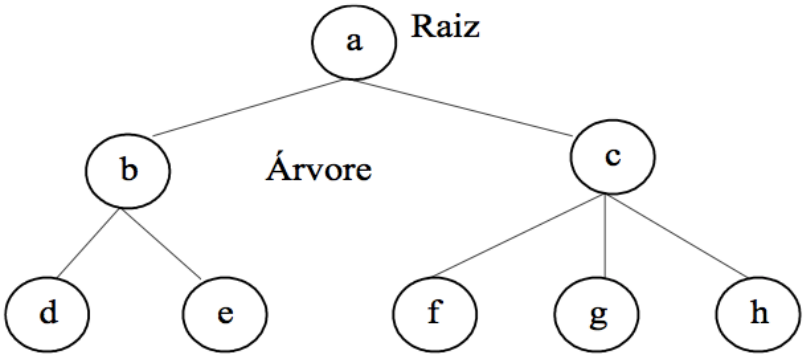
São estruturas de dados (não-lineares) que caracterizam uma relação entre os dados, à relação existente entre os dados é uma relação de **hierarquia** ou de **composição** (um conjunto é subordinado a outro).

6.1 Conceitos Básicos

Definição

É um conjunto finito T de um ou mais nós, tais que:

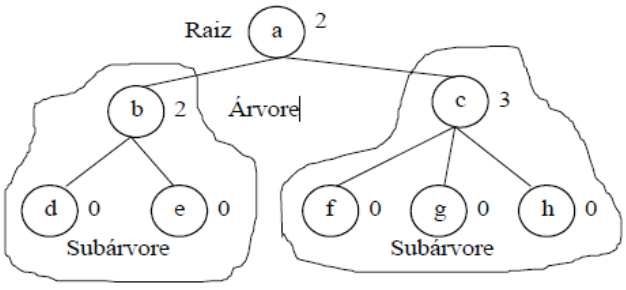
a) Existe um nó principal chamado **raiz** (*root*);
b) Os demais nós formam n >= 0 conjuntos disjuntos T1, T2, ... Tn, onde cada um destes subconjuntos é uma árvore. As árvores Ti (i >= 1 e i <= n) recebem a denominação de sub-árvores.



Terminologia

Grau

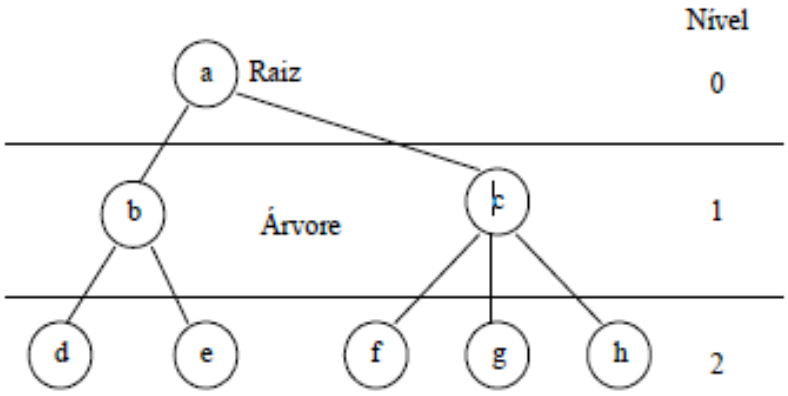
Indica o número de sub-árvores de um nó.



Observação: Se um nodo não possuir nenhuma sub-árvore é chamado de **nó terminal** ou **folha.**

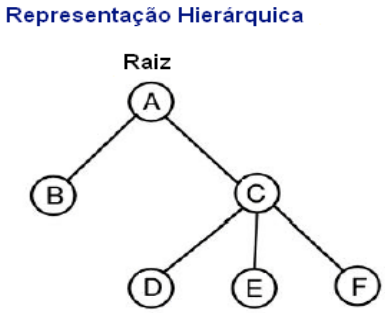
Nível

É o comprimento, ou seja, o número de linhas do caminho da raiz até o nó.



Observação: Raiz é nível zero (0)

Exemplo:



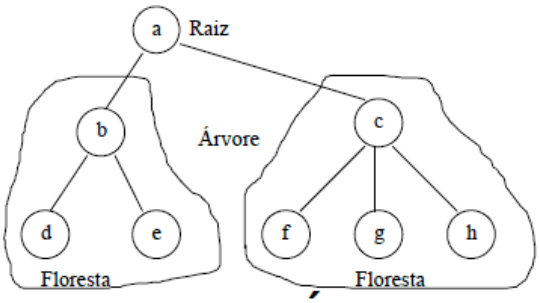
Nodo	Grau	Nível
A	2	0
B	0	1
C	3	1
D	0	2
E	0	2
F	0	2

Altura

É o nível mais alto da árvore. Na árvore acima, a altura é igual a 2.

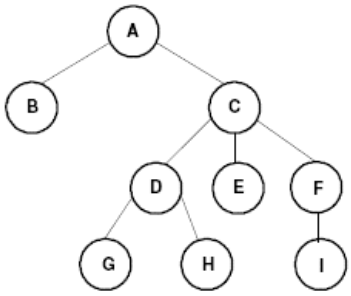
Floresta

É um conjunto de zero ou mais árvores disjuntas, ou seja, se for eliminado o nó raiz da árvore, as sub-árvores que restarem chamam-se de florestas.

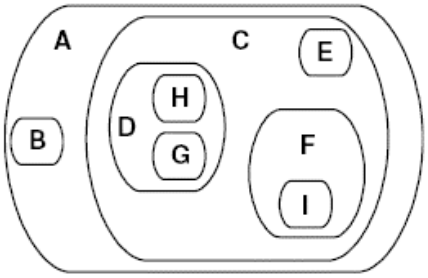


Formas de Representação de Árvores

α) **Representação Hierárquica**



β) **Representação por Conjunto** (Diagrama de Inclusão ou Composição)



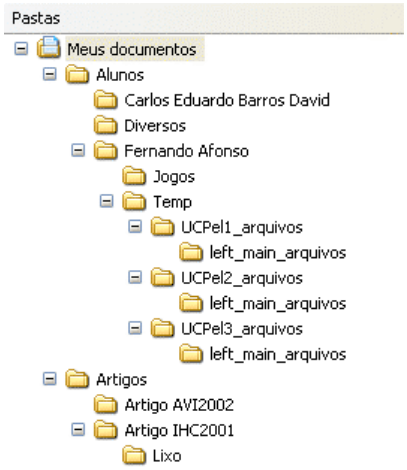
χ) **Representação por Expressão Parentetizada** (Parênteses Aninhados)

(**A** (**B** ( ) **C** (**D** (**G** ( ) **H** ( )) **E** ( ) **F** (**I** ( )))))

δ) **Representação por Expressão não Parentetizada**

A**A** 2 **B** 0 **C** 3 **D** 2 **G** 0 **H** 0 **E** 0 **F** 1 **I** 0

ε) **Representação por Endentação** (Digrama de Barras)

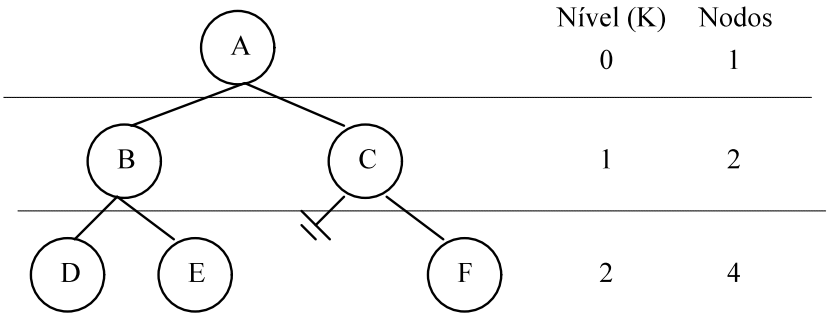


6.2 Árvores Binárias

Uma árvore binária (**T**) é um conjunto finito de nós que pode ser vazio ou pode ser dividida em três sub-conjuntos disjuntos: *raiz*, *sub-árvore esquerda* (**Te**) e *sub-árvore direita* (**Td**). São estruturas onde o grau de cada nó é menor ou igual a dois, ou seja, no máximo grau 2. O número máximo de nós no nível ***i*** é 2^i.

São árvores onde cada nó tem no máximo dois filhos (grau máximo 2), desta forma, obtém-se uma estrutura apropriada para busca binária, pois sabe-se que existe, para cada nodo, duas sub-árvores (**Te** e **Td**).

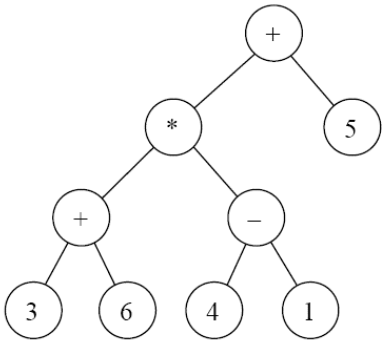
Para cada nó da árvore associa-se uma chave (dado, valor ou informação) que permite a realização de uma classificação. A Construção da árvore deve ser de forma que na sub-árvore à esquerda (**Te**) da raiz só existam nós com chaves menores que a chave da raiz. E a sub-árvore à direita (**Td**) só pode conter nós com valores maiores que a raiz. Com esta reestruturação da árvore, a busca de um determinado nó torna-se trivial. O acesso aos dados pode ser feito então através de funções recursivas.



Propriedades

1) O número máximo de nodos no k-ésimo nível de uma árvore binária é 2^k;
2) O número máximo de nodos em uma árvore binária com altura k é $2^{k+1}-1$, para k >= 0;
3) Árvore completa: árvore de altura k com $2^{k+1}-1$ nodos.

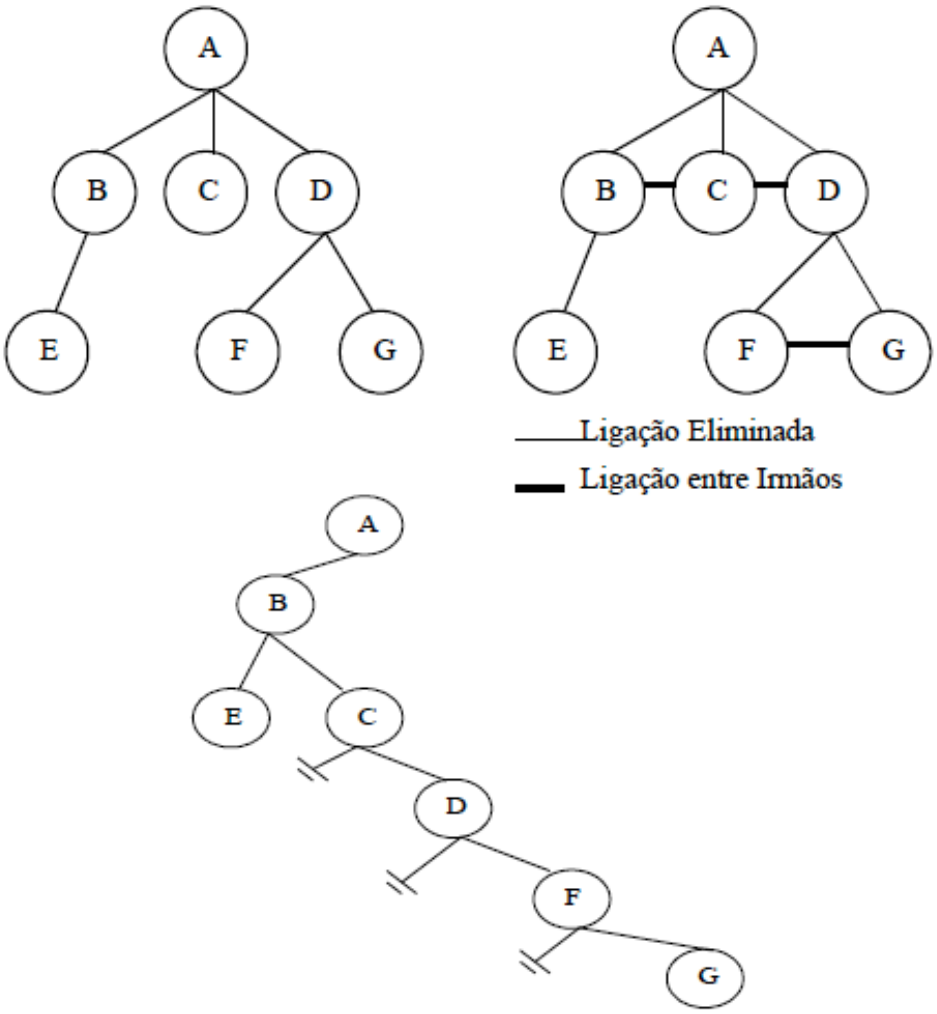
Exemplo de uma aplicação de Árvore Binária
(Analisador de Expressão):



Expressão: **(3 + 6) * (4 – 1) + 5**

Conversão de Árvore Genérica em Árvore Binária

- Ligar os nós irmãos;
- Remover a ligação entre o nó pai e seus filhos, exceto as do primeiro filho.



Nota: O nó da sub-árvore à esquerda é ***filho*** e o nó da sub-árvore à direita é ***irmão***

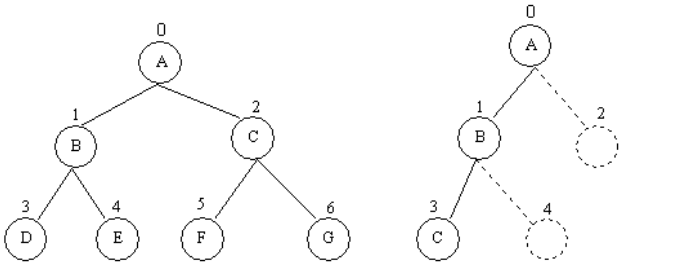
6.3 Representações

6.3.1 Representação por Contigüidade Física (Adjacência)

Os nodos são representados seqüencialmente na memória. Devem ser tomados os seguintes cuidados:

- Ser alocado espaço suficiente para armazenar a estrutura completa;
- Os nodos devem ser armazenados em uma lista, onde cada nodo ***i*** da árvore ocupa o i-ésimo nodo da lista.

Exemplo:



0	1	2	3	4	5	6
A	B	C	D	E	F	G

0	1	2	3	4	5	6
A	B	-	C	-	-	-

Sendo ***i*** a posição de um nodo e ***n*** o número máximo de nodos da árvore.

Observações:

1) O pai de i está em i / 2 sendo i <= n. Se i = 1, i é a raiz e não possui pai.
2) O filho à esquerda de i está em 2i se 2i <= n. Se 2i > n, então tem filho à esquerda.
3) O filho à direita de i está em 2i+1. Se 2i+1 <= n. Se 2i+1 > n, então tem filho à direita.

Observação: Representação por **Contigüidade Física** não é um modo conveniente para representar árvores, na maioria dos casos.

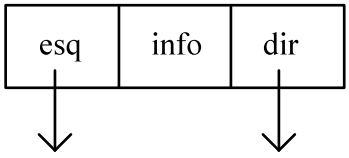
Vantagens

- Adequado para árvores binárias completas.
- Útil para o armazenamento em disco ou fita (seqüencial).

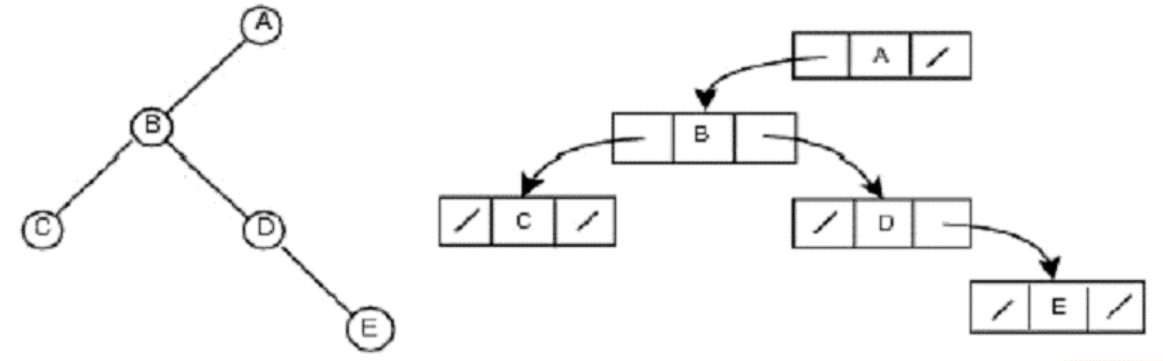
Desvantagem

- A estrutura pode ter muitos espaços sem uso.
- Estrutura possui limite finito no número de nodos.

6.3.2 Representação por Encadeamento



esq: endereço do nodo filho à esquerda
info: contém a informação do nodo
dir: endereço do nodo filho à direita




```
public class Tree {
    public Tree esq;
    public char info;
    public Tree dir;
}
```

6.4 Caminhamento em Árvores

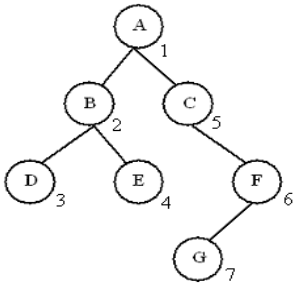
Consiste em processar de forma sistemática e ordenada cada nó da árvore apenas uma vez, obtendo assim uma seqüência linear de nós. Abaixo são mostrados os três tipos de caminhamentos:

6.4.1 Caminhamento Pré-Fixado (Pré-Ordem)

1) Visitar a raiz;
2) Caminhar na sub-árvore da esquerda;
3) Caminhar na sub-árvore a direita.

Observação: "Visitar" significa qualquer operação em relação à informação (***info***) do nodo.

Exemplo:

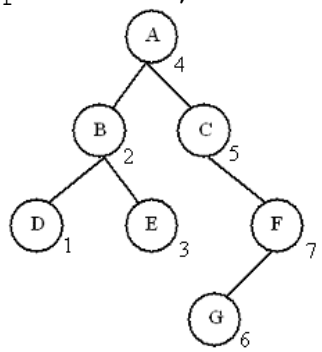


Caminhamento: ABDECFG

6.4.2 Caminhamento In-Fixado (Central)

1) Caminhar na sub-árvore da esquerda;
2) Visitar a raiz;
3) Caminhar na sub-árvore da direita.

Exemplo: Conforme exemplo acima, o caminhamento ***In-Fixado*** é:

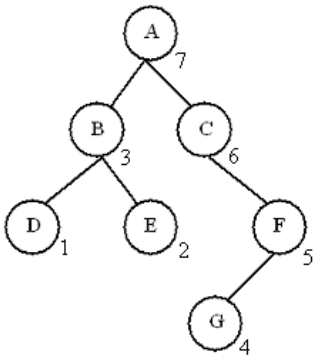


Caminhamento: DBEACGF

6.4.3 Caminhamento Pós-Fixado

1) Caminhar na subárvore da esquerda;
2) Caminhar na subárvore da direita;
3) Visitar a raiz.

Exemplo: Conforme exemplo acima, o caminhamento Pós-Fixado é:



Caminhamento: DEBGFCA

6.4.4 Algoritmos recursivos para percorrer Árvores Binárias

Algoritmos de busca de dados em estruturas hierárquicas, caminhamentos em árvores, por exemplo, utilizam, normalmente, recursividade.

Recursividade é uma técnica utilizada em programação quando se deseja que uma função faça uma chamada a si própria. Este mecanismo utiliza uma estrutura de pilha para fazer o controle do retorno de todas as chamadas realizadas.

Como **vantagens** das funções recursivas tem-se:

a) clareza na interpretação do código (funções pequenas);
b) "simplicidade" e elegância na implementação.

Como **desvantagens** tem-se:

a) dificuldade para encontrar erros (*debug*);
b) dificuldade de encontrar o critério de parada da função;
c) em alguns casos podem ser ineficientes devido a quantidade de chamadas recursivas.

A chamada de uma função recursiva requer espaço para os parâmetros, variáveis locais e endereço de retorno. Todas estas informações são armazenadas em uma pilha e depois desalocadas, ou seja, a quantidade de informações é proporcional ao número de chamadas. Todas as operações envolvidas na recursividade contribuem para um gasto maior de tempo, pois a alocação e liberação de memória consomem tempo.

6.4.4.1 Caminhamento Pré-Fixado (Pré-Ordem)

```
static void caminhamentoPreOrdem(Tree a) {
        if (!treeVazia(a)) {
            System.out.printf("%c ", a.info);   // mostra raiz
            caminhamentoPreOrdem(a.esq);        // mostra sub_esq
            caminhamentoPreOrdem(a.dir);        // mostra sub_dir
        }
}
```

6.4.4.2 Caminhamento In-Fixado (Central)

```
static void caminhamentoInFixado(Tree a) {
        if (!treeVazia(a)) {
            caminhamentoInFixado(a.esq);       // mostra sub_esq
            System.out.printf("%c ", a.info);  // mostra raiz
            caminhamentoInFixado(a.dir);       // mostra sub_dir
        }
}
```

6.4.4.3 Caminhamento Pós-Fixado

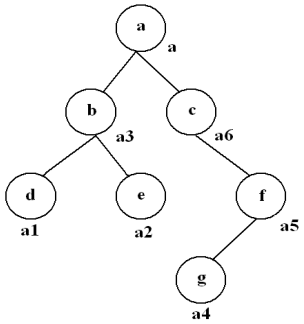
```
static void caminhamentoPosFixado(Tree a) {
        if (!treeVazia(a)) {
            caminhamentoPosFixado(a.esq);       // mostra sub_esq
            caminhamentoPosFixado(a.dir);       // mostra sub_dir
            System.out.printf("%c ", a.info);   // mostra raiz
        }
}
```

Problema: Escreva um programa em Java que cria a árvore da página 137. O programa deve exibir na tela os três tipos de caminhamentos.

Programa exemplo (37): O programa a seguir demonstra os detalhes da implementação de uma árvore.

```
// ------------------------------------------------------------ Fonte: Tree.java

package dados37;

public class Tree {
    public Tree esq;
    public char info;
    public Tree dir;

    // ------------------------------------------- criaTree

    public Tree(Tree esquerda, char informacao, Tree direita) {
        esq = esquerda;
        info = informacao;
        dir = direita;
    }
}
```

```
// ---------------------------------------------------- Fonte: Dados37.java

package dados37;

public class Dados37 {

    public static void main(String[] args) {
        Tree a1 = new Tree(null,'d',null);
        Tree a2 = new Tree(null,'e',null);
        Tree a3 = new Tree(a1,'b',a2);
        Tree a4 = new Tree(null,'g',null);
        Tree a5 = new Tree(a4,'f',null);
        Tree a6 = new Tree(null,'c',a5);
        Tree a = new Tree(a3,'a',a6);

        System.out.printf("Caminhamentos em Árvore\n Pré-Ordem: ");
        caminhamentoPreOrdem(a);
        System.out.printf("\n In-Fixado: ");
        caminhamentoInFixado(a);
        System.out.printf("\nPós-Fixado: ");
        caminhamentoPosFixado(a);
        System.out.println();
        System.exit(0);
    }

    // ---------------------------------------------- treeVazia

    static boolean treeVazia(Tree a) {
        if (a == null) {
            return(true);
        }
        else {
          return(false);
        }
    }

    // ---------------------------------------------- caminhamentoPreOrdem

    static void caminhamentoPreOrdem(Tree a) {
        if (!treeVazia(a)) {
            System.out.printf("%c ", a.info);   // mostra raiz
            caminhamentoPreOrdem(a.esq);        // mostra sub_esq
            caminhamentoPreOrdem(a.dir);        // mostra sub_dir
        }
    }

    // ---------------------------------------------- caminhamentoInFixado

    static void caminhamentoInFixado(Tree a) {
        if (!treeVazia(a)) {
            caminhamentoInFixado(a.esq);       // mostra sub_esq
            System.out.printf("%c ", a.info);  // mostra raiz
            caminhamentoInFixado(a.dir);       // mostra sub_dir
        }
    }

    // ---------------------------------------------- caminhamentoPosFixado

    static void caminhamentoPosFixado(Tree a) {
        if (!treeVazia(a)) {
            caminhamentoPosFixado(a.esq);       // mostra sub_esq
            caminhamentoPosFixado(a.dir);       // mostra sub_dir
            System.out.printf("%c ", a.info);   // mostra raiz
        }
    }
}
```



Resultado do Programa:

Caminhamentos na Árvore

Pré-Ordem: ***a b d e c f g***
In-Fixado: ***d b e a c g f***
Pós-Fixado: ***d e b g f c a***

6.5 Árvore de Busca Binária

Uma **árvore de busca binária** (BST - *Binary Search Tree*) é uma árvore binária cujas chaves (informações ou dados) aparecem em ordem crescente quando a árvore é percorrida em ordem in-Fixado (**esquerda -> raiz -> direita**, *ou seja*, **caminhamento ERD**). Onde a chave de cada nó da árvore deve ser:

- *maior ou igual* que qualquer chave na sua sub-árvore esquerda;
- *menor ou igual* que qualquer chave na sua sub-árvore direita.

Em outras palavras, a ordem **esquerda-raiz-direita** das chaves deve ser crescente.

Problema: Escreva um programa em Java que cria a **árvore binária ordenada** (a, b, c, d, e, f, g). O programa deve permitir ao usuário buscar o endereço de uma determinada informação, ou seja, uma letra de 'a' até 'z'.

Programa exemplo (38): O programa a seguir demonstra os detalhes da implementação de uma árvore.

```
// ---------------------------------------------------------- Fonte: Tree.java

package dados38;
```

```
public class Tree {
    public Tree esq;
    public char info;
    public Tree dir;

    // ------------------------------------------- criaTree (construtor)

    public Tree(Tree esquerda, char informacao, Tree direita) {
        esq = esquerda;
        info = informacao;
        dir = direita;
    }
}

// ------------------------------------------------------- Fonte: Dados38.java

package dados38;

import java.util.Scanner;

public class Dados38 {

    public static void main(String[] args) {
        Tree a1 = new Tree(null,'d',null);
        Tree a2 = new Tree(null,'e',null);
        Tree a3 = new Tree(a1,'b',a2);
        Tree a4 = new Tree(null,'g',null);
        Tree a5 = new Tree(a4,'f',null);
        Tree a6 = new Tree(null,'c',a5);
        Tree a = new Tree(a3,'a',a6);
        Scanner entrada = new Scanner(System.in);
        String s;
        Tree tree = new Tree(null, 'x', null);
        char info;

        System.out.println("In-Fixado: ");
        caminhamentoInFixado(a);
        System.out.println();
        do {
            System.out.print("Info [x - abandona]: ");
            do {
                s = entrada.nextLine();
                info = s.charAt(0);
            } while (!(info >= 'a' && info <= 'z') && info != 'x');
            if (info != 'x') {
                tree = buscaTree(a, info);
                if (tree == null) {
                    System.out.println("Status: Nodo Inválido");
                }
                else {
                    System.out.println("Status: Ok, Nodo Válido");
                }
            }
        } while (info != 'x');
        System.exit(0);
    }

    // ------------------------------------------- treeVazia

    static boolean treeVazia(Tree a) {
        if (a == null) {
            return(true);
        }
        else {
          return(false);
        }
    }
```

```
    // ------------------------------------------- caminhamentoPreOrdem

    static void caminhamentoPreOrdem(Tree a) {
        if (!treeVazia(a)) {
            System.out.printf("%c ", a.info);   // mostra raiz
            caminhamentoPreOrdem(a.esq);        // mostra sub_esq
            caminhamentoPreOrdem(a.dir);        // mostra sub_dir
        }
    }

    // ------------------------------------------- caminhamentoInFixado

    static void caminhamentoInFixado(Tree a) {
        if (!treeVazia(a)) {
            caminhamentoInFixado(a.esq);       // mostra sub_esq
            System.out.printf("%c ", a.info);  // mostra raiz
            caminhamentoInFixado(a.dir);       // mostra sub_dir
        }
    }

    // ------------------------------------------- caminhamentoPosFixado

    static void caminhamentoPosFixado(Tree a) {
        if (!treeVazia(a)) {
            caminhamentoPosFixado(a.esq);       // mostra sub_esq
            caminhamentoPosFixado(a.dir);       // mostra sub_dir
            System.out.printf("%c ", a.info);   // mostra raiz
        }
    }

    // ------------------------------------------- BuscaTree

    static Tree buscaTree(Tree raiz, char chave) {
        Tree a1;

        if (raiz == null) {
            return(null);
        }
        else {
            if (raiz.info == chave) {       // busca na raiz
              return(raiz);
            }
            else {
                a1 = buscaTree(raiz.esq, chave);    // busca na sub-árvore esquerda
               if (a1 == null) {
                 a1 = buscaTree(raiz.dir, chave);  // busca na sub-árvore direita
                }
            }
           return(a1);
        }
   }
}
```

6.6 Árvore AVL

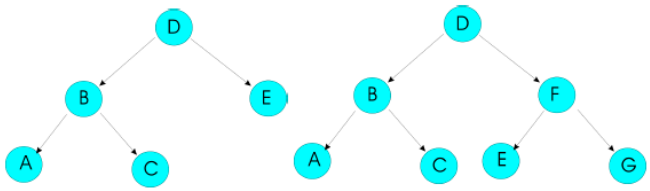
O objetivo principal na utilização de árvores AVL é diminuir o custo de acesso as informações desejadas, ou seja, organizar a árvore de forma a otimizar a busca em uma árvore binária.

Os algoritmos de árvore AVL são muito parecidos com os algoritmos de uma árvore binária, a diferença está no esforço necessário para se manter uma árvore AVL balanceada.

Para manter uma árvore balanceada, precisa-se constantemente refazer a estrutura da árvore nas operações de inserção ou exclusão de elementos. Árvores AVL, B e B++ são árvores balanceadas.

Balanceamento em Árvores

Diz-se que uma árvore está balanceada (equilibrada), se todos os nós folhas estão à mesma distância da raiz, ou seja, uma árvore é dita balanceada quando as suas sub-árvores à esquerda e à direita possuem a mesma altura. Quando uma árvore não está balanceada, chama-se degenerada.



O balanceamento de uma árvore binária pode ser: **estático** ou **dinânico** (AVL). O balanceamento **estático** de uma árvore binária consiste em construir uma nova versão da árvore, reorganizando-a, enquanto que no balanceamento **dinâmico** (AVL) a cada nova operação realizada na árvore binária, ela sobre rotações deixando-a balanceada.

O termo **AVL** foi colocado em homenagem aos matemáticos russos **A**delson-**V**elskii e **L**andis.

Adelson-Velskii e Landis em 1962 apresentaram uma árvore de busca binária que é balanceada levando-se em consideração a altura das suas sub-árvores, ou seja, uma árvore AVL é uma árvore binária de pesquisa onde a diferença em altura entre as sub-árvores esquerda e direita é no máximo 1 (positivo ou negativo).

Está diferença é chamado de **fator de balanceamento** (fb). Este fator deve ser calculado para todos os nós da árvore binária. O **fator de balanceamento** de um nó folha é sempre zero, ou seja, fb = 0.

O **fator de balanceamento** (fb) é um número inteiro igual a:

fb (nodo) = altura (subárvore direita) – altura (subárvore esquerda);

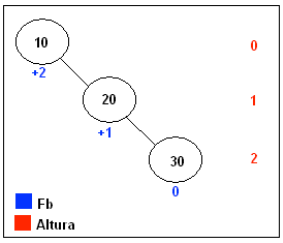
Definição: Uma árvore binária vazia é sempre balanceada por altura. Se **T** não é vazia e **Te** e **Td** são sub-árvores da esquerda e direita, respectivamente, então **T** é balanceada por altura se: a) **Te** e **Td** são balanceadas por altura; b) **altura Te – altura Td** for igual a 0, 1 ou -1.

6.6.1 Inserção em uma árvore AVL

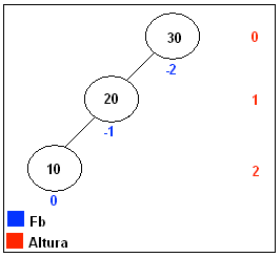
Quando um novo elemento é inserido em uma árvore binária, é necessário verificar se esta inserção quebrou a propriedade de balanceamento da árvore, ou seja, é necessário calcular o **fator de balanceamento** dos nodos da árvore. Se o **Fb** de algum nodo for diferente de 0, 1 ou -1, é necessário reestruturar a árvore para que volte a ser balanceada.

Na inserção de um nodo em uma árvore AVL, pode-se ter quatro situações distintas, cuja a forma de tratamento é diferente:

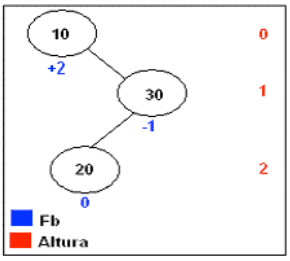
a) **Inserção dos nós:** 10, 20 e 30



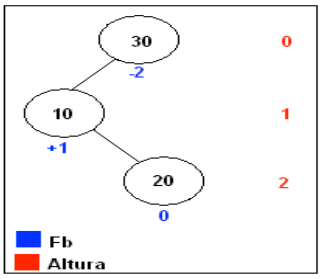
b) **Inserção dos nós:** 30, 20 e 10



c) **Inserção dos nós:** 10, 30 e 20



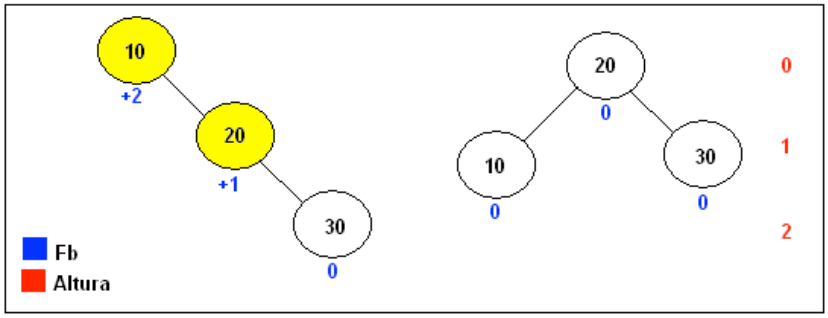
d) **Inserção dos nós:** 30, 10 e 20



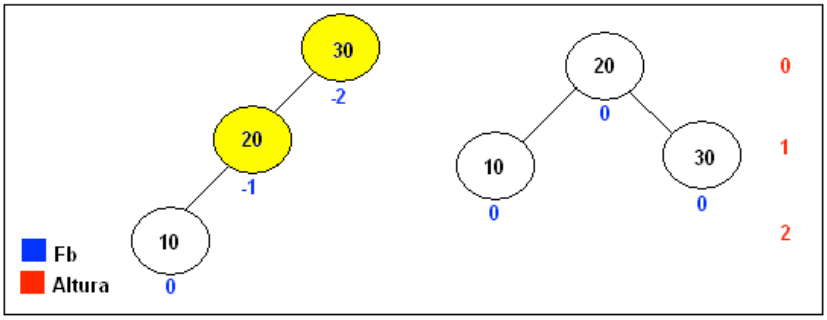
Note que nas quatro situações acima (**a**, **b**, **c** e **d**), o fator de balanceamento de algum nodo ficou fora da faixa [-1..1], ou seja, algum nodo teve o Fator de Balanceamento (**Fb**) 2 ou -2.

Para cada caso acima, aplica-se um tipo de rotação a árvore:

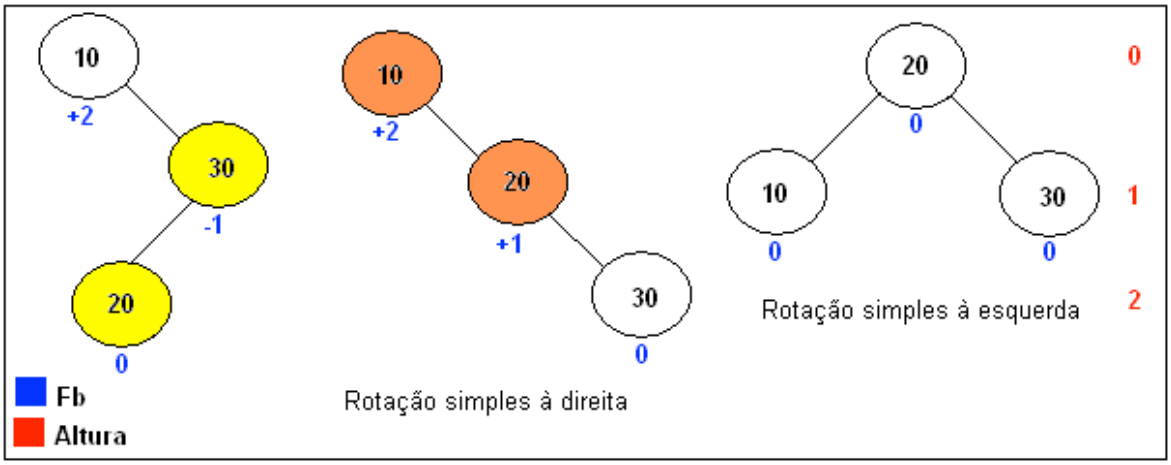
a) **Rotação simples à esquerda**



b) **Rotação simples à direita**

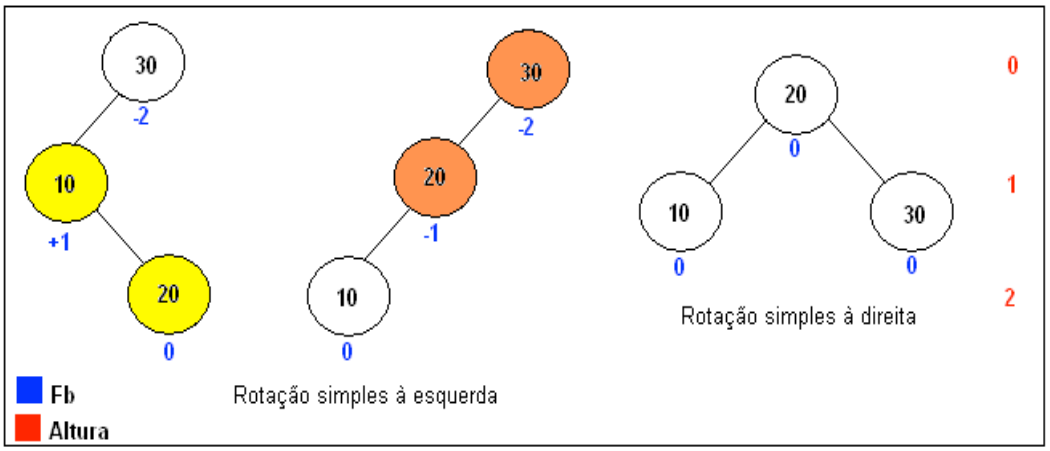


c) **Rotação dupla à esquerda**



(rotação simples à direita + rotação simples à esquerda)

d) **Rotação dupla à direita**



(rotação simples à esquerda + rotação simples à direita)

Dicas:

a) Para identificar quando uma rotação é **simples** ou **dupla** deve-se observar os sinais do Fb:
- Sinal for igual, a rotação é simples
- Sinal for diferente a rotação é dupla

b) Se **Fb** for positivo (+) a rotação para à esquerda
c) Se **Fb** for negativa (-) a rotação para à direita

Problema: Escreva um programa em Java que cria uma árvore binária balanceada utilizando as características de uma árvore AVL (10, 20, 30, 40, ...). O programa deve permitir a entrada de números inteiros até que o usuário digite zero (0). Ao final a árvore é exibida.

Programa exemplo (39): O programa a seguir demonstra os detalhes de uma árvore binária balanceada.

```
// ------------------------------------------------------------ Fonte: AvlNode.java

package Dados39;

class AvlNode {
    AvlNode esq;
    int info;
    AvlNode dir;
    int altura;

    // ---------------------------------------- Construtor

    AvlNode(int info) {
        this(null, info, null);
    }

    // ---------------------------------------- Construtor

    AvlNode(AvlNode esquerda, int info, AvlNode direita) {
        this.esq = esquerda;
        this.info = info;
        this.dir = direita;
        this.altura = 0;
    }
}
```

```
// ------------------------------------------------------------ Fonte: AvlTree.java

package Dados39;

public class AvlTree {

  // ---------------------------------------- maxAVL

  private static int maxAVL(int a,  int b) {
      if (a > b) {
          return(a);
      }
      else {
          return(b);
      }
```

```
}

// ------------------------------------------- alturaAVL

private static int alturaAVL(AvlNode t) {
    if (t == null) {
        return(-1);
    }
    else {
        return(t.altura);
    }
}

// ------------------------------------------- rotacaoEsquerdaAVL

private static AvlNode rotacaoEsquerdaAVL( AvlNode nodoOut) {
   AvlNode nodoIn = nodoOut.esq;
   int a, b;

   nodoOut.esq = nodoIn.dir;
   nodoIn.dir = nodoOut;
   a = AvlTree.alturaAVL(nodoOut.esq);
   b = AvlTree.alturaAVL(nodoOut.dir);
   nodoOut.altura = AvlTree.maxAVL(a, b) + 1;
   a = AvlTree.alturaAVL(nodoIn.esq);
   b = nodoOut.altura;
   nodoIn.altura = AvlTree.maxAVL(a, b) + 1;
   return(nodoIn);
}

// ------------------------------------------- rotacalDireitaAVL

private static AvlNode rotacaoDireitaAVL( AvlNode nodoOut) {
   AvlNode nodoIn = nodoOut.dir;
   int a, b;

   nodoOut.dir = nodoIn.esq;
   nodoIn.esq = nodoOut;
   a = AvlTree.alturaAVL(nodoOut.esq);
   b = AvlTree.alturaAVL(nodoOut.dir);
   nodoOut.altura = AvlTree.maxAVL(a, b) + 1;
   a = AvlTree.alturaAVL(nodoIn.dir);
   b = nodoOut.altura;
   nodoIn.altura = AvlTree.maxAVL(a, b) + 1;
   return(nodoIn);
}

// ------------------------------------------- rotacaoDuplaEsquerdaAVL

private static AvlNode rotacaoDuplaEsquerdaAVL(AvlNode nodo) {
  nodo.esq = AvlTree.rotacaoDireitaAVL(nodo.esq);
  return(AvlTree.rotacaoEsquerdaAVL(nodo));
}

// ------------------------------------------- rotacaoDuplaDireitaAVL

private static AvlNode rotacaoDuplaDireitaAVL(AvlNode nodo) {
  nodo.dir = AvlTree.rotacaoEsquerdaAVL(nodo.dir);
  return(AvlTree.rotacaoDireitaAVL(nodo));
}

private AvlNode raiz;

// ------------------------------------------- Constriutor

public AvlTree() {
  this.raiz = null;
}
```

```
// ------------------------------------------ inserteAVL

public void insertAVL(int info) {
  this.raiz = insertAVL(info, this.raiz);
}

// ------------------------------------------ inserteAVL (sobreCarga)

private AvlNode insertAVL( int x, AvlNode avl) {
    int a, b;

    if (avl == null) {
        avl = new AvlNode(null, x, null);
    }
    else {
        if (x > avl.info) {
            avl.dir = insertAVL(x, avl.dir);
            a = AvlTree.alturaAVL(avl.dir);
            b = AvlTree.alturaAVL(avl.esq);
            if (a - b == 2) {
                if (x > avl.dir.info) {
                    avl = AvlTree.rotacaoDireitaAVL(avl);
                    }
                else {
                    avl = AvlTree.rotacaoDuplaDireitaAVL(avl);
                    }
            }
        }
        else {
            if (x < avl.info) {
                avl.esq = insertAVL(x, avl.esq);
                a = AvlTree.alturaAVL(avl.esq);
                b = AvlTree.alturaAVL(avl.dir);
                if (a - b == 2) {
                    if (x < avl.esq.info) {
                        avl = AvlTree.rotacaoEsquerdaAVL(avl);
                    }
                    else {
                        avl = AvlTree.rotacaoDuplaEsquerdaAVL(avl);
                    }
                }
            }
        }
     }
     a = AvlTree.alturaAVL(avl.esq);
     b = AvlTree.alturaAVL(avl.dir);
     avl.altura = AvlTree.maxAVL(a, b) + 1;
     return(avl);
}

// ------------------------------------------ vaziaAVL

 public boolean vaziaAVL() {
     return this.raiz == null;
}

// ------------------------------------------ exibeAVL

 public void exibeAVL() {
  if (vaziaAVL()) {
    System.out.println("Status: AVL Vazia");
  } else {
    exibeAVL(this.raiz, 1);
  }
}

// ------------------------------------------ exibeAVL (SobreCarga)
```

```
   private void exibeAVL(AvlNode avl, int t) {
               if (avl != null) {
            exibeAVL(avl.dir, t + 1);
            for (int i = 0;i < t;i++) {
                System.out.printf("   ");
            }
            System.out.printf("%d \n",avl.info);
            exibeAVL(avl.esq, t + 1);
        }
  }
}

// ------------------------------------------------------- Fonte: avl.java

package Dados39;

import java.util.Scanner;

public class AVL {

    public static void main(String [] args) {
        Scanner entrada = new Scanner(System.in);
        AvlTree avl = new AvlTree();
        int x;

        do {
            System.out.print("Info: ");
            x = entrada.nextInt();
            if (x != 0) {
              avl.insertAVL(x);
            }
        } while (x != 0);
        avl.exibeAVL();
    }
}
```

Programa *online* que demonstra inserções e remoções em uma árvore AVL:

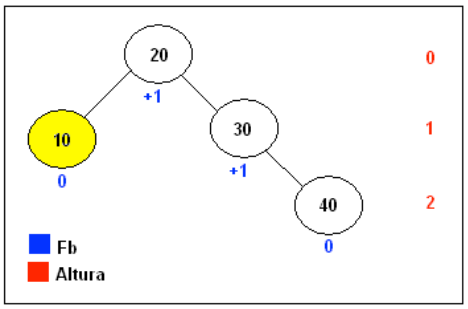
http://www.site.uottawa.ca/~stan/csi2514/applets/avl/BT.html

6.6.2 Remoção em uma árvore AVL

Quando um elemento é removido de uma árvore balanceada **AVL**, é necessário verificar se esta operação quebrou a propriedade de balanceamento da árvore, ou seja, é necessário calcular o **Fator de Balanceamento** dos nodos da árvore. Se o **Fb** de algum nodo for diferente de 0, 1 ou -1, é necessário reestruturar a árvore para que volte a ser balanceada.

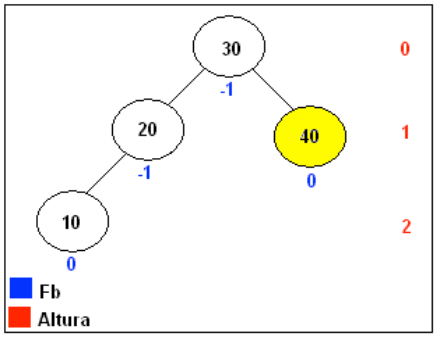
Na remoção de um nodo em uma árvore AVL, pode-se ter quatro situações distintas, cuja a forma de tratamento é diferente:

a) Remoção do nó: 10



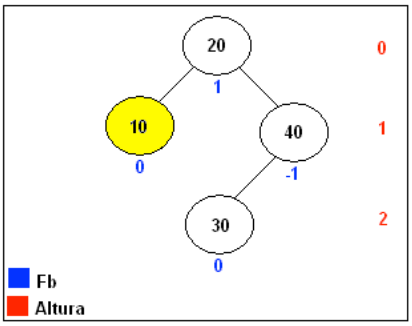
Resultado: Com a remoção do nodo **10** a árvore ficará desbalanceada, pois o nodo raiz ficará com Fb = 2. A solução será fazer uma **rotação simples à esquerda**.

b) Remoção do nó: 40



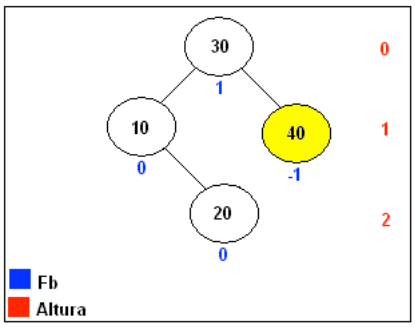
Resultado: Com a remoção do nodo **40** a árvore ficará desbalanceada, pois o nodo raiz ficará com Fb = -2. A solução será fazer uma **rotação simples à direita**.

c) Remoção do nó: 10



Resultado: Com a remoção do nodo **10** a árvore ficará desbalanceada, pois o nodo raiz ficará com Fb = 2. A solução será fazer uma **rotação dupla à esquerda** (rotação simples à direita + rotação simples à esquerda).

d) Remoção do nó: 40



Resultado: Com a remoção do nodo **40** a árvore ficará desbalanceada, pois o nodo raiz ficará com Fb = 2. A solução será fazer uma **rotação dupla à direita** (rotação simples à esquerda + rotação simples à direita).

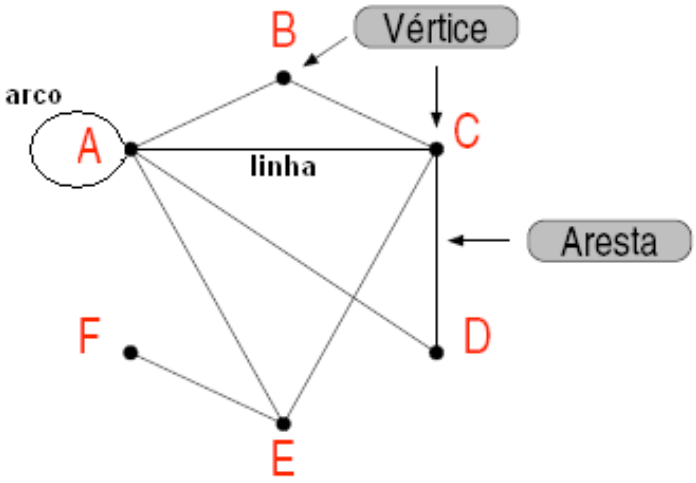
7. Grafos

7.1 Conceitos

Um grafo **G** é definido como G (V, A) onde V é um conjunto finito e não vazio de **vértices** e A é um conjunto finito de **arestas**, ou seja, linhas, curvas ou setas que interligam dois vértices.

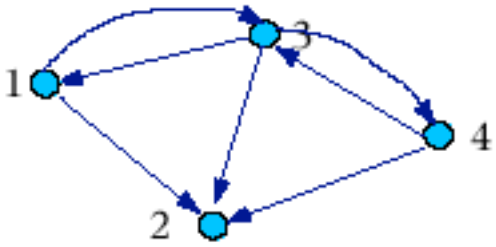
"Grafo é um par ordenado de conjuntos disjuntos (V, A), onde V é um conjunto arbitrário que se designa por conjunto dos vértices e A um subconjunto de pares não ordenados de elementos (distintos) de V que se designa por conjunto das arestas."

Um **vértice** é representado por um ponto ou círculo representando um nó, nodo ou informação. Uma **aresta** pode ser uma reta, seta ou arco representando uma relação entre dois nodos.



Quando uma **aresta** possui indicação de sentido (uma seta), ela é chamada de **arco**, caso contrário é chamada de **linha** (veja grafo acima).

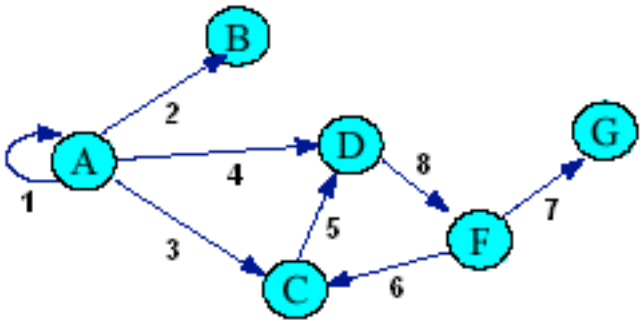
Orientação é a direção para a qual uma seta aponta, um grafo deste tipo é chamado **grafo dirigido** ou **orientado**.



G(4,7) – 4 vértices e 7 arestas

Cardinalidade (ordem) de um conjunto de vértices é igual a quantidade de seus elementos. Grafo **denso** possui alta cardinalidade de vértices, enquanto que grafo **pouco povoado** possui baixa cardinalidade.

A **ordem** (V) de um grafo G é o número de vértices do grafo enquanto que a **dimensão** (A) é o número de arestas do grafo. No exemplo acima, a ordem do grafo é **4** enquanto que a dimensão é **7**.

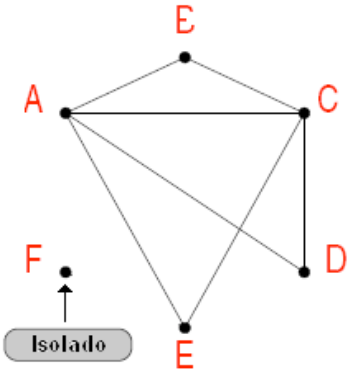


O conjunto de arcos do grafo acima é:

{<A,A>, <A,B>, <A,C>, <A,D>, <C,D>, <F,C>, <F,G>, <D,F>}

1 2 3 4
5 6 7 8

Um vértice que não possui nenhuma aresta incidente é chamado de **isolado** (figura abaixo). Um grafo com nenhum vértice é chamado de **vazio.**



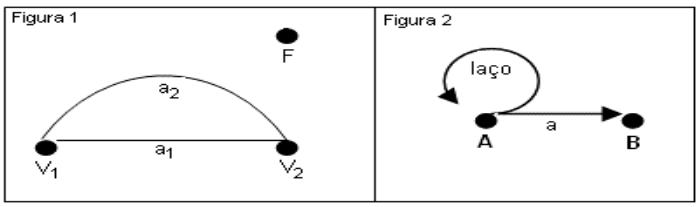
Passeio é uma seqüência de vértices e arestas onde o caminho é um passeio sem vértices repetidos (seqüência de vértices em que cada dois vértices consecutivos são ligados por um arco). **Trajeto** é um passeio sem arestas repetidas.

Passeio é uma seqüência <v0, a1, v1, a2, ..., vk-1, ak, vk> onde v0, v1,....,vk são **vértices**, a1, a2,...., ak são **arcos** e, para cada (i, ai) é um arco de vi-1 a vi. O vértice v0 é o início do passeio e o vértice vk é o seu término.

Um **caminho** é um *passeio* sem vértices repetidos. A **dimensão** de um caminho ou trajeto é chamado de **comprimento**.

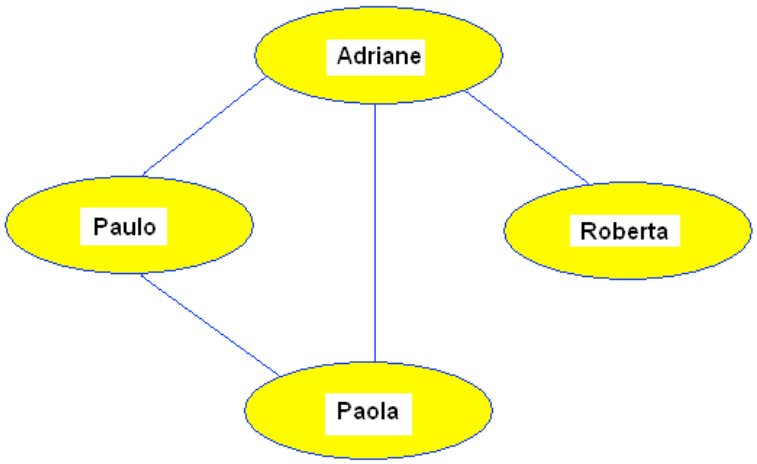
Ciclo é um caminho de comprimento não nulo fechado, ou seja, tem os vértices extremos iguais (é um passeio onde v0 = vk.). **Circuito** é um trajeto de comprimento não nulo fechado (é um ciclo sem vértices, com exceção feita a v0 e vk).

Laço é uma aresta que retorna ao mesmo vértice. Se a aresta não tiver seta ela é dita sem orientação. Diz-se que uma aresta é **incidente** com os vértices que ela liga, não importando a orientação. Dois vértices são **adjacentes** se estão ligados por uma aresta. Um vértice é dito **isolado** se não existe aresta incidente sobre ele. Duas arestas incidentes nos mesmos vértices, não importando a orientação, a1 = (v,w) e a2 = (v,w), então as arestas são **paralelas**.



Diz-se que o grafo é **conexo** se para cada par de vértices existe pelo menos um passeio que os une.

Chama-se grafo **não-orientado** ou **não-dirigido** se as arestas representam relacionamento nas duas direções, ou seja, as arestas não possuem uma seta indicando sentido.

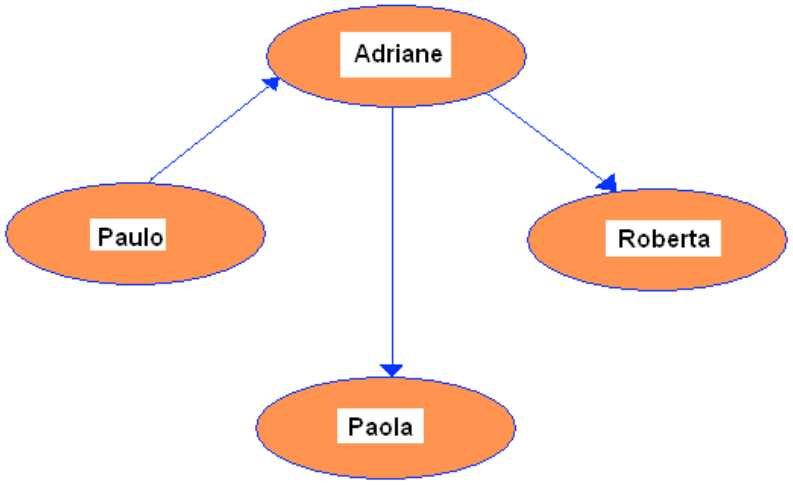


Grafo não-orientado ou não-dirigido

V = { Paulo, Adriane, Paola, Roberta }
A = { (Paulo, Adriane) , (Paulo, Paola) , (Adriane, Paola) , (Adriane, Roberta) }

Explicação do grafo acima: O exemplo representa uma relação de amizade, ou seja, se *Paulo* é amigo de *Adriane* o inverso é verdade, isto se chama: *relação simétrica*.

Quando um grafo possui **arcos** (seta indicando uma direção) ele é denominado **grafo dirigido** ou **dígrafo** (veja próxima figura).

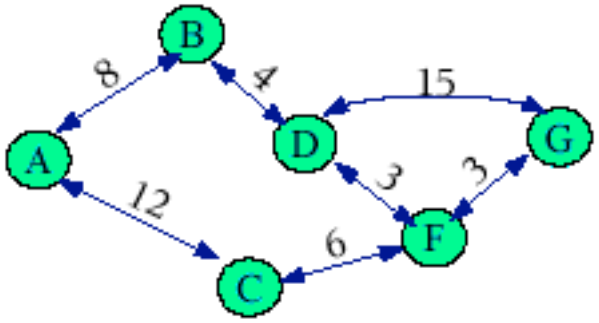


Grafo dirigido ou dígrafo

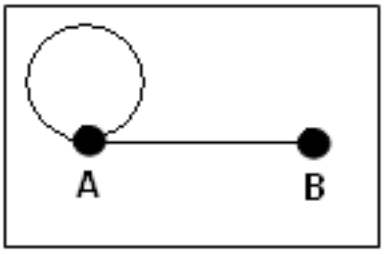
Explicação do grafo acima: O exemplo representa uma relação de subordinação, ou seja, se "*Paulo*" é chefe de "*Adriane*" o inverso **não** é verdade, isto se chama: *relação não-simétrica*.

Grau de um vértice, em um grafo não-dirigido, é o número de arestas incidentes ao vértice. Em um grafo dirigido, pode-se dividir o grau em dois: **grau de emissão** (número de arestas que saem do vértice) e **grau de recepção** (número de arestas que chegam no vértice).

Toda árvore é um grafo, mas nem todo grafo é uma árvore. Um grafo onde existe um número associado a cada arco (**peso**) é chamado de **rede** ou **grafo ponderado**. Um exemplo deste tipo é um grafo representando cidades e distâncias entre as cidades (veja figura abaixo).



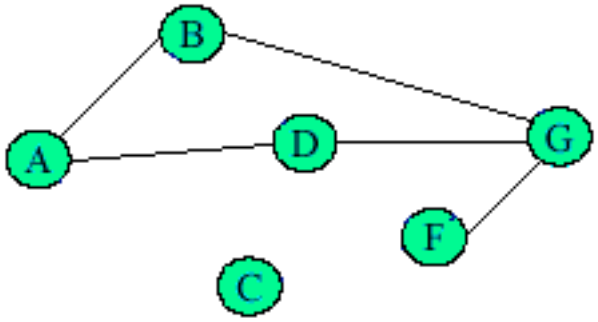
Grau de um Vértice: É igual ao número de arestas que são incidentes ao vértice. Um laço é contado duas vezes.



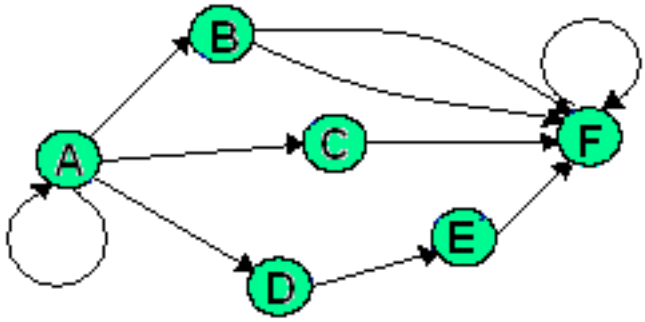
No exemplo acima, o vértice **A** tem grau 3 enquanto que o vértice **B** tem grau 1. Em um grafo dirigido o grau de um vértice é a soma do número de arestas que saem e chegam no vértice.

Tipos de Grafos

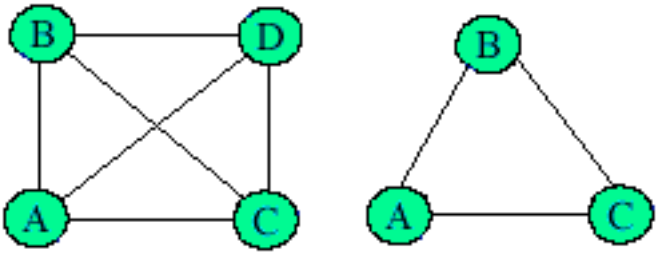
a) **Simples:** É um grafo que não possui laços nem arestas paralelas.



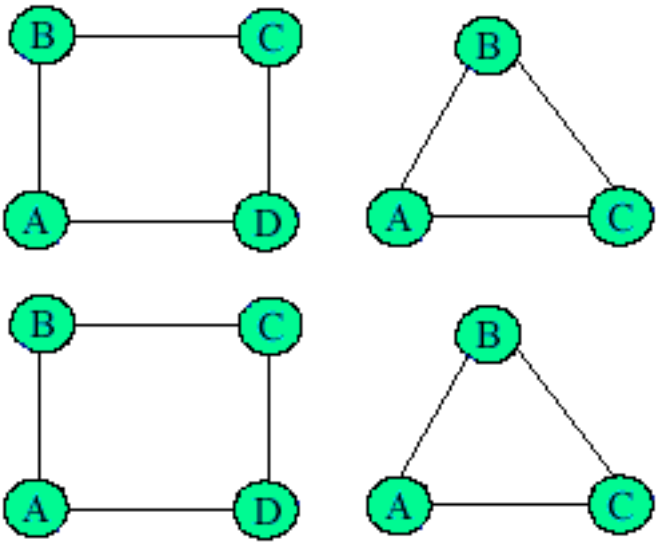
b) **Dirigido** (dígrafo ou direcionado): Consiste de dois conjuntos finitos:
a) vértices e b) arestas dirigidas, onde cada aresta é associada a um par ordenado de vértices chamados de nós terminais.



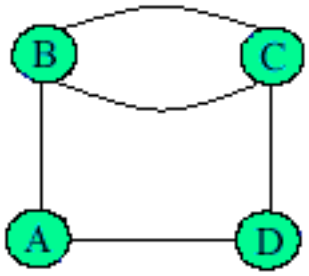
c) **Completo:** Um grafo completo de n vértices, denominado Kn, é um grafo simples com n vértices v1, v2, . . . , vn, cujo conjunto de arestas contém exatamente uma aresta para cada par de vértices distintos.



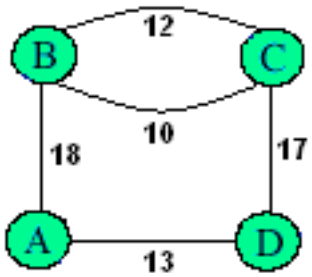
d) **Ciclo:** Um grafo ciclo de n vértices, denominado C_n, onde n é maior ou igual a 3, é um grafo simples com n vértices v_1, v_2, . . . , v_n, e arestas v_1v_2, v_2v_3, . . ., $v_{n-1}v_n$, v_nv_1.



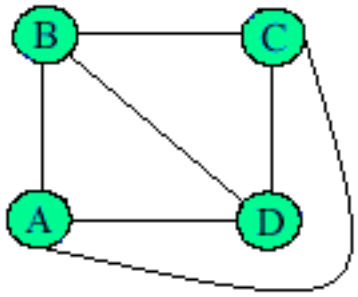
e) **Multigrafo:** É um grafo que não possui laços, mas pode ter arestas paralelas.



f) **Valorado:** É um grafo em que cada aresta tem um valor associado (peso), ou seja, possui um conjunto de valores (pesos) associados a cada aresta.

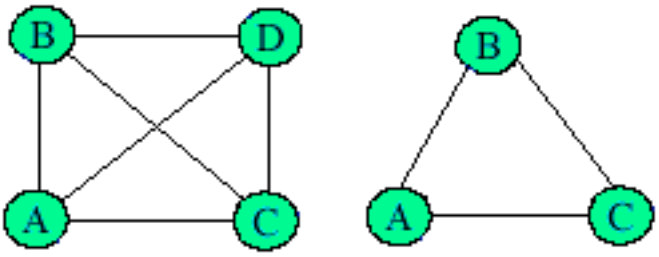


g) **Planar:** É um grafo onde não há cruzamento de arestas.



h) **Imersível:** Um grafo é imersível em uma superfície S se puder ser representado geograficamente em S de tal forma que arestas se cruzem nas extremidades (vértices). Um **grafo planar** é um grafo que é imersível no plano. Um exemplo de grafo imersível é a representação das conexões de uma placa de circuito impresso, onde as arestas não podem se cruzar, ou seja, os cruzamentos são permitidos apenas nas extremidades.

i) **Regular:** Um grafo é regular quando todos os seus vértices têm o mesmo grau. Grafos completos com 2, 3, 4, e 5 vértices são grafos regulares.



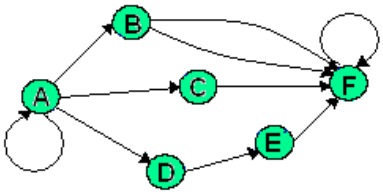
Grafos podem ser representados de duas formas: **Matriz de Adjacências** (forma apropriada para representar grafos densos) ou **Lista de Adjacências** (forma apropriada para representar grafos esparsos).

7.2 Representação por Lista e Matriz de Adjacências

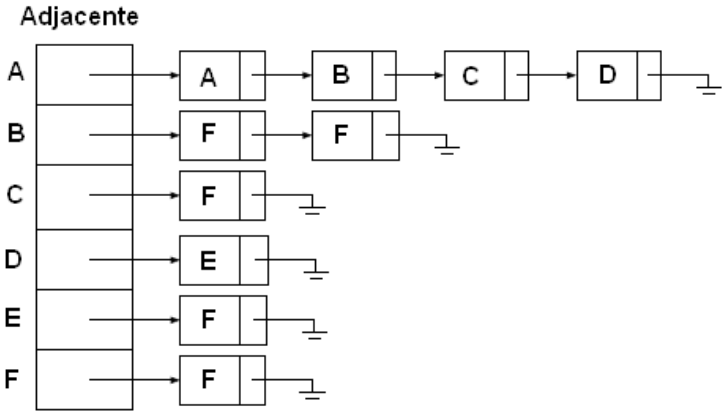
7.2.1 Lista de Adjacências

Um grafo pode ser representado por uma lista **Adjacente**[v_i] = [v_a, v_b] onde v_a, v_b, ... representam os vértices que se relacionam com o vértice v_i.

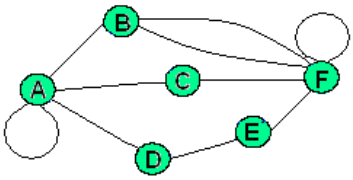
Lista de Adjacências para um grafo dirigido:



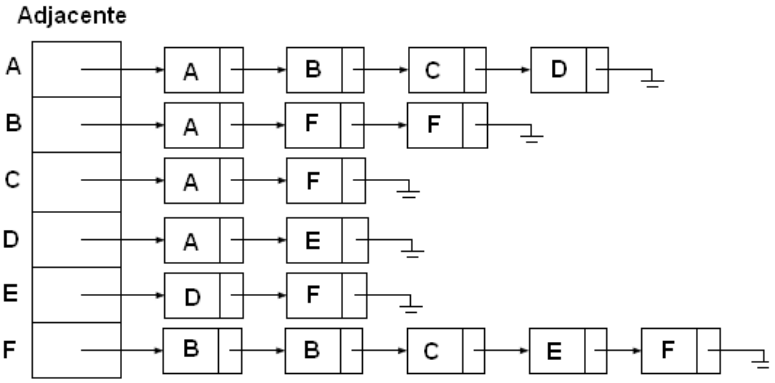
Adjacente[A] = [A, B, C, D]
Adjacente[B] = [F, F]
Adjacente[C] = [F]
Adjacente[D] = [E]
Adjacente[E] = [F]
Adjacente[F] = [F]



Lista de Adjacências para um grafo não-dirigido:



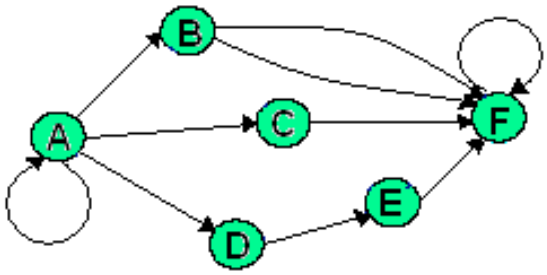
Adjacente[A] = [A, B, C, D]
Adjacente[B] = [A, F, F]
Adjacente[C] = [A, F]
Adjacente[D] = [A, E]
Adjacente[E] = [D, F]
Adjacente[F] = [B, B, C, E, F]



7.2.2 Matriz de Adjacências

Um grafo pode ser representado por uma matriz **A** = (a_{ij}), onde a_{ij} representa o número de arestas de v_i **para** v_j.

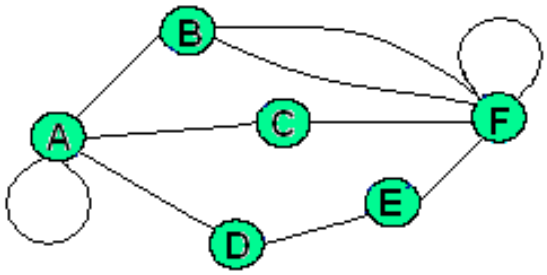
Matriz de Adjacências para um grafo dirigido:



A

	A	B	C	D	E	F
A	1	1	1	1	0	0
B	0	0	0	0	0	2
C	0	0	0	0	0	1
D	0	0	0	0	1	0
E	0	0	0	0	0	1
F	0	0	0	0	0	1

Matriz de Adjacências para um grafo não-dirigido:



A

	A	B	C	D	E	F
A	1	1	1	1	0	0
B	1	0	0	0	0	2
C	1	0	0	0	0	1
D	1	0	0	0	1	0
E	0	0	0	1	0	1
F	0	2	1	0	1	1

7.3 Percurso em Amplitude e Percurso em Profundidade

Existem dois critérios para percorrer grafos: **Percurso em Amplitude** e **Percurso em Profundidade.**

Em ambos os percursos parte-se de um nodo qualquer escolhido arbitrariamente e visita-se este nodo. A seguir, considera-se cada um dos nodos adjacentes ao nodo escolhido.

Percurso em amplitude ou caminhamento em amplitude:

a) Seleciona-se um vértice para iniciar o caminhamento.
b) Visitam-se os vértices adjacentes, marcando-os como visitados.
c) Coloca-se cada vértice adjacente numa fila.

d) Após visitar os vértices adjacentes, o primeiro da fila torna-se o novo vértice inicial. Reinicia-se o processo.
e) O caminhamento termina quanto todos os vértices tiverem sido visitados ou o vértice procurado for encontrado.

Percurso em profundidade ou caminhamento em profundidade:

a) Seleciona-se um vértice para iniciar o caminhamento.
b) Visita-se um primeiro vértice adjacente, marcando-o como visitado.
c) Coloca-se o vértice adjacente visitado numa pilha.
d) O vértice visitado torna-se o novo vértice inicial.
e) Repete-se o processo até que o vértice procurado seja encontrado ou não haja mais vértices adjacentes. Se verdadeiro, desempilha-se o topo e procura-se o próximo adjacente, repetindo o algoritmo.
f) O processo termina quando o vértice procurado for encontrado ou quando a pilha estiver vazia e todos os vértices tiverem sido visitados.

7.4 Determinação do Caminho Mínimo

O caminho de um vértice a outro vértice é mínimo se não existe outro caminho entre eles que tenha menos arcos.

O problema de encontrar o caminho mais curto entre dois nós de um grafo ou uma rede é um dos problemas clássicos da Ciência da Computação. Este problema consiste, genericamente, em encontrar o caminho de menor custo entre dois nós da rede, considerando a soma dos custos associados aos arcos percorridos.

O mais famoso algoritmo para resolver o problema de caminho mínimo em grafos é o algoritmo de **Dijkstra** (1959). Este algoritmo apenas funciona se os custos associados aos arcos não forem negativos, mas isso não é muito importante na maioria dos problemas práticos pois, em geral, os custos associados aos arcos são em geral grandezas fisicamente mensuráveis.

7.4.1 Algoritmo de Dijkstra

Dado um grafo, G=(V,E) , dirigido ou não, com valores não negativos em cada arco ou ramo, utiliza-se o

algoritmo de **Dijkstra** para encontrar a distância mínima entre um vértice inicial (**s**) e um vértice final (**v**). Ele determina a distância mínima entre **s** e os outros vértices na ordem dessas distâncias mínimas, ou seja, os vértices que se encontram mais próximos de **s** em primeiro lugar.

O algoritmo vai usar dist(v), uma estrutura que armazena e referencia a distância de **s** ao nó **v**. De início, dist(s)=0 , pois a menor distância de **s** a si mesmo será sempre zero.

O símbolo LIMITE representa um valor maior que o comprimento de qualquer caminho sem ciclos em G. Por outro lado, se dist(v) = LIMITE, indica que ainda não foi encontrado nenhum caminho com distância mínima entre s e v. De início, dist(v) = LIMITE. Será utilizado um conjunto S que contém os vértices cuja distância a s é mínima e conhecida naquele momento da execução do algoritmo. Esta distância será definitiva para cada um deles. Este conjunto será de início constituído somente pelo nó s, com dist(s) = 0.

Descrição do Algoritmo

Começa-se considerando a distância ao próprio s como zero. Faze-se então dist(s)= 0. Para todos os outros vértices, considera-se a sua distância a s como valendo LIMITE. Fazer então dist(v) = LIMITE.

O processo de introdução de um nó Vn não pertencente a S, assumindo portanto que V(S) é diferente do conjunto vazio, consiste no seguinte:

1. Qualquer um que seja Vm não pertencente a S tal que v foi o último nó a 'entrar' em S e (V, Vm) pertencente ao conjunto V, se dist(Vm) > dist(V) + distância associada a (V,Vm) então dist(Vm) = dist(V) + distância associada a (V, Vm);

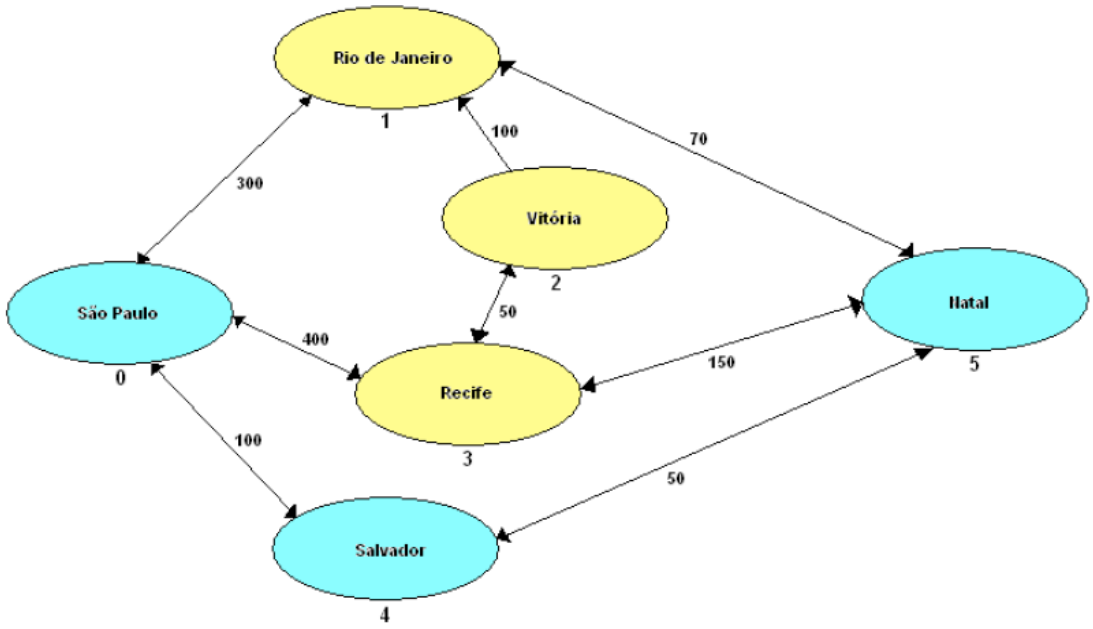
2. Determinar qualquer que seja Vm não pertencente a S o menor de entre os valores de dist(vm). Seja dist(vj) esse valor;

3. Fazer Vn = Vj , dist(Vn) = dist(Vj) e Vn passa a ser o novo elemento de S;

Se o vértice vn coincidir com o vértice final então dist(Vn) é a menor distância entre s e Vn , e parar a execução. Se não coincidir, voltam-se a repetir os passos 1 , 2 e 3.

Se não for possível aplicar aqueles passos, ou porque V(S) igual ao conjunto vazio ou qualquer que seja Vm não pertencente a S então (V, Vm) não pertence ao conjunto V, então não é possível determinar a distância mínima de s ao vértice final.

Problema: Escreva um programa em Java que cria um grafo representando a ligação entre seis cidades com suas respectivas distâncias (São Paulo, Rio de Janeiro, Vitória, Recife, Salvador e Natal). O programa deve permitir a entrada da cidade origem (0..5) e da cidade destino (0..5) e exibir o caminho mínimo entre estas duas cidades através da utilização do **Algoritmo de Dijkstra.**



Programa exemplo (40): O programa a seguir exibe o caminho mínimo entre duas cidades qualquer (se existir) através da utilização do algoritmo de **Dijkstra.**

```
// ---------------------------------------------------------------- Fonte: Grafo.java

package dados40;

public class Grafo {
    final int maxVertices = 6;
    final int limite = 32767;
    int [][] valor = new int[maxVertices][maxVertices];
    char [][] adj = new char[maxVertices][maxVertices];
    String [] rotulos = {"Spa", "Rio", "Vit", "Rec", "Sal", "Nat"};
```

```
// ----------------------------------- inicializaGrafo

public void inicializaGrafo() {
    for (int l = 0; l < maxVertices; l++) {
        for (int c = 0; c < maxVertices; c++) {
            valor[l][c] = limite;
           adj[l][c] = 'N';
        }
    }
}

// ----------------------------------- criaAresta

public void criaAresta(int origem, int destino, int info) {
    adj[origem][destino] = 'S';
    valor[origem][destino] = info;
    adj[destino][origem] = 'S';
    valor[destino][origem] = info;
}

// ----------------------------------- imprimeGrafo

public void imprimeGrafo () {
    for (int l = 0; l < maxVertices; l++) {
       System.out.printf("   %d", l);
    }
    newline();
    for (int l = 0; l < maxVertices; l++) {
       System.out.printf(" %s", rotulos[l]);
    }
    newline();
}

// ----------------------------------- imprimeMatriz

public void imprimeMatriz() {
    System.out.printf("Matriz de Adjacencias\n");
    for (int l = 0; l < maxVertices; l++) {
        for (int c = 0; c < maxVertices; c++) {
            System.out.printf("%5d ",valor[l][c]);
        }
        newline();
    }
    newline();
}

// ---------------------------------------------- dijkstra

public void dijkstra(int origem, int destino, int [] precede) {
    int k, result;
    int aux1, aux2;
    int [] distancia = new int[maxVertices];
    boolean [] menorCaminho = new boolean[maxVertices];
    int atual, dc, menordist, novadist;
    char parar = 'N';

    for (int i = 0; i < maxVertices; i++) {
        distancia[i] = limite;
        menorCaminho [i] = false;
        precede[i] = -1;
    }
    menorCaminho [origem] = true;
    distancia[origem] = 0;
    atual = origem;
    k = atual;
    while (atual != destino && parar == 'N') {
        menordist = limite;
```

```
        dc = distancia[atual];
        for (int i = 0; i < maxVertices; i++) {
            if (menorCaminho [i] == false) {
                if (adj[atual][i] =='S') {
                    novadist = dc + valor[atual][i];
                }
                else {
                     novadist = valor[atual][i];
                }
                if (novadist < distancia[i]) {
                    distancia[i] = novadist;
                    precede[i] = atual;
                }
                if (distancia[i] < menordist) {
                    menordist = distancia[i];
                    k = i;
                }
            }
        }
        if (atual == k) {
            parar = 'S';
        }
        else {
              atual = k;
              menorCaminho[atual] = false;
        }
   }
   result = distancia[destino];
              if (result == limite || result == 0) {
            System.out.printf("\nNão há trajeto entre %s e %s",
                        rotulos[origem], rotulos[destino]);
        }
        else {
            System.out.printf("\nMenor caminho entre %s e %s = %d",
                        rotulos[origem], rotulos[destino], result);
            System.out.printf("\nCaminho INVERSO Percorrido: ");
            aux1 = precede[destino];
            aux2 = destino;
            while (aux1 != origem) {

                System.out.printf("\n %s -> %s (%d)", rotulos[aux1],
                                rotulos[aux2], valor[aux1][aux2]);
               aux2 = aux1;
               aux1 = precede[aux1];
                System.out.printf("\n %s -> %s (%d)", rotulos[aux1],
                                rotulos[aux2], valor[aux1][aux2]);
            }
        }

}

// ------------------------------------ imprimeCidades

public void imprimeCidades() {
    String [] cidades = {"[0] - (Spa) - Sao Paulo",
                         "[1] - (Rio) - Rio de janeiro",
                         "[2] - (Vit) - Vitoria",
                         "[3] - (Rec) - Recife",
                         "[4] - (Sal) - Salvador",
                         "[5] - (Nat) - Natal"};
    for (int i = 0;i < maxVertices;i++) {
       System.out.printf("%s\n",cidades[i]);
    }
    newline();
 }

 // ------------------------------------ newline
```

```
    public void newline() {
        System.out.println();
    }

}

// ------------------------------------------------------ Fonte: Dados40.java

package dados40;

import java.util.Scanner;

public class Dados40 {

    public static void main(String[] args) {
        final int maxVertices = 6;
        final int limite = 32767;
        Grafo grafo = new Grafo();
        int [] precede = new int [maxVertices];
        char tecla;
        int origem, destino;
        Scanner entrada = new Scanner(System.in);
        String s;

        grafo.inicializaGrafo();
        grafo.criaAresta(0, 1, 300);
        grafo.criaAresta(0, 3, 400);
        grafo.criaAresta(0, 4, 100);
        grafo.criaAresta(1, 2, 100);
        grafo.criaAresta(1, 5, 70);
        grafo.criaAresta(2, 3, 50);
        grafo.criaAresta(4, 5, 50);
        grafo.criaAresta(3, 5, 150);
        do {
            grafo.imprimeMatriz();
            grafo.imprimeCidades();
            grafo.imprimeGrafo();
            int tam = maxVertices - 1;
            System.out.printf("\n Origem [0..%d]: ", tam);
            do {
                s = entrada.nextLine();
                origem = Integer.parseInt(s);
            } while (origem < 0 || origem > tam);
            System.out.printf("Destino [0..%d]: ", tam);
            do {
                s = entrada.nextLine();
                destino = Integer.parseInt(s);
            } while (destino < 0 || destino > tam);
            grafo.dijkstra(origem, destino, precede);
            System.out.println();
            System.out.printf("\nRepetir [S/N]? ");
            do {
                s = entrada.nextLine();
                tecla = s.charAt(0);
            } while (!strChr("SsNn", tecla));
         } while (strChr("Ss",tecla));
         System.exit(0);
    }

    // ------------------------------------------------------ strChr

    static boolean strChr(String s, char ch) {

        for (int i = 0; i < s.length(); i++) {
            if (s.charAt(i) == ch) {
                return(true);
            }
        }
```

```
            return(false);
        }
    }
```

8. Programas exemplos

A seguir serão vistos alguns exemplos da utilização das estruturas de dados vistas neste livro.

8.1 Torre de Hanoi (Pilha - *Stack*)

A seguir é visto a implementação de uma Torre de Hanoi em Java usando a classe ***Stack*** (Pilha).

```
// ------------------------------------------------- Fonte: Torre.java

package torre;

import java.util.Stack;
import java.util.Scanner;

public class Torre {

    public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);
        String s;
        int in, out;
        Stack p1 = new Stack();
        Stack p2 = new Stack();
        Stack p3 = new Stack();
        boolean ganhou = false, moveu = false;

        p1.push(3);
        p1.push(2);
        p1.push(1);
        exibePilhas(p1, p2, p3);
        do {
            do {
                System.out.print("Move: ");
                s = entrada.nextLine();
                in = Integer.parseInt(s);
                if (in == 0) {
                   System.exit(0);
                }
            } while (in < 1 || in > 3);
            do {
                System.out.print("Para: ");
                s = entrada.nextLine();
                out = Integer.parseInt(s);
                if (in == 0) {
                   System.exit(0);
                }
            } while (out < 1 || out > 3);
            if (testaMovimento(p1, p2, p3, in)) {
               if (testaMaior(p1, p2, p3, in, out)) {
                   moveu = movePilha(p1, p2, p3, in, out);
               }
            }
            exibePilhas(p1, p2, p3);
            if (moveu) {
               ganhou = testaVitoria(p1);
               if (ganhou) {
                   System.out.println("Status: VENCEDOR (p1)");
                   System.exit(0);
               }
               ganhou = testaVitoria(p2);
               if (ganhou) {
```

```
                System.out.println("Status: VENCEDOR (p2)");
                System.exit(0);
            }
            ganhou = testaVitoria(p3);
            if (ganhou) {
                System.out.println("Status: VENCEDOR (p3)");
                System.exit(0);
            }
        }
    } while (true);
}

// ------------------------------------------- testaVitoria

static boolean testaVitoria(Stack p1) {
    int a, b, c;

    a = p1.search(1);
    b = p1.search(2);
    c = p1.search(3);
    if (a != -1 && b != -1 && c != -1) {
        return(true);
    }
    else {
        return(false);
    }
}

// ------------------------------------------- testaMaior

static boolean testaMaior(Stack p1, Stack p2, Stack p3, int in, int out) {
    Object nodo;
    int a = 0, b = 0;

    switch (in) {
        case 1: nodo = p1.peek();
                a = nodo.hashCode();
                break;
        case 2: nodo = p2.peek();
                a = nodo.hashCode();
                break;
        case 3: nodo = p3.peek();
                a = nodo.hashCode();
                break;
    }
    switch (out) {
        case 1: if (p1.empty()) {
                   return(true);
                }
                else {
                      nodo = p1.peek();
                      b = nodo.hashCode();
                }
                break;
        case 2: if (p2.empty()) {
                   return(true);
                }
                else {
                      nodo = p2.peek();
                      b = nodo.hashCode();
                }
                break;
        case 3: if (p3.empty()) {
                   return(true);
                }
                else {
                      nodo = p3.peek();
                      b = nodo.hashCode();
```

```
                }
                break;
    }
    if (a < b)
        return(true);
    else {
        return(false);
    }
}

// ------------------------------------------ testaMovimento

static boolean testaMovimento(Stack p1, Stack p2, Stack p3, int in) {
    boolean pode = false;

    switch (in) {
        case 1: if (p1.empty()) {
                    System.out.println("Status: Impossivel mover (p1)");
                }
                else {
                    pode = true;
                }
                break;
        case 2: if (p2.empty()) {
                    System.out.println("Status: Impossivel mover (p2)");
                }
                else {
                    pode = true;
                }
                break;
        case 3: if (p3.empty()) {
                    System.out.println("Status: Impossivel mover (p3)");
                }
                else {
                    pode = true;
                }
                break;
    }
    return(pode);
}

// ------------------------------------------ movePilha

static boolean movePilha(Stack p1, Stack p2, Stack p3, int a, int b) {
    Object nodo;
    int in = 0;
    boolean moveu = false;

    switch (a) {
        case 1: if (!p1.empty()) {
                    nodo = p1.pop();
                    in = nodo.hashCode();
                }
                else {
                    in = 0;
                }
                break;
        case 2: if (!p2.empty()) {
                    nodo = p2.pop();
                    in = nodo.hashCode();
                }
                else {
                    in = 0;
                }
                break;
        case 3: if (!p3.empty()) {
                    nodo = p3.pop();
                    in = nodo.hashCode();
```

```
                }
                else {
                    in = 0;
                }
                break;
        }
        if (in != 0) {
           switch (b) {
               case 1: p1.push(in);
                       moveu = true;
                       break;
               case 2: p2.push(in);
                       moveu = true;
                       break;
               case 3: p3.push(in);
                       moveu = true;
                       break;
           }
        }
        return(moveu);
    }

    // ---------------------------------------- exibePilhas

    static void exibePilhas(Stack p1, Stack p2, Stack p3) {
        System.out.println("p1: " + p1);
        System.out.println("p2: " + p2);
        System.out.println("p3: " + p3);
    }
```

Observação: Subistitua o método **exibePilhas** pelo método abaixo.

```
// ---------------------------------------- exibePilhas

    static void exibePilhas(Stack p1, Stack p2, Stack p3) {
        char a1 = '|', a2 = '|', a3 = '|';
        char b1 = '|', b2 = '|', b3 = '|';
        char c1 = '|', c2 = '|', c3 = '|';
        Object nodo;

        nodo = p1.search(1);
        a1 = nodo.toString().charAt(0);
        if (a1 != '-') {
            a1 = '1';
        }
        nodo = p1.search(2);
        a2 = nodo.toString().charAt(0);
        if (a2 != '-') {
            a2 = '2';
        }
        nodo = p1.search(3);
        a3 = nodo.toString().charAt(0);
        if (a3 != '-') {
            a3 = '3';
        }

        nodo = p2.search(1);
        b1 = nodo.toString().charAt(0);
        if (b1 != '-') {
            b1 = '1';
        }
        nodo = p2.search(2);
        b2 = nodo.toString().charAt(0);
        if (b2 != '-') {
            b2 = '2';
        }
        nodo = p2.search(3);
```

```
        b3 = nodo.toString().charAt(0);
        if (b3 != '-') {
            b3 = '3';
        }

        nodo = p3.search(1);
        c1 = nodo.toString().charAt(0);
        if (c1 != '-') {
            c1 = '1';
        }
        nodo = p3.search(2);
        c2 = nodo.toString().charAt(0);
        if (c2 != '-') {
            c2 = '2';
        }
        nodo = p3.search(3);
        c3 = nodo.toString().charAt(0);
        if (c3 != '-') {
            c3 = '3';
        }
        if (c3 == '-' && c2 == '-' && c1 != '-') {
            c3 = c1;
            c1 = '-';
        }
        if (c3 == '-' && c2 != '-' && c1 == '-') {
            c3 = c2;
            c2 = '-';
        }
        if (c3 == '-' && c2 != '-' && c1 != '-') {
            c3 = c2;
            c2 = c1;
            c1 = '-';
        }
        if (b3 == '-' && b2 == '-' && b1 != '-') {
            b3 = b1;
            b1 = '-';
        }
        if (b3 == '-' && b2 != '-' && b1 == '-') {
            b3 = b2;
            b2 = '-';
        }
        if (b3 == '-' && b2 != '-' && b1 != '-') {
            b3 = b2;
            b2 = b1;
            b1 = '-';
        }
        if (a3 == '-' && a2 == '-' && a1 != '-') {
            a3 = a1;
            a1 = '-';
        }
        if (a3 == '-' && a2 != '-' && a1 == '-') {
            a3 = a2;
            a2 = '-';
        }
        if (a3 == '-' && a2 != '-' && a1 != '-') {
            a3 = a2;
            a2 = a1;
            a1 = '-';
        }
        System.out.printf("    %c         %c         %c    \n", a1, b1, c1);
        System.out.printf("    %c         %c         %c    \n", a2, b2, c2);
        System.out.printf("    %c         %c         %c    \n", a3, b3, c3);
        System.out.printf("--------- --------- ---------\n");
        System.out.printf("   p1        p2        p3\n\n");
    }
}
```

8.2 Analisador de Expressões (Pilha - *Stack*)

Problema: Implementar em Java um **Analisador de Expressões** utilizando Pilhas.

Exemplo: (3 * (4 + 5))

String	0	1	2	3	4	5	6	7	8
	(	3	+	(	4	*	5	)	)

Dicas:

1. Crie duas PILHAS (números e operandos);
2. "(" não faça nada;
3. Número (***Push*** pilha 2) empilhe na pilha de números;
4. Operando (***Push*** pilha 1) empilhe na pilha de operandos;
5. ")" (***Pop*** pilha 2, ***Pop*** pilha 2 e ***Pop*** pilha 1) execute a operação;
6. Até que i = e[0]

Valores Válidos:

Números: 0 1 2 3 4 5 6 7 8 9 (**Números Inteiros Positivos**)
Operandos: + - * /

	topo	0	1	2	3	4	5	6	7	8	9
P1	2	3	4	5	-	-	-	-	-	-	
P2	1	*	+	-	-	-	-	-	-	-	

	topo	0	1	2	3	4	5	6	7	8	9
P1	1	3	9	-	-	-	-	-	-	-	
P2	0	*	-	-	-	-	-	-	-	-	

	topo	0	1	2	3	4	5	6	7	8	9
P1	0	27	-	-	-	-	-	-	-	-	
P2	-1	-	-	-	-	-	-	-	-	-	

// ---------- Fonte: Dados25.java

```
package dados25;

public class PilhaInt {
    final int m = 50;
    final int SUCESSO = 0;
    final int PILHA_CHEIA = 1;
    final int PILHA_VAZIA = 2;
    int topo;
    int [] elem = new int[m];

    // ------------------------------------------- criaPilha

    public void criaPilha() {
        topo = -1;
    }

    // ------------------------------------------- push

    public int push(int dado) {
        if (topo == m - 1) {
           return(PILHA_CHEIA);
        }
        else {
            topo++;
            elem[topo] = dado;
            return(SUCESSO);
        }
     }

     // ------------------------------------------- pop

     public int pop() {
         if (topo != -1) {
             topo--;
             return(elem[topo + 1]);
         }
         return(0);
     }

     // ------------------------------------------- exibePilha

     public void exibePilha() {
         if (topo != -1) {
             System.out.print("PilhaInt: ");
             for (int i = topo;i >= 0;i--) {
                  System.out.print(elem[i] + " ");
             }
             System.out.println();
         }
     }
}

// ------------------------------------------------------ Fonte: Dados25.java

package dados25;

public class PilhaChar {
    final int m = 50;
    final int SUCESSO = 0;
    final int PILHA_CHEIA = 1;
    final int PILHA_VAZIA = 2;
    int topo;
    char [] elem = new char[m];

    // ------------------------------------------- criaPilha

    public void criaPilha() {
```

```
        topo = -1;
    }

    // ---------------------------------------------- push

    public int push(char dado) {
        if (topo == m - 1) {
           return(PILHA_CHEIA);
        }
        else {
            topo++;
            elem[topo] = dado;
            return(SUCESSO);
        }
     }

     // ---------------------------------------------- pop

     public char pop() {
         if (topo != -1) {
             topo--;
             return(elem[topo + 1]);
         }
         return('0');
     }

     // ---------------------------------------------- exibePilha

     public void exibePilha() {
         if (topo != -1) {
             System.out.print("PilhaChar: ");
             for (int i = topo;i >= 0;i--) {
                  System.out.print(elem[i] + " ");
             }
             System.out.println();
         }
     }
}

// ------------------------------------------------------ Fonte: Dados25.java

package dados25;

import java.util.Scanner;

public class Dados25 {

    public static void main(String[] args) {
        PilhaInt p1 = new PilhaInt();
        PilhaChar p2 = new PilhaChar();
        final int m = 50;
        Scanner entrada = new Scanner(System.in);
        String str;
        int tipo, erro = 0;
        int [] valor = {0};
        char [] operador = {'0'};
        int v1, v2, resposta;

        System.out.println("Analisador de Expressões");
        System.out.print("Expressão: ");
        str = entrada.nextLine();
        int n = str.length();
        char [] s = new char[n];
        for (int i = 0;i < n;i++) {
            s[i] = str.charAt(i);
        }
        if (n > m) {
           System.out.println("ERRO: Expressão muito Longa");
```

```
    }
    else {
        if (testaExpressao(str) == true) {
           p1.criaPilha();
           p2.criaPilha();
           for (int i = 0;i < n;i++) {
                tipo = codifica(s[i], valor, operador);
               switch (tipo) {
                    case 1: erro = p1.push(valor[0]);
                            p1.exibePilha();
                           break;
                   case 2: erro = p2.push(operador[0]);
                            p2.exibePilha();
                           break;
                   case 3: v2 = p1.pop();
                           v1 = p1.pop();

                           operador[0] = p2.pop();

                           resposta = calcula(v1, v2, operador[0]);
                           erro = p1.push(resposta);
                           break;
               }
           }
           if (erro == 0) {
               resposta = p1.pop();
               System.out.println("Resposta: " + resposta);
           }
       }
       else {
            System.out.println("Erro: Expressão Inválida");
       }
    }
}

// ---------------------------------------------- codifica

static int codifica(char ch, int [] valor, char [] op) {
    int codifica = 4;

    if (ch >= '0' && ch <= '9') {
        codifica = 1;
        valor[0] = ch - 48;
    }
    if (ch == '+' || ch == '-' || ch == '*' || ch == '/') {
        codifica = 2;
        op[0] = ch;
    }
    if (ch == ')') {
        codifica = 3;
    }
    return(codifica);
}

// ---------------------------------------------- testaExpressao

static boolean testaExpressao(String s) {
    int abre = 0,fecha = 0;

    int n = s.length();
    for (int i = 0;i < n;i++) {
         if (s.charAt(i) == '(') {
             abre++;
         }
         else {
           if (s.charAt(i) == ')') {
                fecha++;
            }
```

```
            }
        }
        if (abre == fecha) {
           return(true);
        }
        return(false);
    }

    // ---------------------------------------- Calcula

    static int calcula(int v1, int v2, char op) {
        switch (op) {
            case '+': return(v1 + v2);
            case '-': return(v1 - v2);
            case '*': return(v1 * v2);
            case '/': return(v1 / v2);
        }
        return(0);
    }
}
```

8.3 Analisador de Expressões usando Stack

Programa abaixo implementa um **Analisador de Expressões** usando a estrutura ***"Stack"***. O programa resolve expressões do tipo: *(4*(5+(4*3)/(3-1))) ou (3*(4+5))*

Exemplo de entrada de dados:

```
Expressao: (4*(5+(4*3)/(3-1))) <enter>
Pilha: 4.0
Pilha:

Pilha: 4.0 5.0
Pilha: *

Pilha: 4.0 5.0 4.0
Pilha: * +

Pilha: 4.0 5.0 4.0 3.0
Pilha: * + *

Pilha: 4.0 5.0 12.0
Pilha: * +
Pilha: 4.0 5.0 12.0 3.0
Pilha: * + /

Pilha: 4.0 5.0 12.0 3.0 1.0
Pilha: * + / -

Pilha: 4.0 5.0 12.0 2.0
Pilha: * + /
Pilha: 4.0 5.0 6.0
Pilha: * +
Pilha: 4.0 11.0
Pilha: *

Pilha: 44.0
Pilha:
Resposta: 44.0
```

// ---------- Fonte: AnalisadorExpressao.java

```
import java.util.Stack;
import java.util.Scanner;

public class AnalisadorExpressao {

    public static void main(String[] args) {
        Stack numeros = new Stack();
        Stack operadores = new Stack();
        Scanner entrada = new Scanner(System.in);
        String s;
        float resp = 0;

        System.out.print("Expressao: ");
        s = entrada.nextLine();
        int n = s.length();
        if (testaExpressao(s)) {
            for (int i = 0;i < n;i++) {
                if (s.charAt(i) == '(') {
                    continue;
                }
                if ("0123456789".indexOf(s.charAt(i)) != -1) {
                    char ch = s.charAt(i);
                    float t = (float) ch - 48;
                    numeros.push(t);
                    exibePilha(numeros);
                    exibePilha(operadores);
                    System.out.println();
                }
                else {
                    if ("+-*/".indexOf(s.charAt(i)) != -1) {
                        char op = s.charAt(i);
                        operadores.push(op);
                    }
                }
                if (s.charAt(i) == ')') {
                    Object nodo = numeros.pop();
                    String st = nodo.toString();
                    float y = Float.parseFloat(st);
                    nodo = numeros.pop();
                    st = nodo.toString();
                    float x = Float.parseFloat(st);
                    nodo = operadores.pop();
                    String temp = nodo.toString();
                    char op = temp.charAt(0);
                    resp = calculaOperacao(x, y, op);
                    numeros.push(resp);
                    exibePilha(numeros);
                    exibePilha(operadores);
                 }
            }
        System.out.println();
        if (operadores.empty()) {
            Object nodo = numeros.pop();
            String st = nodo.toString();
            resp = Float.parseFloat(st);
        }
        else {
            resp = 0;
            while (!operadores.empty()) {
                Object nodo = numeros.pop();
                String st = nodo.toString();
                float y = Float.parseFloat(st);
                nodo = numeros.pop();
                st = nodo.toString();
                float x = Float.parseFloat(st);
                nodo = operadores.pop();
                String temp = nodo.toString();
                char op = temp.charAt(0);
```

```
            resp = calculaOperacao(x, y, op);
            numeros.push(resp);
            exibePilha(numeros);
            exibePilha(operadores);
        }
    }
    System.out.println("Resposta: " + resp);
    }
    else {
        System.out.println("Erro: Expressao Invalida");
    }
    System.exit(0);
}

// ---------------------------------------- calculaOperacao

static float calculaOperacao(float x, float y, char op) {
    float resp = 0;

    switch (op) {
        case '+': resp = x + y;
                  break;
        case '-': resp = x - y;
                  break;
        case '*': resp = x * y;
                  break;
        case '/': if (y != 0) {
                       resp = x / y;
                  }
                  else {
                        System.out.println("Erro Fatal: Divisão por Zero");
                        System.exit(0);
                  }
                  break;
    }
    return(resp);
}

// ---------------------------------------- testaExpressao

static boolean testaExpressao(String exp) {
    int abre = 0, fecha = 0;

    for (int i = 0;i < exp.length();i++) {
        if (exp.charAt(i) == '(') {
            abre++;
        }
        if (exp.charAt(i) == ')') {
            fecha++;
        }
    }
    if (abre == fecha) {
    return(true);
    }
    else
        return(false);
}

// ---------------------------------------- exibePilha

static void exibePilha(Stack pilha) {
    Object [] nodo = pilha.toArray();

    System.out.print("Pilha: ");
    for (int i = 0;i < nodo.length;i++) {
        System.out.print(nodo[i] + " ");
    }
    System.out.println();
```

```
    }
}
```

8.4 Calculadora Polonesa Reversa

A seguir um programa simples que implementa uma **calculadora Polonesa Reversa,** ou seja, o usuário digita primeiro dois valores e depois o operador.

Exemplo de entrada de dados:

```
Polonesa Reversa: <enter> abandona
3
2
/
1.5
2
*
3
<enter>
by Polonesa Reversa
```

```
// ------------------------------------------------- Fonte: Polonesa.java

package polonesa;

public class Polonesa {

    public static void main(String[] args) {
        PolonesaReversa polonesa = new PolonesaReversa();

        polonesa.calculaPolonesaReversa();
    }
}

// ------------------------------------------------- Fonte: PolonesaReversa.java

package polonesa;

import java.util.Scanner;

public class PolonesaReversa {

// ........................................... lista de atributos da classe

    private double x, y, resp = 0;
    private char op;
    private int erro;

// ........................................... métodos públicos da classe

    public PolonesaReversa () {     // construtor
        erro = 0;
    }

    public void calculaPolonesaReversa() {
        Scanner entrada = new Scanner(System.in);
        String s;

        System.out.println("Polonesa Reversa: <enter> abandonar");
        do {
```

```
            if (resp == 0) {
                s = entrada.nextLine();
                if (s.equals("")) {
                    System.out.println("by Polonesa Reversa");
                    System.exit(0);
                }
                x = Double.parseDouble(s);
            }
            else {
                x = resp;
            }
            s = entrada.nextLine();
            if (s.equals("")) {
                System.out.println("by Polonesa Reversa");
                System.exit(0);
            }
            y = Double.parseDouble(s);
            do {
                s = entrada.nextLine();
                if (s.equals("")) {
                    System.out.println("by Polonesa Reversa");
                    System.exit(0);
                }
                op = s.charAt(0);
            } while (!strChr("+-*/Pp", op));
            operaCalculadora();
            exibeCalculadora();
        } while (!s.equals(""));
    }

    // ---------------------------- operaCalculadora

    private void operaCalculadora() {
        switch (op)
        {
            case '+':   resp = soma(x, y);
                        break;
            case '-':   resp = subtracao(x, y);
                        break;
            case '*':   resp = multiplicacao(x, y);
                        break;
            case '/':   if (y == 0)
                            erro = 1;   // divisão por zero
                        else
                            resp = divisao(x, y);
                        break;
            case 'P':
            case 'p':   resp = power(x, y);    // potência
                        break;
        }
    }

    // ---------------------------- exibeCalculadora

    private void exibeCalculadora() {
      switch (erro) {
           case 1: System.out.println("Erro: Divisão por Zero");
           break;
           case 2: System.out.println("Erro: Raiz Complexa");
           break;
           case 3: System.out.println("Erro: Tangente Inválida");
           break;
           default: System.out.println(resp);
          }
    }

// ..................................................... métodos privados da classe
```

```
    // ---------------------------- strChr

    private boolean strChr(String s, char ch) {

        for (int i = 0; i < s.length(); i++) {
            if (s.charAt(i) == ch) {
                return(true);
            }
        }
        return(false);
    }

    // ---------------------------- soma

    private double soma(double x, double y) {
        return(x + y);
    }

    // ---------------------------- subtração

    private double subtracao(double x, double y) {
        return(x - y);
    }

    // ---------------------------- multiplicação

    private double multiplicacao(double x, double y) {
        return(x * y);
    }

    // ---------------------------- divisão

    private double divisao(double x, double y) {
        if (y == 0) {
            erro = 1;
            return(-1);
        }
        return(x / y);
    }

    // ---------------------------- power

    private double power(double x, double y) {
        return(Math.pow(x, y));
    }
}
```