Aula 4

# Estrutura de Dados

Prof. Vinicius Pozzobon Borin

## Conversa Inicial

- O objetivo desta aula é apresentar os conceitos que envolvem a estrutura de dados do tipo árvore
- Será mostrado como realizar as manipulações dos dados dentro de uma estrutura de árvore binária, como a inserção, busca e visualização de dados

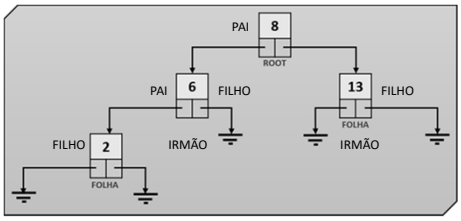
- Os tipos de listas e suas características serão:
  - Árvore binária
  - Árvore de *Adelson-Velsky e Landis* (árvore binária balanceada)

## Árvore Binária: Definições

### Definições

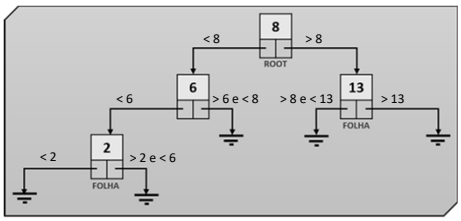
- Estrutura de dados não linear
- Organizada com elementos não necessariamente encadeados
- Forma ramificações e subdivisões na organização da estrutura de dados

## Árvore binária




- Número de filhos por nó: 0, 1 ou 2
- Inserção de valores de uma forma organizada:
  - Valores menores que o do nó pai ficam à esquerda
  - Valores maiores que o do nó pai ficam à direita

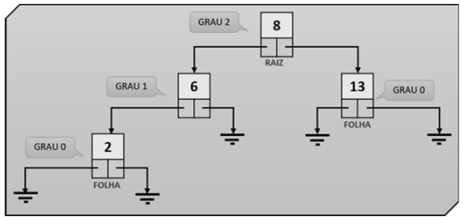
- Voltando ao nosso exemplo...



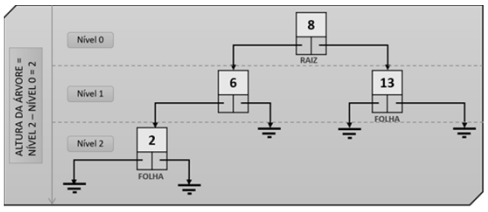

- Aplicações:
  - Árvore de busca binária
  - *Binary Search Tree* (BST)

## Grau de um nó da árvore binária

- Número de subárvores de cada nó




## Altura e nível de uma árvore



## Pseudocódigo: elemento da árvore

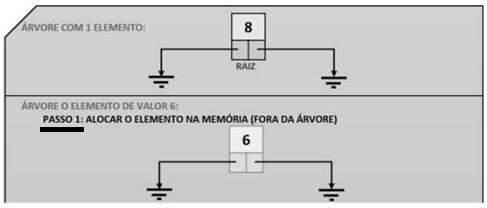
```
registro ElementoDaArvoreBinaria
    dado: inteiro
    esquerda: ElementoDaArvoreBinaria[->)
    direita: ElementoDaArvoreBinaria[->)
fimregistro
```

## Árvore Binária: Inserção de Dados

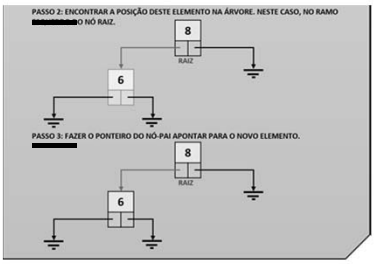
## Inserção na árvore

- Não existe inserção no início, meio ou fim como tínhamos nas listas
- A inserção sempre se dará em um dos nós folha da árvore, seguindo esta regra:
  - Dados de valores menores que seu antecessor são inseridos no ramo esquerdo da árvore
  - Dados de valores maiores que seu antecessor são inseridos no ramo direito da árvore

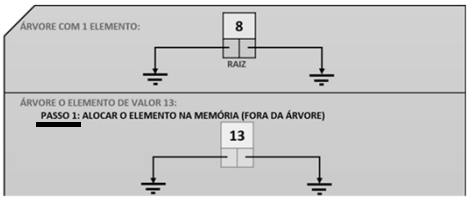
- Inserção à esquerda



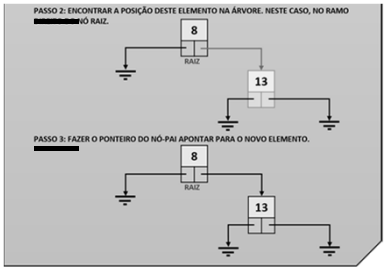

- Inserção à esquerda




- Inserção à direita



- **Inserção à direita**

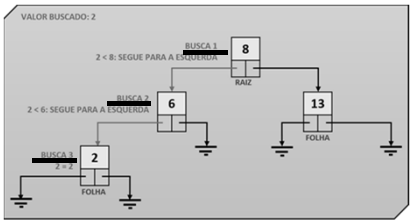



```
1  //Cria um Procedimento que recebe como parâmetro o dado a ser inserido na árvore
2  //E o endereço do nó atualmente a ser testado, iniciando pela raiz
3  função InserirNaArvore
4  (ElementoVarredura[->][->]: registro ElementoDaArvore, numero: inteiro): sem retorno
5  var
6     //Declara um novo nó do tipo REGISTRO
7     NovoElemento[->]: registro ElementoDaArvore                          PASSO 1
8  inicio
9     //Inicializa  novo nó fora da lista
10    NovoElemento[->] = NULO                                              PASSO 1
11    NovoElemento->dado = numero
12
13    //Verifica se o elemento recebido está vazio
14    se (ElementoVarredura[->] == NULO) então
15       //Se o elemento está vazio, coloca ele na árvore
16       NovoElemento->esquerda = NULO
17       NovoElemento->direita  = NULO
18       //Faz o elemento atual (vazio) da árvore receber o novo elemento      PASSO 2
19       ElementoVarredura[->] = NovoElemento[->]
20    senão
21       //Verifica-se RECURSIVAMENTE se a árvore deve
22       //seguir para o ramo esquerdo ou direito
23       se (numero < ElementoVarredura->dado) então
24          //Segue ara a esquerda recursivamente
25          InserirNaArvore( (->)ElementoVarredura->esquerda, numero)        PASSO 3
26       senão
27          //Segue ara a direita recursivamente
28          InserirNaArvore( (->)ElementoVarredura->direita, numero)         PASSO 3
29       fimse
30    fimse
31 fimfunção
```

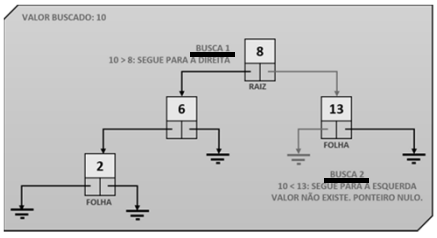
## Árvore Binária: Busca de Dados

## Busca na árvore

- **Busca com valor encontrado**




- **Busca com valor não encontrado**



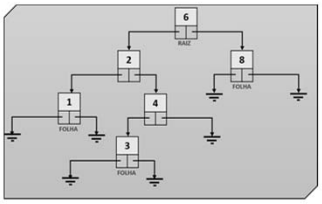

```
1  //Cria um Procedimento que recebe como parâmetro o dado a ser buscado na árvore
2  //E o endereço do nó atualmente a ser testado, iniciando pela raiz
3  função BuscarNaArvore
4  (ElementoVarredura[->][->]: registro ElementoDaArvore, numero: inteiro)
5  : registro ElementoDaArvore
6  var
7  inicio
8     //Verifica se o elemento recebido está vazio
9     se (ElementoVarredura[->] <> NULO) então
10       //Verifica-se RECURSIVAMENTE se a árvore deve
11       //seguir para o ramo esquerdo ou direito
12       se (numero < ElementoVarredura->dado) então
13          //Segue para a esquerda recursivamente
14          BuscarNaArvore( (->)ElementoVarredura->esquerda, numero)
15       senão
16          se (numero > ElementoVarredura->dado) então
17             //Segue para a direita recursivamente
18             BuscarNaArvore( (->)ElementoVarredura->direita, numero)
19             senão
20                se (numero == ElementoVarredura->dado) então
21                   //VALOR ENTRADO
22                   retorne ElementoVarredura[->]
23                fimse
24             fimse
25       fimse
26    senão
27       retorne NULO
28    fimse
29 fimfunção
```

## Árvore Binária: Visualização dos Dados

### Visualização na árvore

- Possibilidade de visualização:
  - Consultar em ordem: esquerda, raiz e direita. Dessa forma, os elementos listados ficarão apresentados em ordem crescente
  - Consultar em pré-ordem: raiz, esquerda e direita
  - Consultar em pós-ordem: esquerda, direita e raiz

- Consultar em ordem: 1, 2, 3, 4, 6, 8
- Consultar em pré-ordem: 6, 2, 1, 4, 3, 8
- Consultar em pós-ordem: 3, 4, 2, 8, 6




- Pseudocódigo da consulta em ordem:

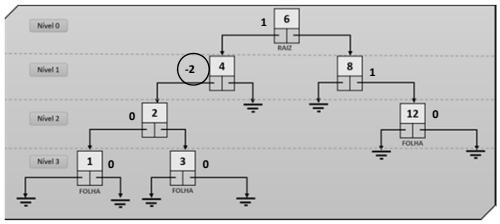
```
1   função VisualizarArvore_Ordem
2   (ElementoVarredura[->]: registro ElementoDaArvore): sem retorno
3   var
4   inicio
5       se (ElementoVarredura <> NULO) então
6           //Segue para a esquerda
7           VisualizarArvore_Ordem (ElementoVarredura->esquerda)
8           //Imprime o elemento atual
9           escreva(ElementoVarredura->dado)
10          //Segue para a direita
11          VisualizarArvore_Ordem (ElementoVarredura->direita)
12      fimse
13  fimfunção
```

## Árvore de *Adelson-Velsky e Landis* (AVL)

### Árvore AVL

- Árvore binária balanceada
- Evita ramos muito longos que podemos ter em árvores não balanceadas
- Melhora o tempo de acesso aos dados
- Mantém todas as características de uma árvore binária, com uma propriedade a mais:
  - A diferença de altura entre uma subárvore direita e uma subárvore esquerda sempre deverá ser 1, 0 ou -1

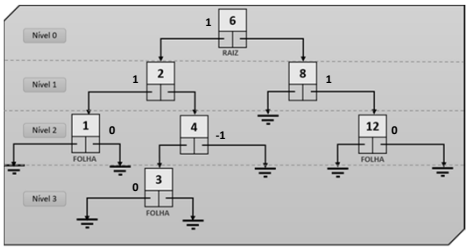
## Árvore binária não balanceada




## Rotação da árvore binária

| Diferença de altura de um nó | Diferença de altura do nó filho do nó desbalanceado | Tipo de rotação |
|---|---|---|
| 2 | 1 | Simples à esquerda |
| | 0 | Simples à esquerda |
| | -1 | Dupla com filho para a direita e pai para a esquerda |
| -2 | 1 | Dupla com filho para a esquerda e pai para a direita |
| | 0 | Simples à direita |
| | -1 | Simples à direita |

## Árvore binária balanceada




## Referências


- ASCENCIO, A. F. G. Estrutura de dados: algoritmos, análise da complexidade e implementações em JAVA e C/C++. São Paulo: Pearson, 2011.
- ASCENCIO, A. F. G. Fundamentos da programação de computadores: algoritmos, Pascal, C/C++ (padrão ANSI) JAVA. 3. ed. São Paulo: Pearson, 2012.
- CORMEN, T. H. Algoritmos. Teoria e prática. 3. ed. Rio de Janeiro: Elsevier, 2012.
- LAUREANO, M. Estrutura de dados com algoritmos e C. Rio de Janeiro: Brasport, 2008.
- MIZRAHI, V. V. Treinamento em linguagem C. 2. ed. São Paulo: Peason, 2008.
- PUGA, S.; RISSETI, G. Lógica de programação e estrutura de dados. 3. ed. São Paulo: Pearson, 2016.