

Unidade 12 – Árvores Binárias





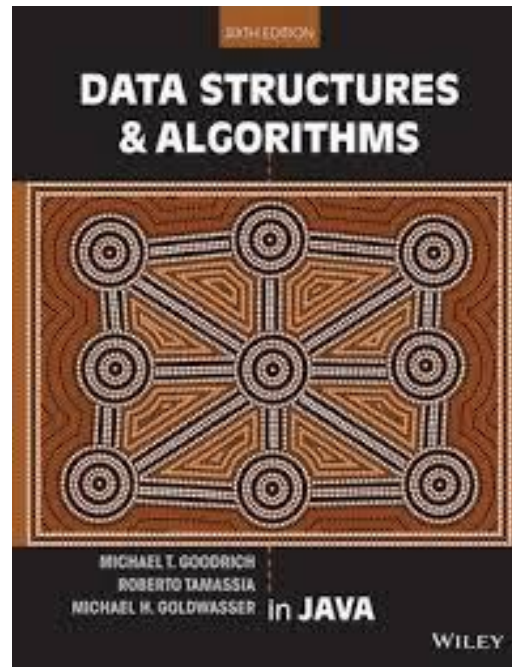
Bibliografia

- Data Structures and Algorithms in Java – Fourth Edition – Roberto Tamassia – Michael T. Goodrich – John Wiley & Sons, Inc
- Head First Java, 2nd Edition by Kathy Sierra and Bert Bates
- Estrutura de Dados e Algoritmos – Bruno R. Preiss, Editora Campus, 2001
- Estrutura de Dados e Algoritmos em Java – Robert Lafore, Editora Ciência Moderna, 2005
- Algoritmos e Estrutura de Dados – Niklaus Wirth – Editora Prentice Hall do Brasil, 1989
- Estrutura de Dados e Algoritmos em C++, Adam Drozdek – Thompson
- Introdução à Estrutura de Dados, Celes, Cerqueira, Rangel – Elsevier
- FREITAS, Aparecido V. de – 2022 – Estruturas de Dados: Notas de Aula.
- CALVETTI, Robson - 2015 – Estruturas de Dados: Notas de Aula.



Leitura Recomendada para a Unidade 5

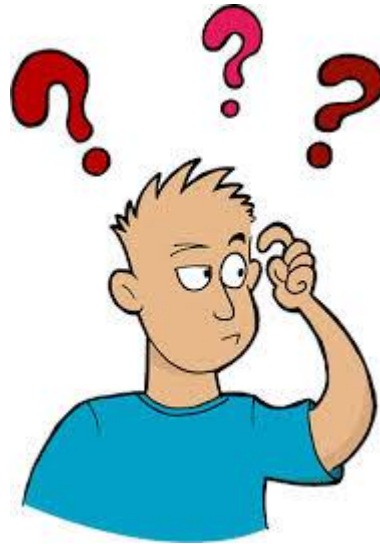
- ⊕ Data Structures and Algorithms in Java (*), Roberto Tamassia and Michael T. Goodrich, Sixty Edition – **2014**, Seção **8.2**



(*) Em português, Estrutura de Dados e Algoritmos em Java

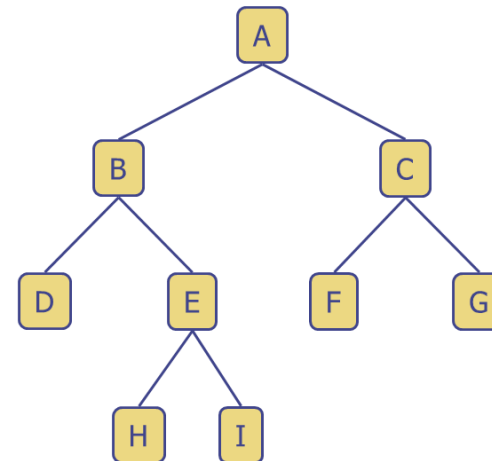


O que é árvore binária ?



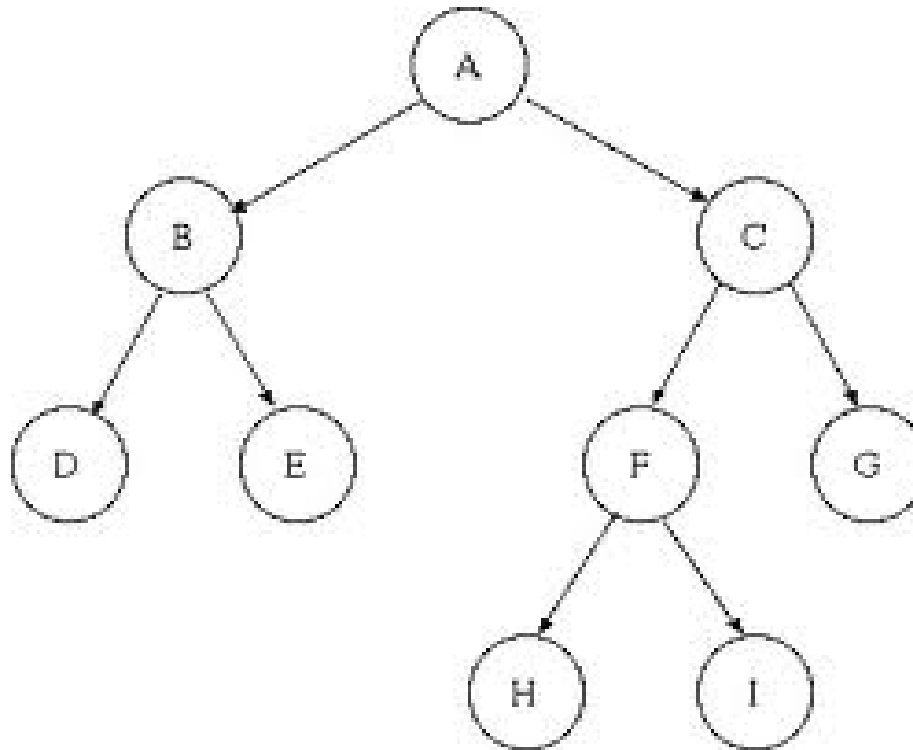
Árvore Binária

- É uma árvore ordenada com as seguintes propriedades:
 - ◆ Todo nó tem no máximo 2 filhos.
 - ◆ Cada filho é rotulado como sendo filho a esquerda ou filho a direita.
 - ◆ Um filho a esquerda precede o filho a direita na ordenação dos filhos de um nó.
 - ◆ Assim, filhos formam um par ordenado.



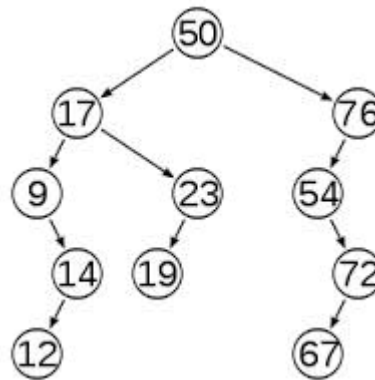
Árvore Binária própria

- ◆ Uma árvore binária é **própria** se cada nó tem 0 ou 2 filhos.
- ◆ Em uma árvore binária **própria** cada nó interno tem exatamente 2 filhos.



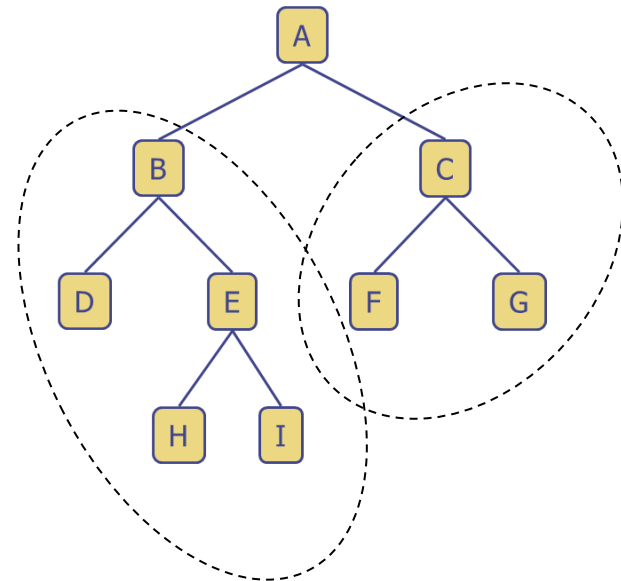
Árvore Binária Imprópria

- Uma árvore é **imprópria** se não for própria, ou seja, a árvore tem pelo menos um nó com apenas um filho.



Definição Recursiva

- Uma árvore binária é:
 - ◆ Uma árvore que consiste de apenas um nó, ou
 - ◆ Uma árvore cuja raiz tem um par ordenado de filhos, onde cada qual é uma árvore binária.

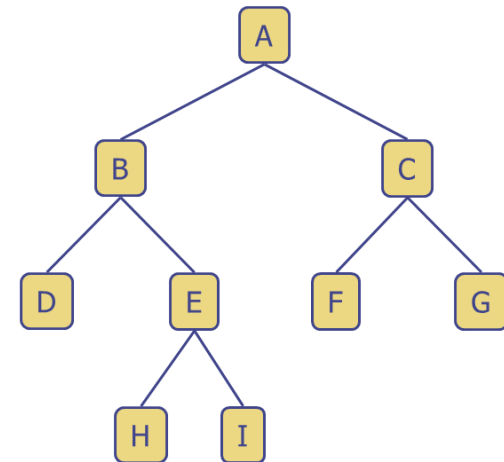


Quais as aplicações de árvores binárias ?



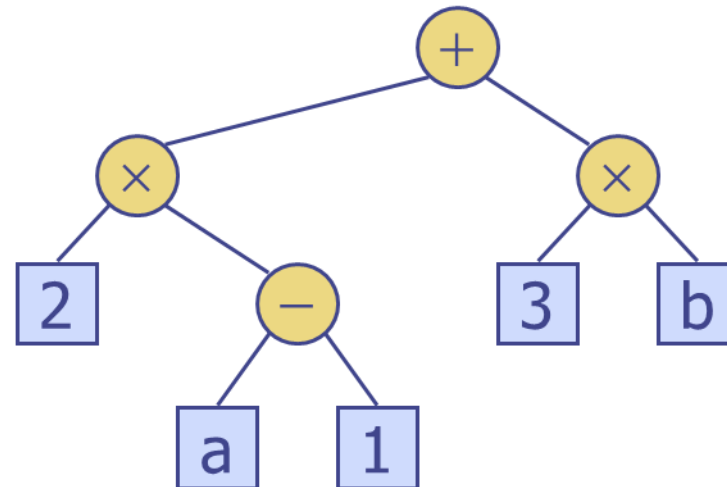
Árvores Binárias – Aplicações

- ⌚ Expressões aritméticas
- ⌚ Processos de decisão
- ⌚ Searching



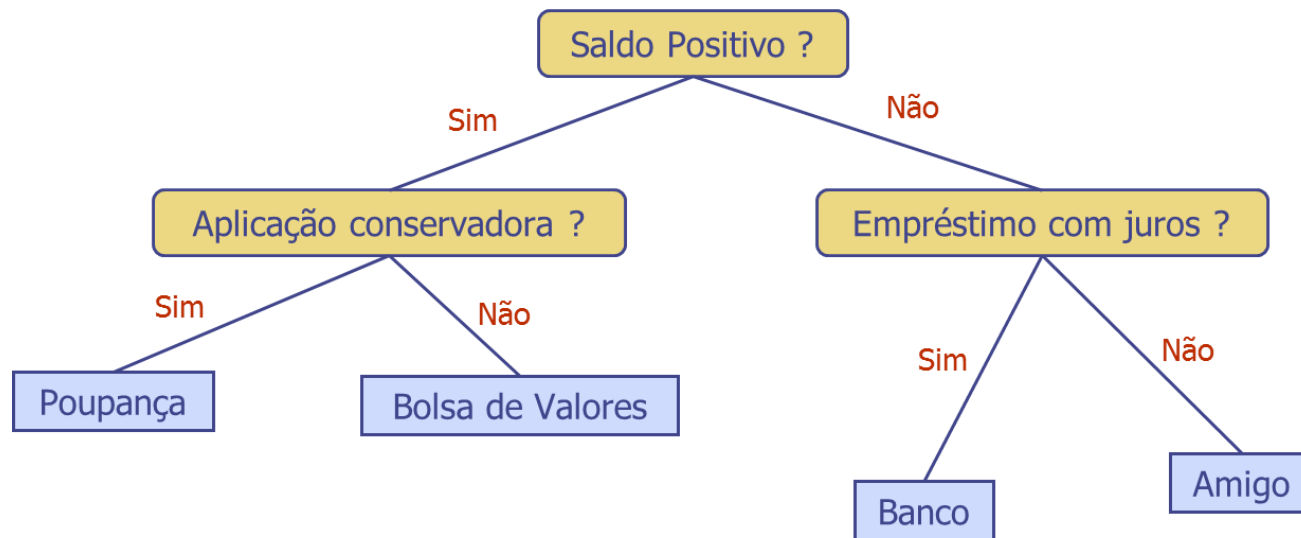
Árvore aritmética de expressões

- ⊕ Árvore binária associada à uma expressão aritmética.
- ⊕ Nós internos contém operadores (+, -, *, /, log, etc...)
- ⊕ Nós externos contém operandos (variáveis ou constantes)
- ⊕ Exemplo: $(2 \times (a - 1) + (3 \times b))$



Árvore de Decisão

- ⊕ Árvore binária associada a um processo de decisão.
- ⊕ Nós internos: questões com resposta sim/não
- ⊕ Nós externos: decisões



ADT – Árvore Binária

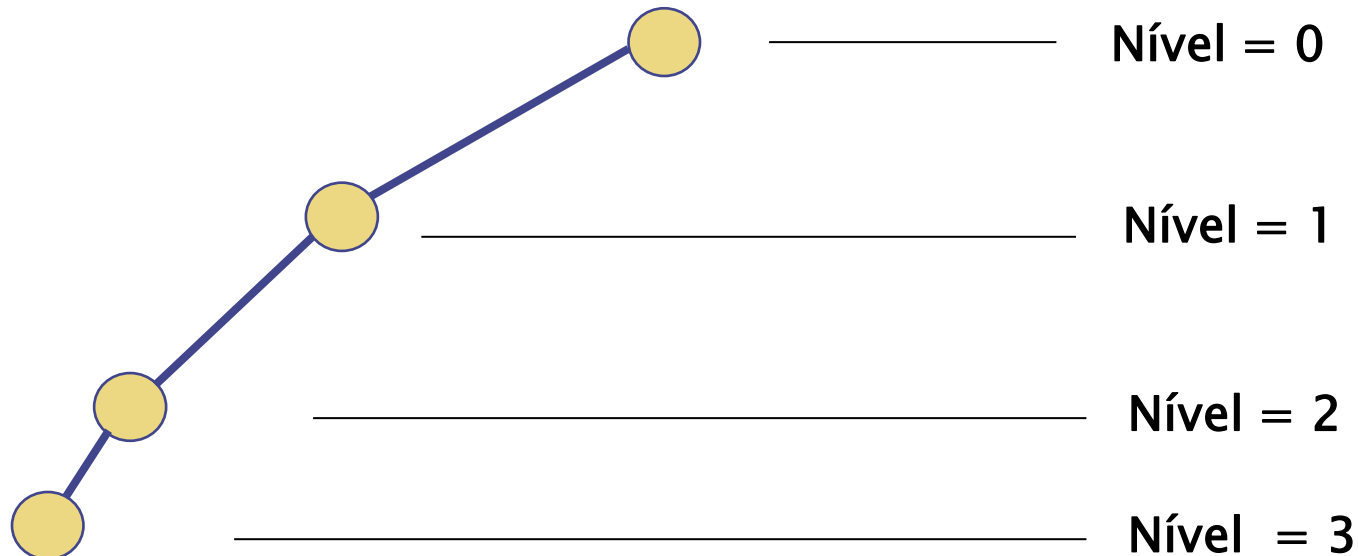
- ⊕ A árvore binária estende a ADT Árvore, isto é, herda todos os métodos vistos no capítulo anterior (árvores genéricas).
- ⊕ Adicionalmente, suporta os seguintes métodos:

left(): retorna o filho esquerdo de um nó
right(): retorna o filho direito de um nó
hasLeft(): testa se o nó tem filho a esquerda
hasRight(): testa se o nó tem filho a direita
inorder(): percurso inorder



Número mínimo de nós

- ⊕ O número mínimo de nós em uma árvore binária de altura **h** , é **$n \geq h+1$** .
- ⊕ Ao menos um nó em cada um dos níveis d .

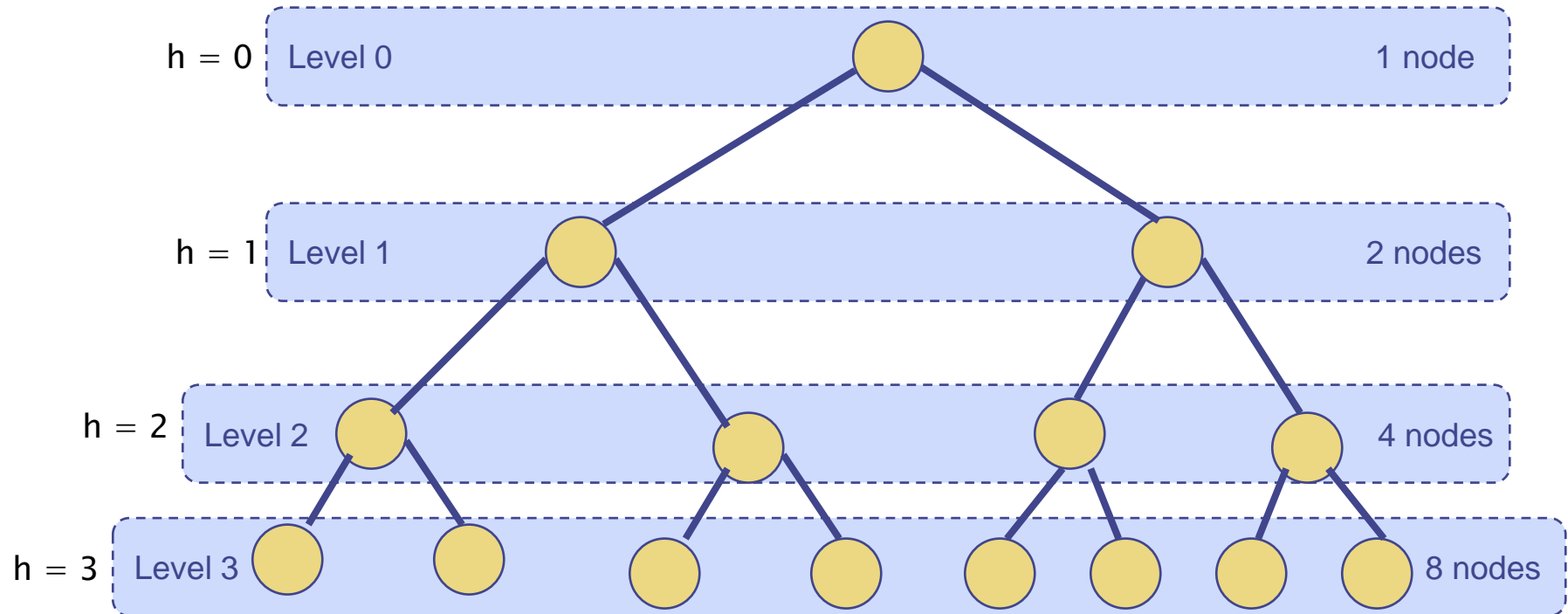


Número mínimo de nós é **$h+1$**

- ⊕ **Altura de um nó:** Tamanho do caminho de **n** até seu mais profundo descendente.



Máximo número de nós



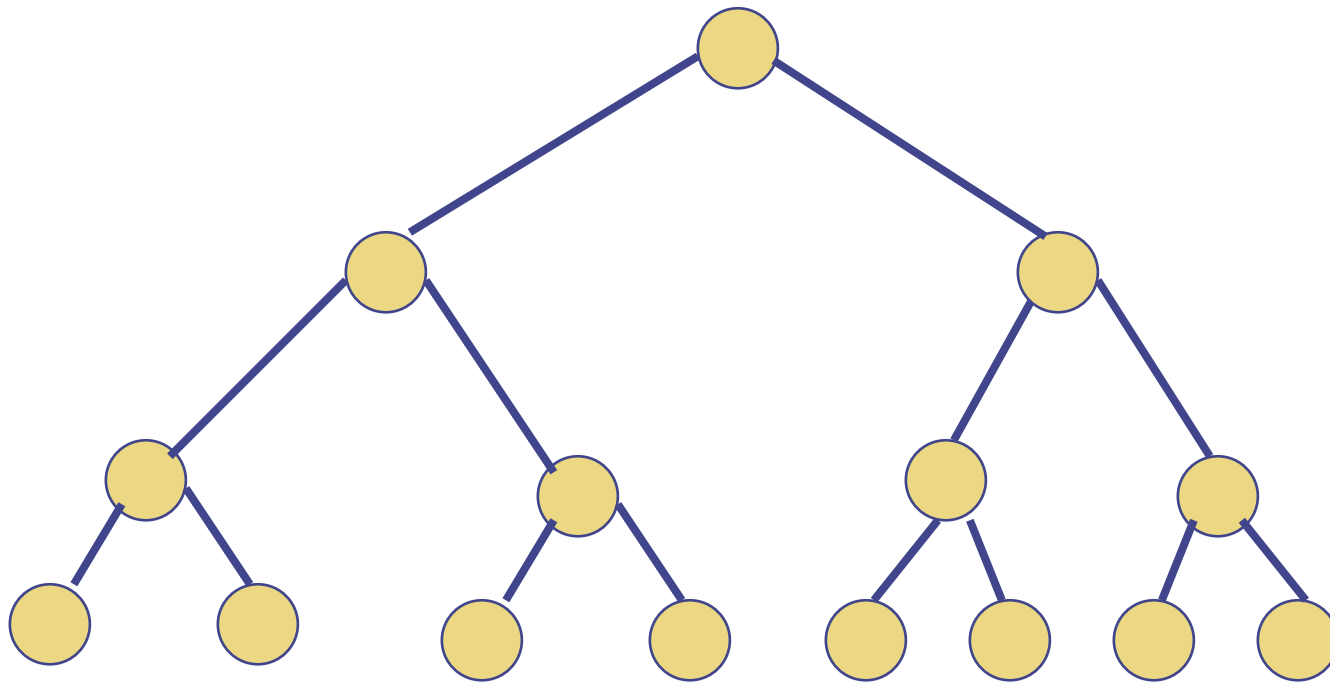
$$\text{Máximo número de nós} = 1 + 2 + 4 + 8 + \dots + 2^h$$

$$n = 2^{h+1} - 1$$



Árvore Binária Completa (Full)

Uma árvore binária completa de altura h tem $2^{h+1} - 1$ nós.



Árvore binária completa de altura 3



Representação de árvores binárias

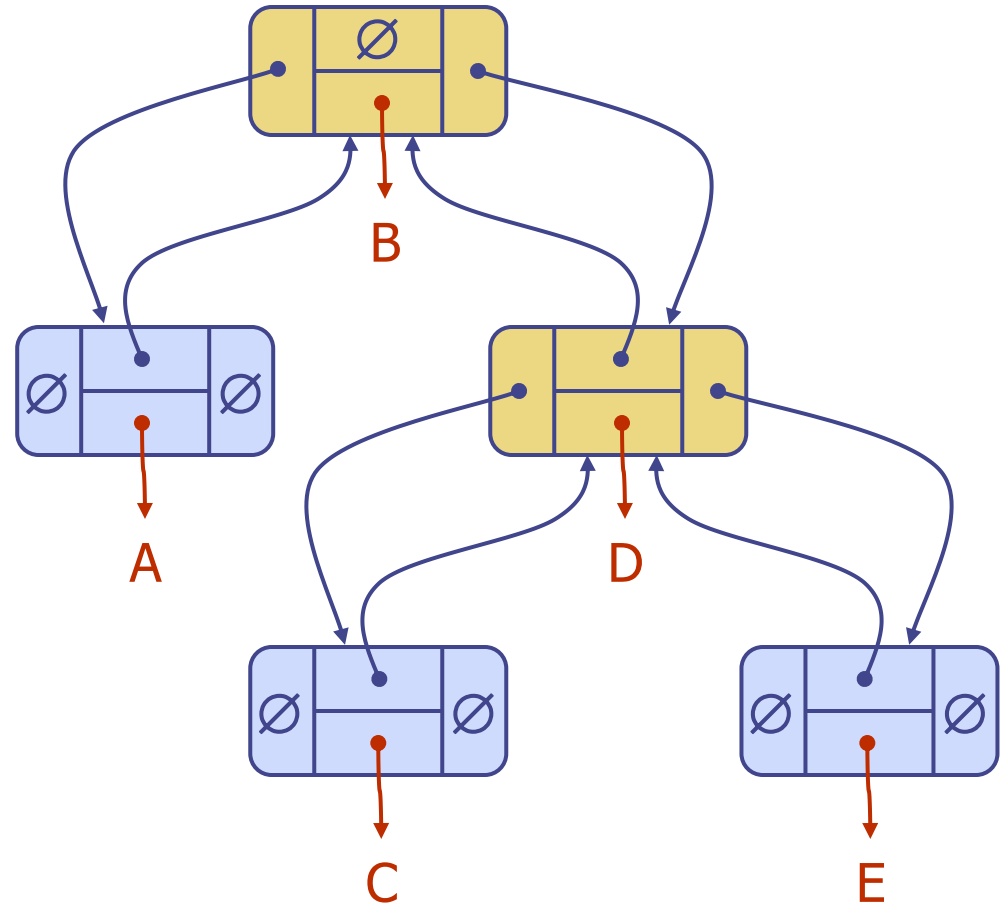
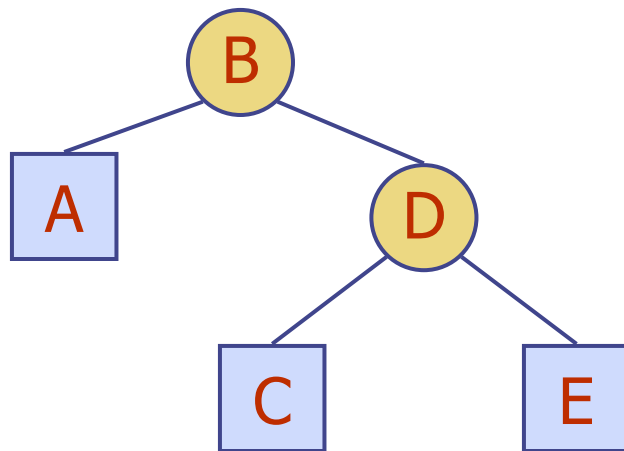
- 1. Linked Structure**
- 2. Array List**



Representação por lista ligada

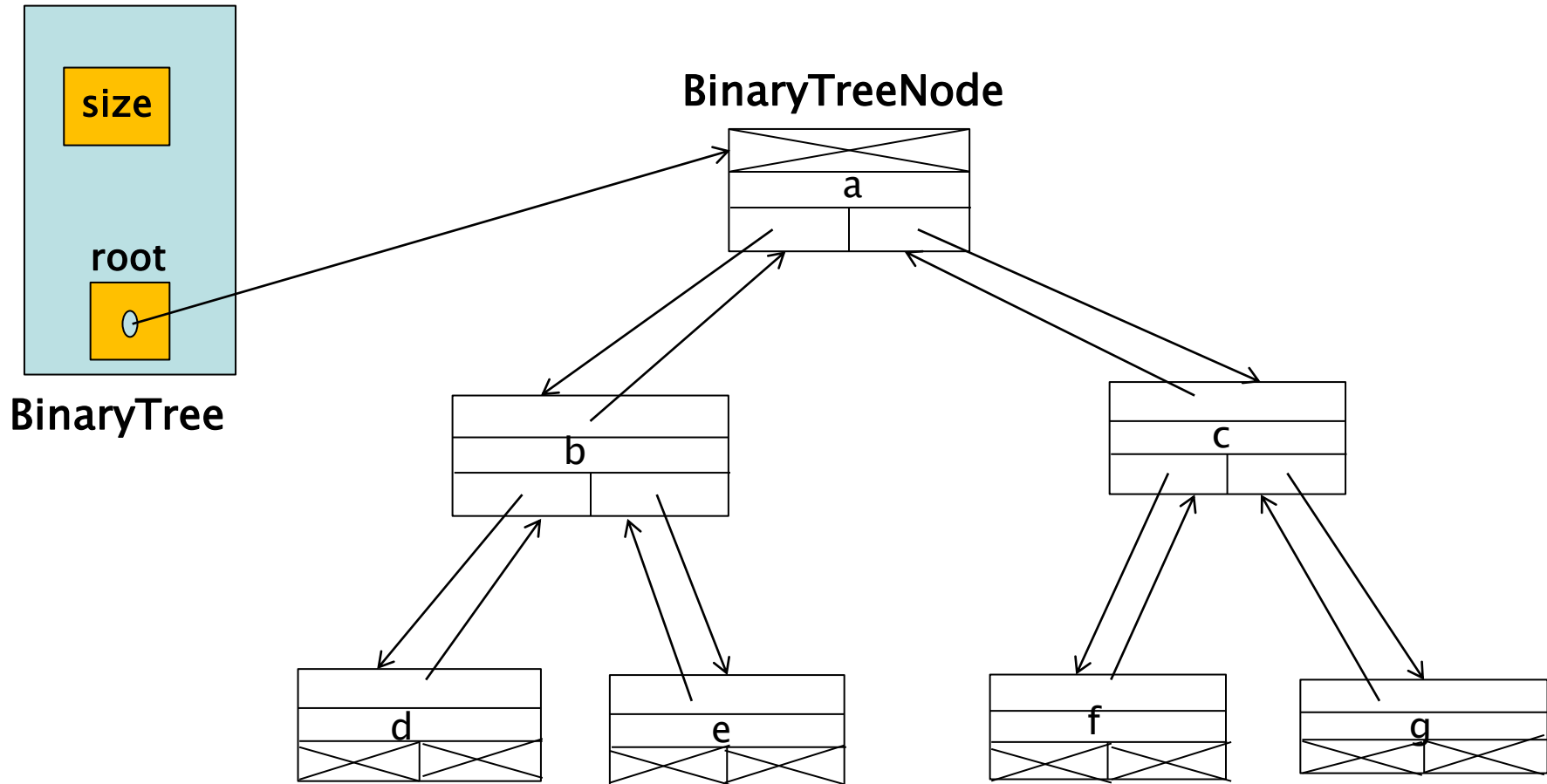
◆ Um nó é representado por um objeto armazenando:

- Elemento
- Nó pai
- Nó Left child
- Nó Right child



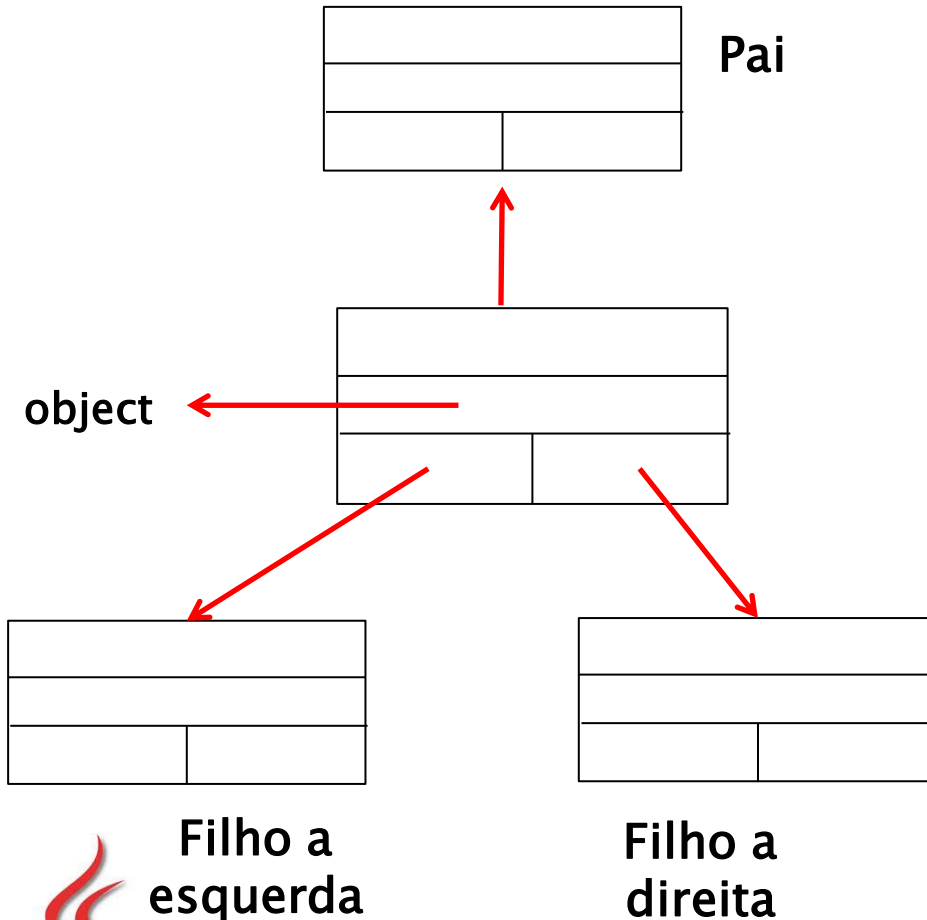
Representação por lista ligada

size -> #nós da árvore



Representando nó da Árvore

- Cada nó tem quatro referências: item, pai, filho a esquerda e filho a direita.



```
Class BinaryTreeNode {
```

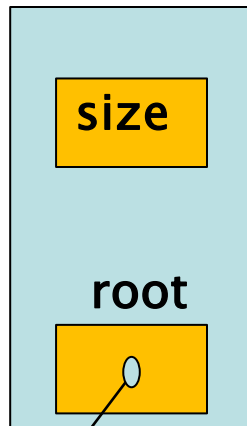
```
    Object item;  
    BinaryTreeNode parent;  
    BinaryTreeNode left;  
    BinaryTreeNode right;
```

```
}
```



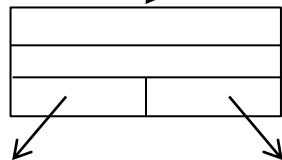
Representando a Árvore

Size -> #nós da árvore



BinaryTree

No_root



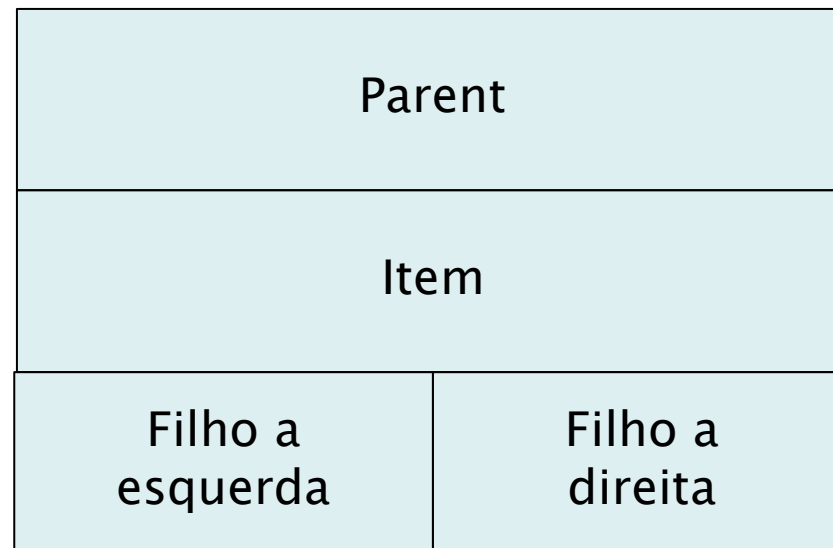
```
Class BinaryTree {
```

```
    BinaryTreeNode root;  
    int size;
```

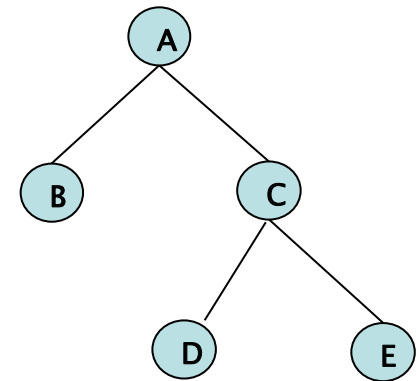
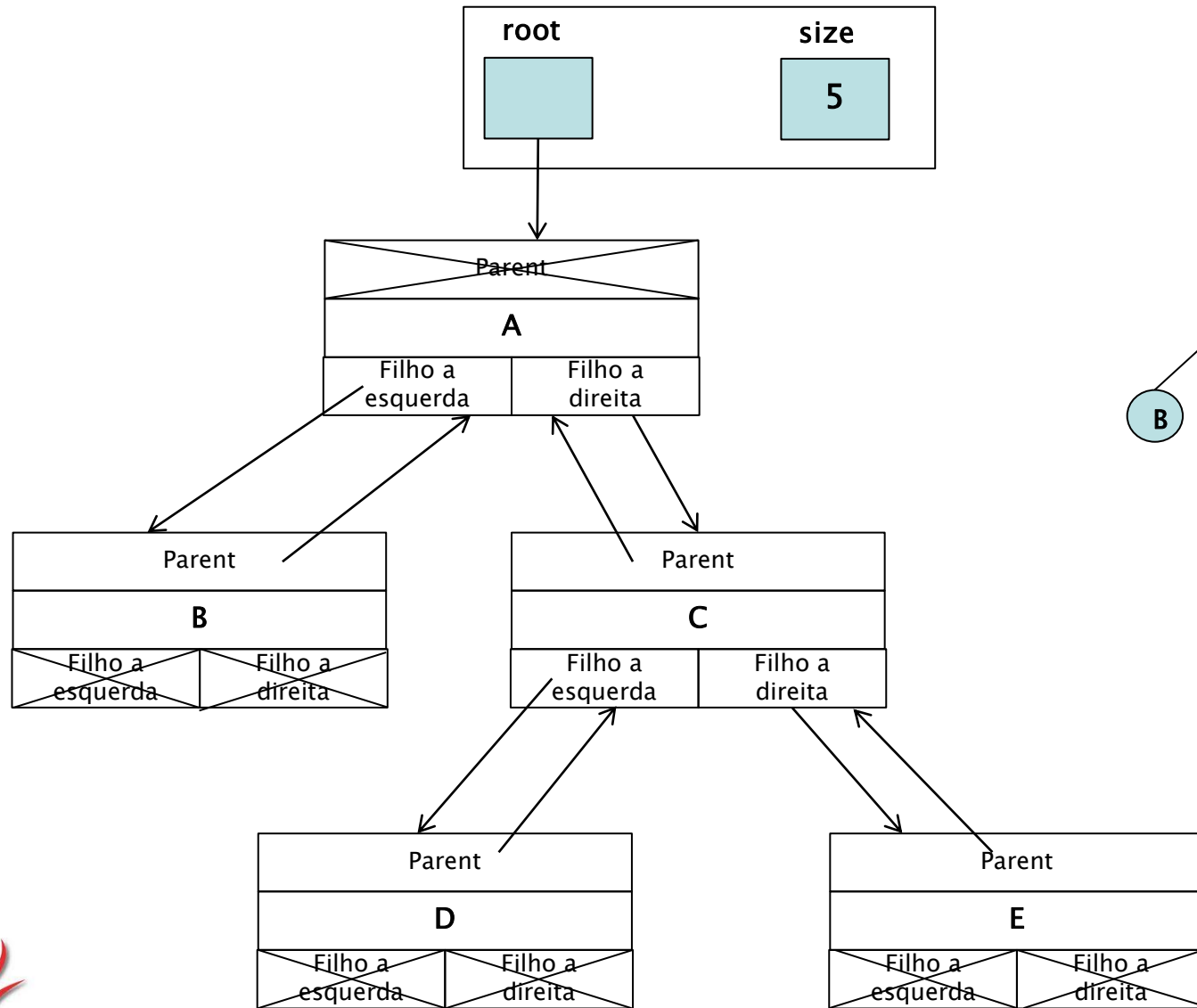
```
}
```



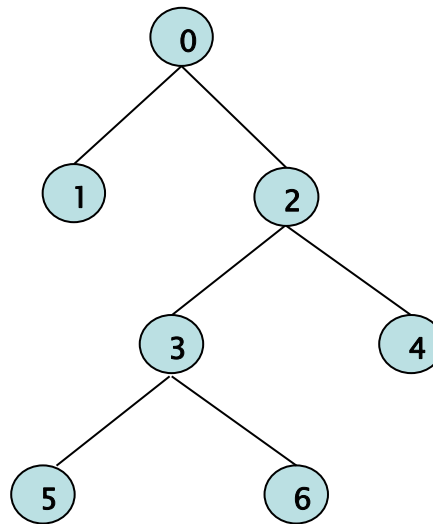
Representando Nó da árvore



Exemplo



Exemplo




```
public class BinaryTree {  
  
    BinaryTreeNode root;  
    int size;  
  
    public BinaryTree() {  
  
        this.root = null;  
        this.size = 0;  
    }  
  
    public void insert_root(int valor) {  
  
        BinaryTreeNode node = new BinaryTreeNode(valor);  
        this.root = node;  
        this.size = 1;  
    }  
}
```



```
public BinaryTreeNode ret_Root() {  
    return (this.root);  
}  
  
public int size() {  
    return this.size;  
}  
  
public boolean isEmpty() {  
    if (this.size == 0 )  
        return true;  
    else return false;  
}  
}
```



```
package uscs;

public class BinaryTreeNode {

    int item;
    BinaryTreeNode parent;
    BinaryTreeNode left;
    BinaryTreeNode right;

    public BinaryTreeNode(int item) {

        this.item = item;
        this.parent = null;
        this.left = null;
        this.right = null;

    }
```



```
public BinaryTreeNode left() {  
    if (this.left == null)  
        return null;  
    else return this.left ;  
  
}  
  
public boolean isLeft() {  
    if (this.left == null)  
        return false;  
    else return true ;  
  
}  
  
public BinaryTreeNode right() {  
    if (this.right == null)  
        return null;  
    else return this.right ;  
  
}
```



```
public boolean isRight() {  
    if (this.right == null)  
        return false;  
    else return true ;  
  
}  
  
public void binaryPreorder() {  
  
    System.out.println(this.item);  
    if (this.isLeft())  
        this.left.binaryPreorder();  
    if (this.isRight())  
        this.right.binaryPreorder();  
  
}
```



```
public void binaryPostorder() {
```

```
    if (this.isLeft())  
        this.left.binaryPostorder();  
    if (this.isRight())  
        this.right.binaryPostorder();  
    System.out.println(this.item);
```

```
}
```

```
public void binaryInorder() {
```

```
    if (this.isLeft())  
        this.left.binaryInorder();  
    System.out.println(this.item);  
    if (this.isRight())  
        this.right.binaryInorder();
```

```
}
```

```
}
```



package uscs;

public class Teste_BinaryTreeNode {

public static void main(String[] args) {

BinaryTree x = **new** BinaryTree();
x.insert_root(0);

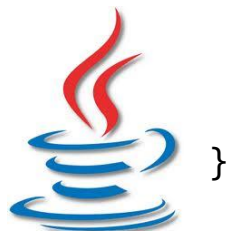
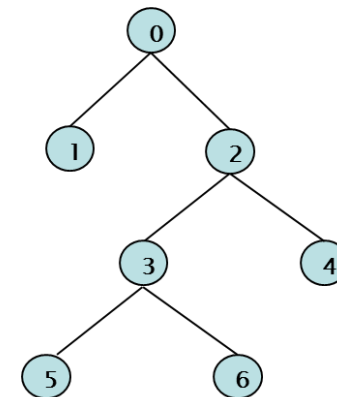
BinaryTreeNode no_1 = **new** BinaryTreeNode(1) ;
BinaryTreeNode no_2 = **new** BinaryTreeNode(2) ;
BinaryTreeNode no_3 = **new** BinaryTreeNode(3) ;
BinaryTreeNode no_4 = **new** BinaryTreeNode(4) ;
BinaryTreeNode no_5 = **new** BinaryTreeNode(5) ;
BinaryTreeNode no_6 = **new** BinaryTreeNode(6) ;

x.root.left = no_1;
x.root.right = no_2;
no_2.left = no_3;
no_2.right = no_4;
no_3.left = no_5;
no_3.right = no_6;

x.root.binaryPreorder();
x.root.binaryPostorder();
x.root.binaryInorder();

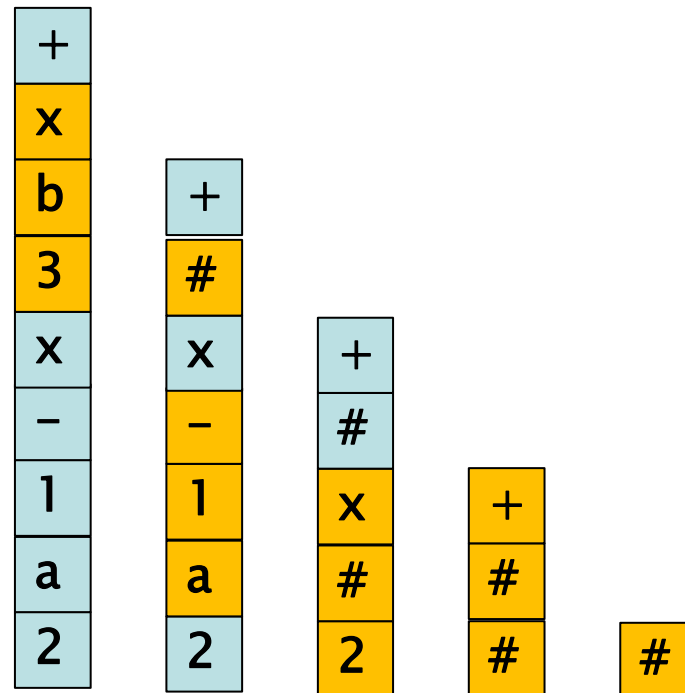
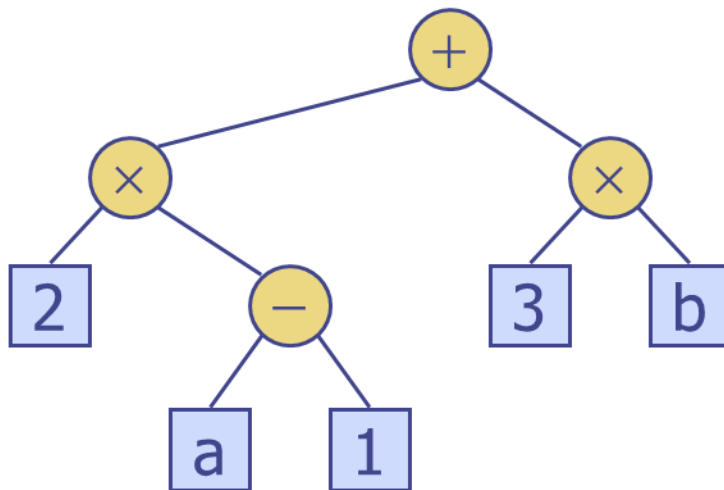
}

}



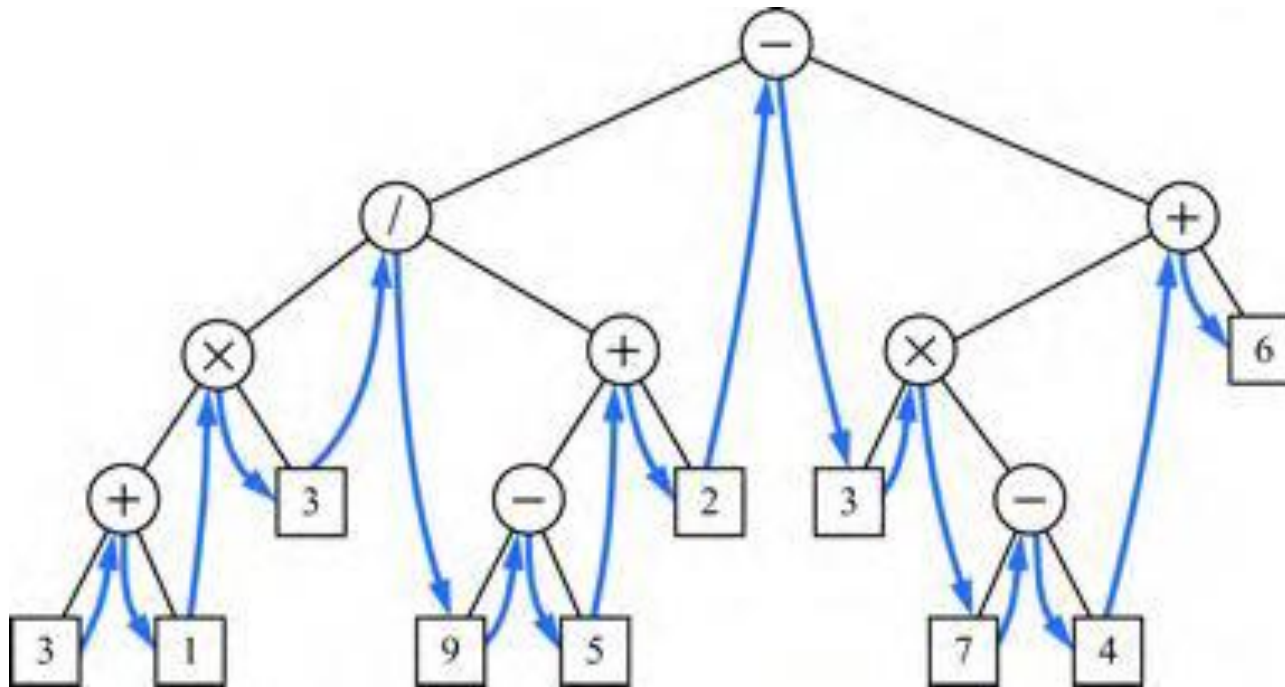
Árvore aritmética de expressões

- ⊕ O método de travessia postorder pode ser usado para resolver o problema da avaliação de uma árvore de expressões.
- ⊕ A travessia **postorder** da árvore abaixo é dada por: 2 a 1 – x 3 b x +



Travessia Inorder

- ⊕ Representa um método de travessia adicional válido para árvores binárias.
- ⊕ Nesta travessia, visitamos um nó entre as chamadas recursivas das subárvores esquerda e direita.



FIM

