

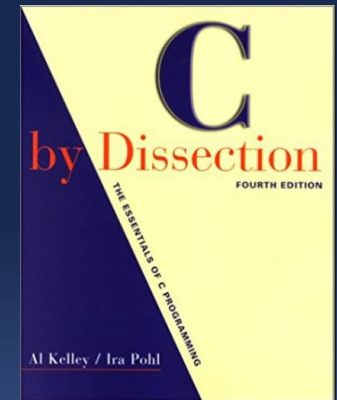
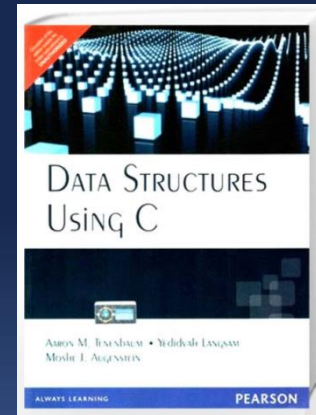
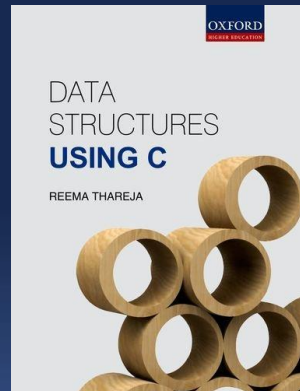
Unidade 11 – Árvores Binárias



Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP
aparecidovfreitas@gmail.com

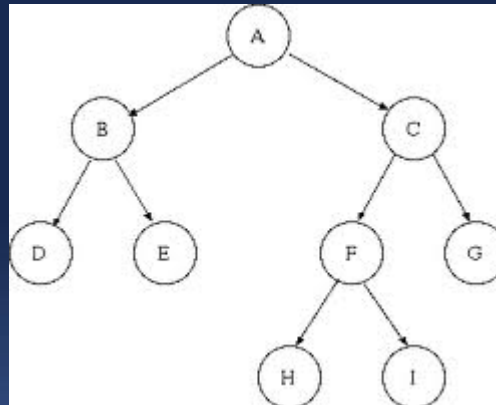
Bibliografia

- Algoritmos – Teoria e Prática – Cormen – Segunda Edição – Editora Campus, 2002
- Data Structures using C – Oxford University Press – 2014
- Data Structures Using C – A. Tenenbaum, M. Augensem, Y. Langsam, Pearson 1995
- C By Dissection – Kelley, Pohh – Third Edition – Addison Wesley



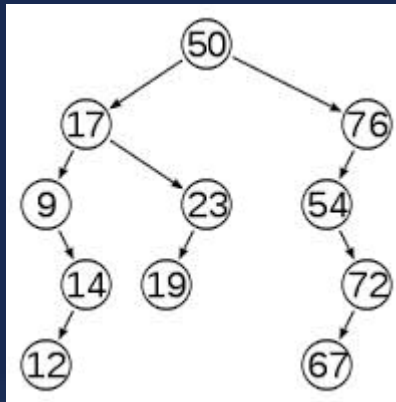
Árvore Binária própria

- ❖ 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.



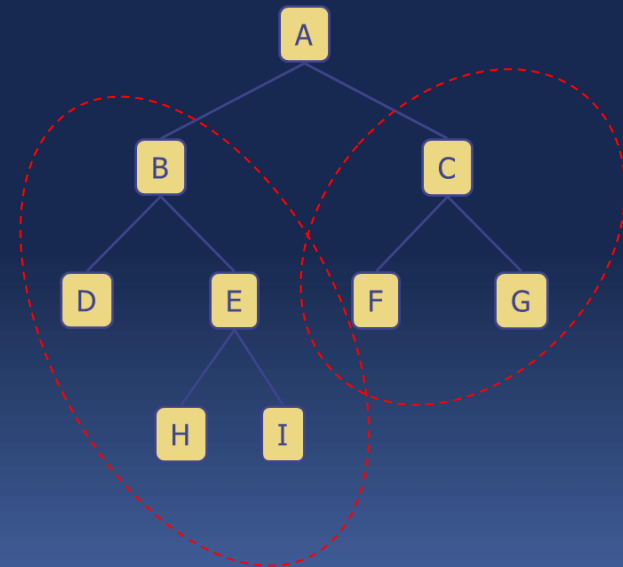
Árvore Binária Imprópria

- 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.



ADT – Árvore Binária

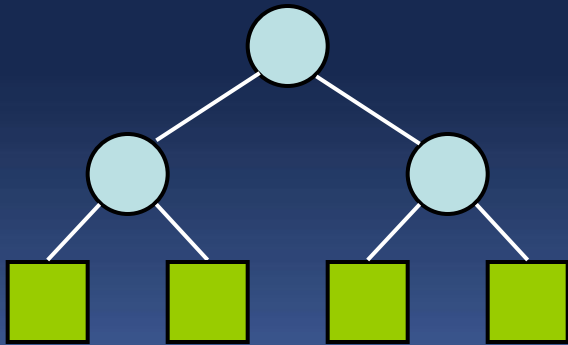
- ❖ A árvore binária estende a ADT Árvore, isto é, herda todos os métodos vistos no capítulo anterior (árvores genéricas).
- ❖ Adicionalmente, suporta os seguintes métodos:

`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

Árvore Binária Própria – Propriedades

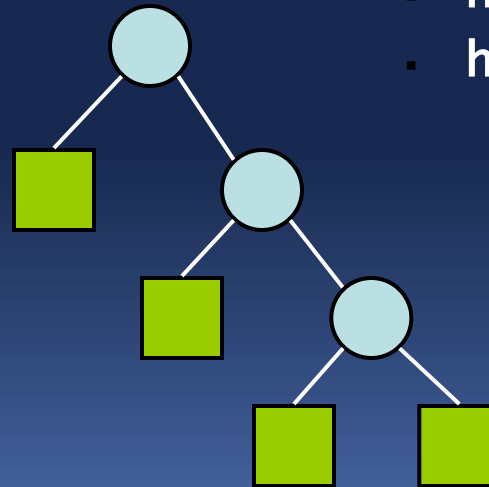
❖ Notação

- n número de nós
- e número de nós externos
- i número de nós internos
- h altura
- b número de arestas



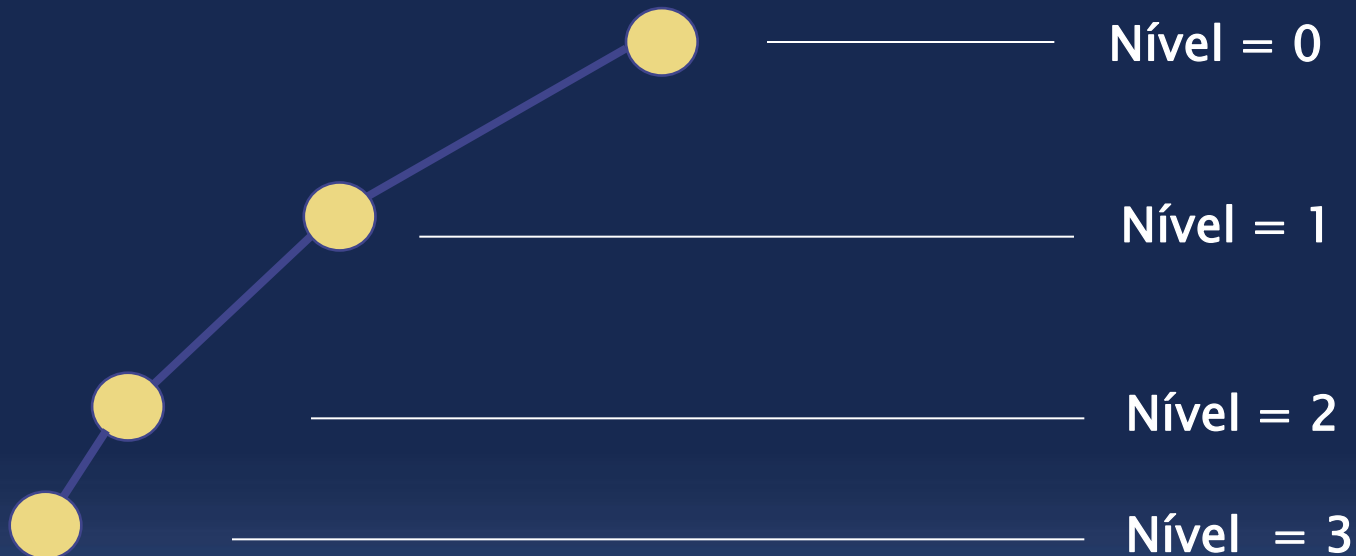
❖ Propriedades

- $e = i + 1$
- $n = 2e - 1$
- $h \leq i$
- $h \leq (n - 1)/2$
- $e \leq 2^h$
- $h \geq \log_2 e$
- $h \geq \log_2 (n + 1) - 1$



Número mínimo de nós

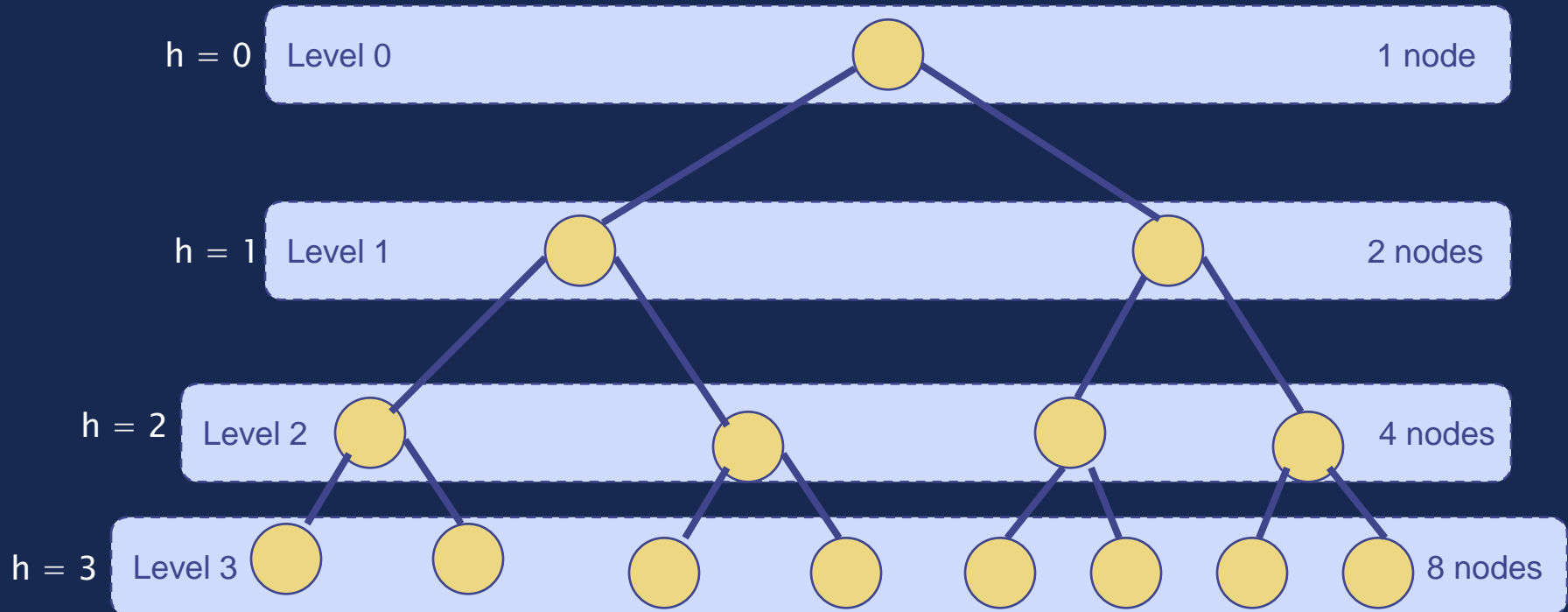
- ❖ O número mínimo de nós em uma árvore binária de altura h , é $n \geq h+1$.
- ❖ Ao menos um nó em cada um dos níveis d .



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

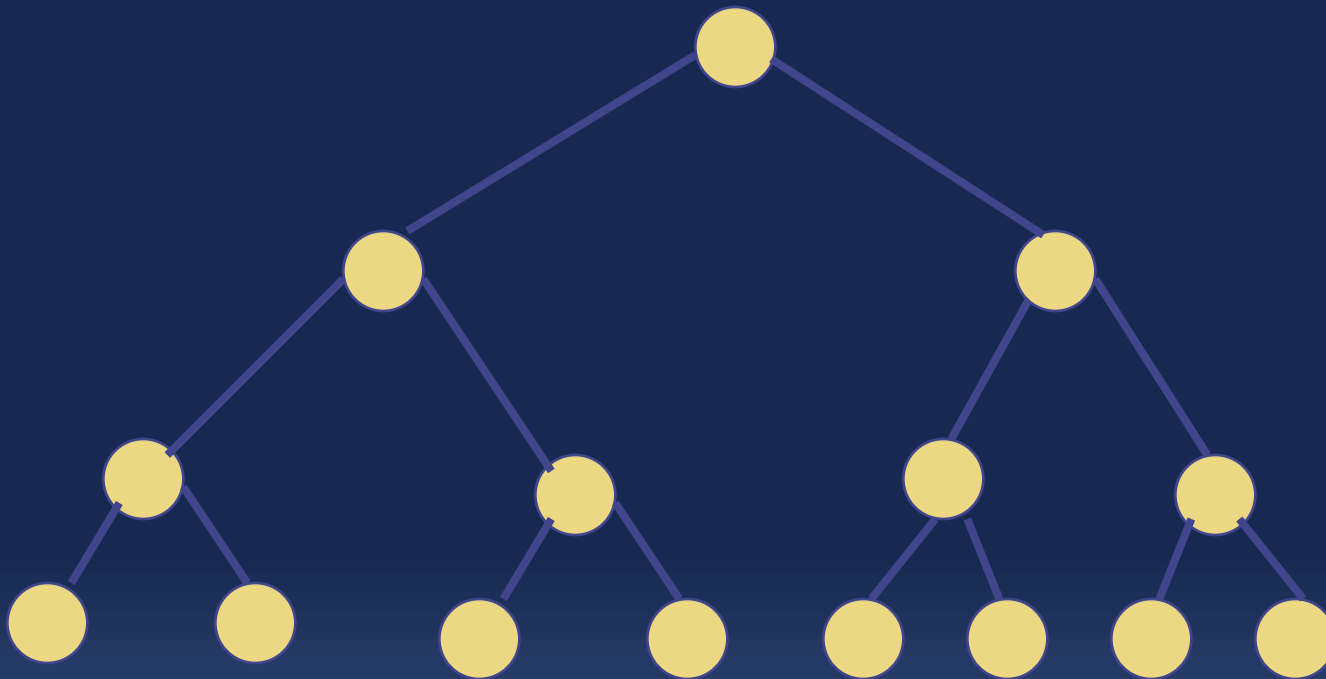


$$\text{Máximo número de nós} = 1 + 2 + 4 + 8 + \dots + 2^h$$

$$n \leq 2^{h+1} - 1$$

Árvore Binária Completa (Full)

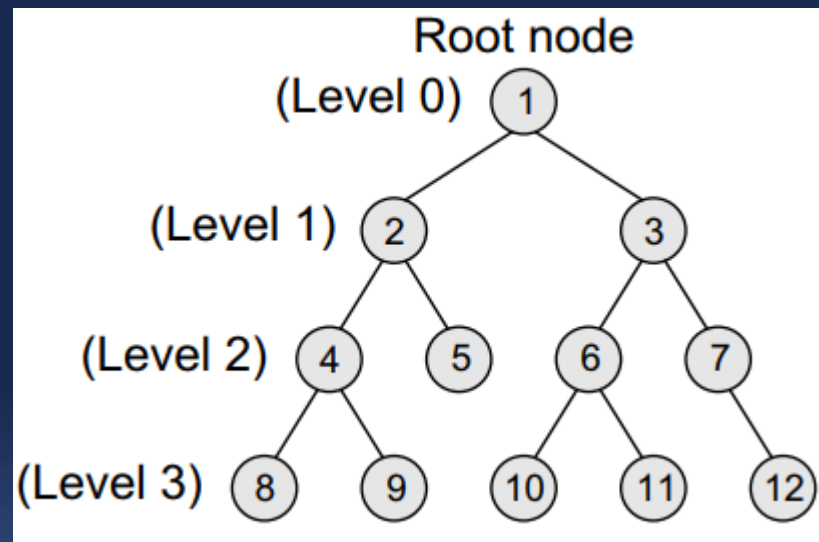
Uma árvore binária completa de altura h tem $2^{h+1} - 1$ nós.



Árvore binária completa de altura 3

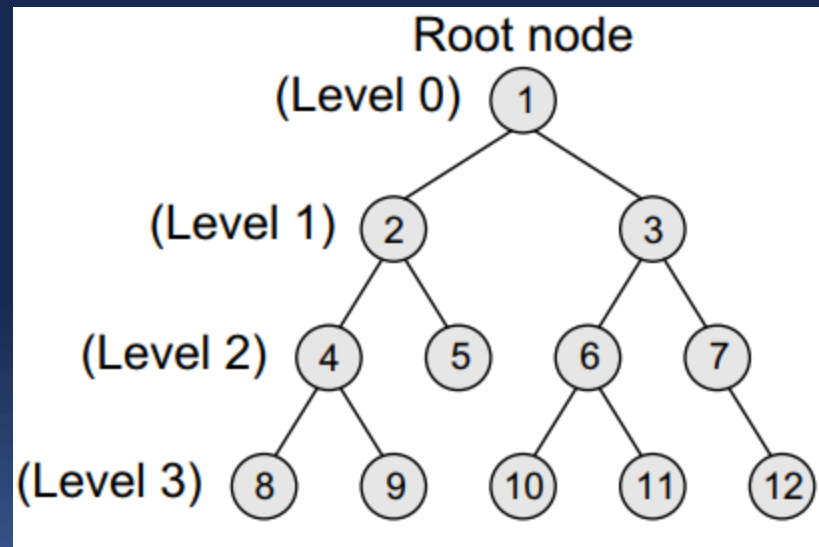
Nível de um nó

- ✓ Cada nó em uma árvore binária é associado a um número de nível;
- ✓ O nó root tem nível zero.



Grau de um nó

- ✓ Corresponde ao número de filhos que um determinado nó possui.
- ✓ O grau de um nó folha é zero;
- ✓ Por exemplo, o grau do nó 4 é 2, o grau do nó 5 é zero e o grau do nó 7 é 1.



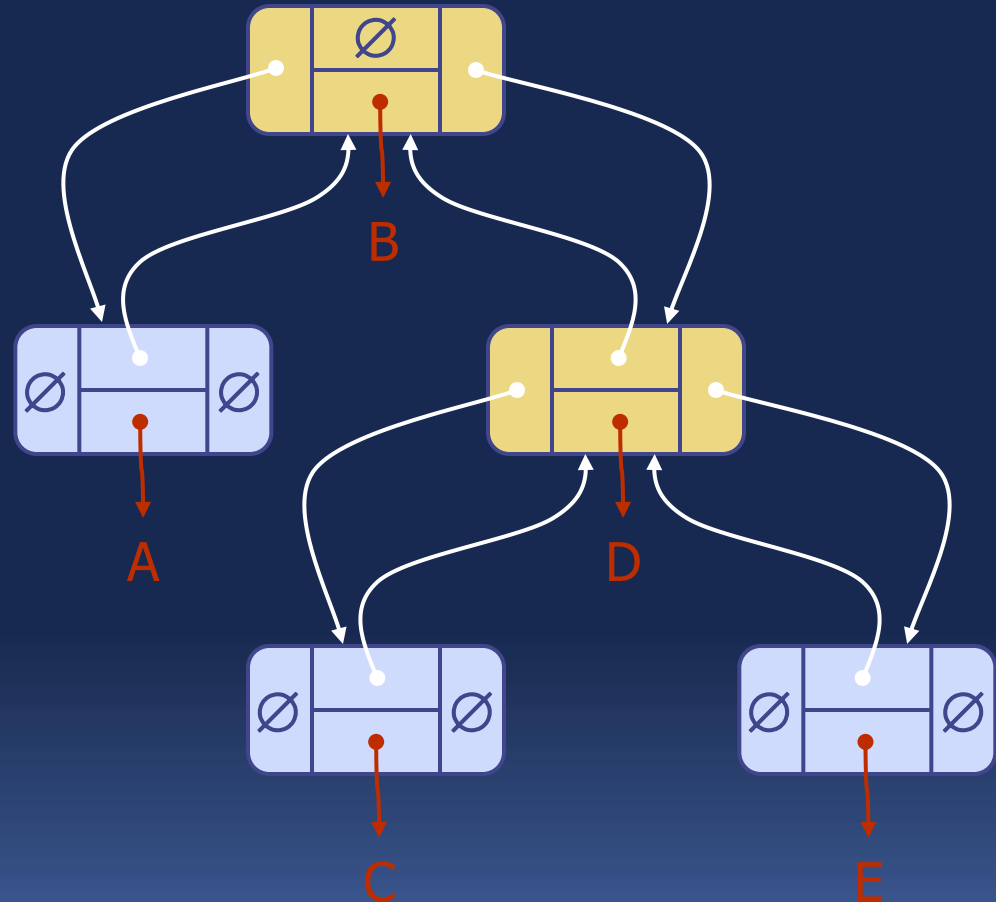
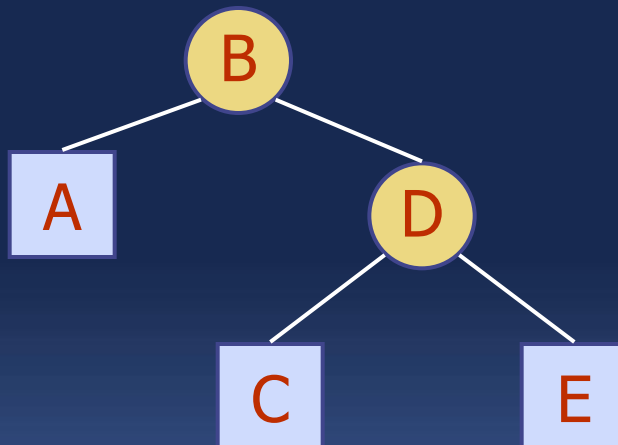
Representação de árvores binárias em Memória

1. **Linked Structure**
2. **Representação sequencial**

Representação por lista ligada

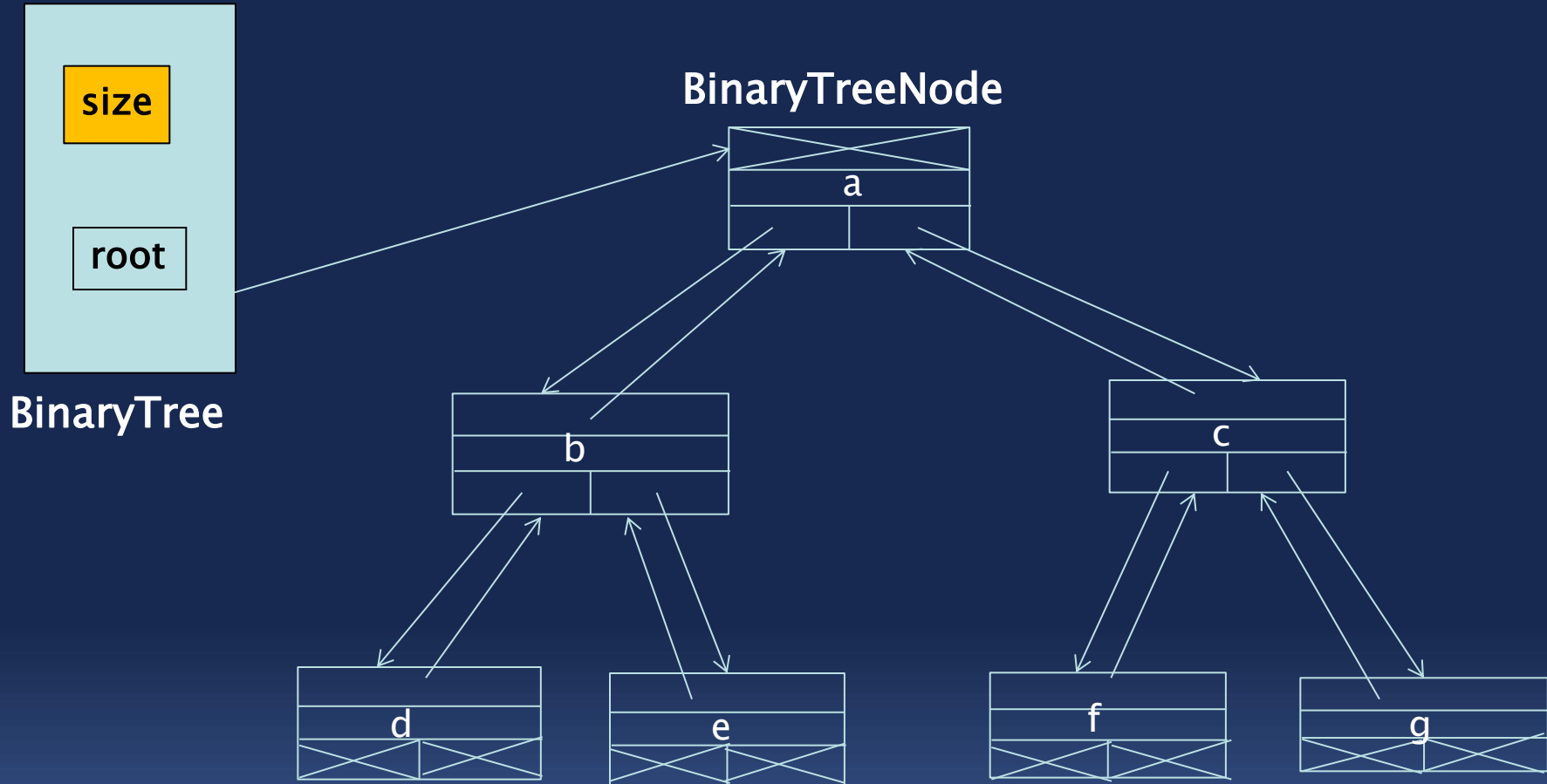
◆ Um nó é representado por um objeto armazenando:

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



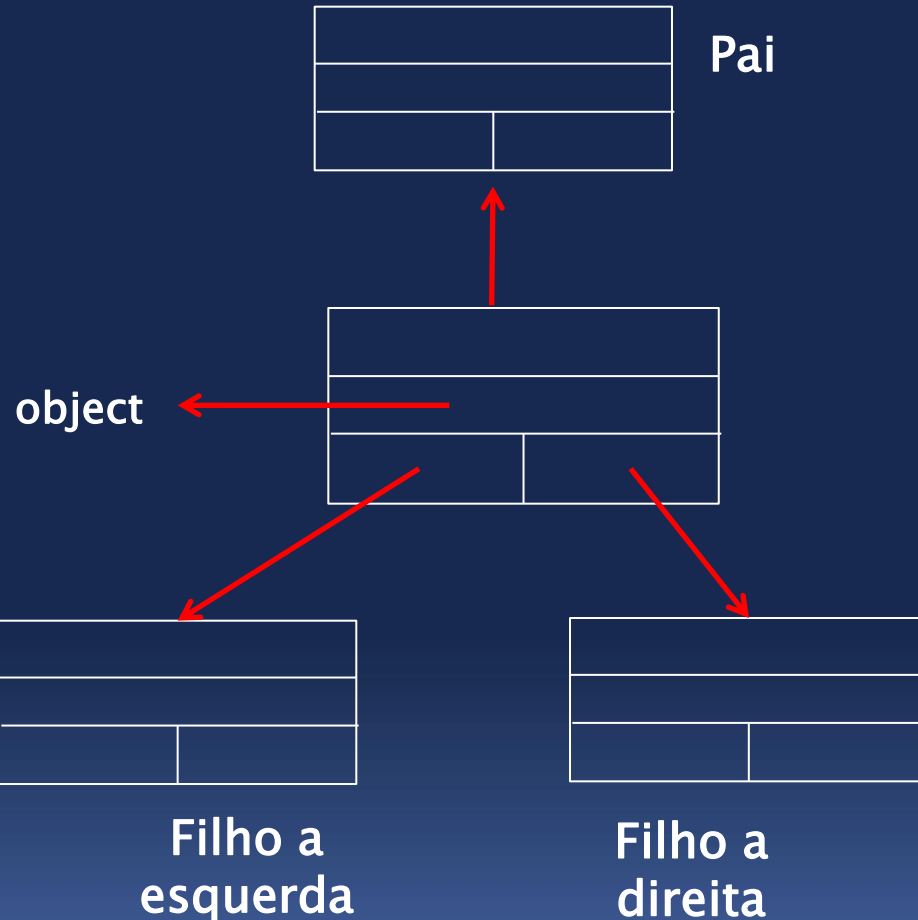
Representação por lista ligada

size -> #nós da árvore



Representando nó da Árvore

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

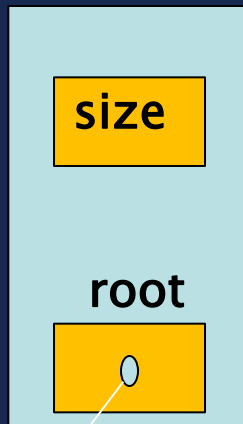


```
struct node {
    int data;
    struct node * parent;
    struct node * left;
    struct node * right;
};

struct tree {
    int size;
    struct node * root;
};
```

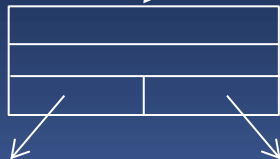

Representando a Árvore

Size -> #nós da árvore



BinaryTree

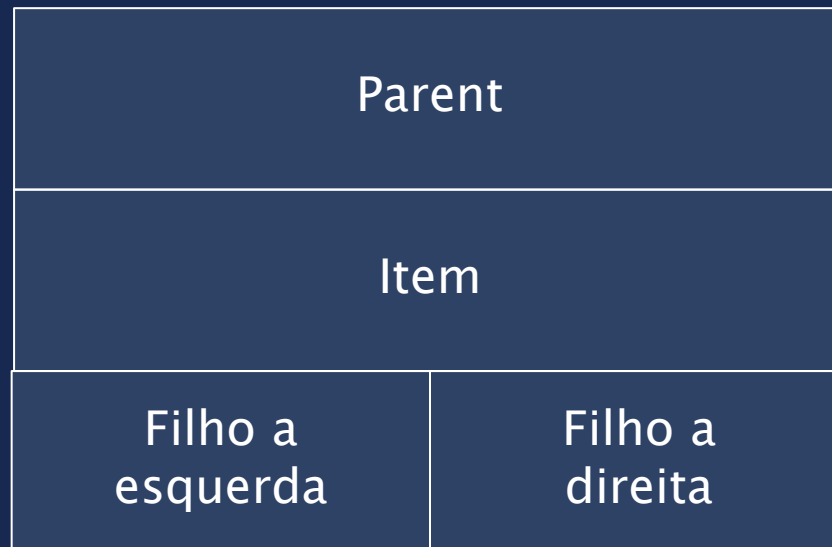
No_root



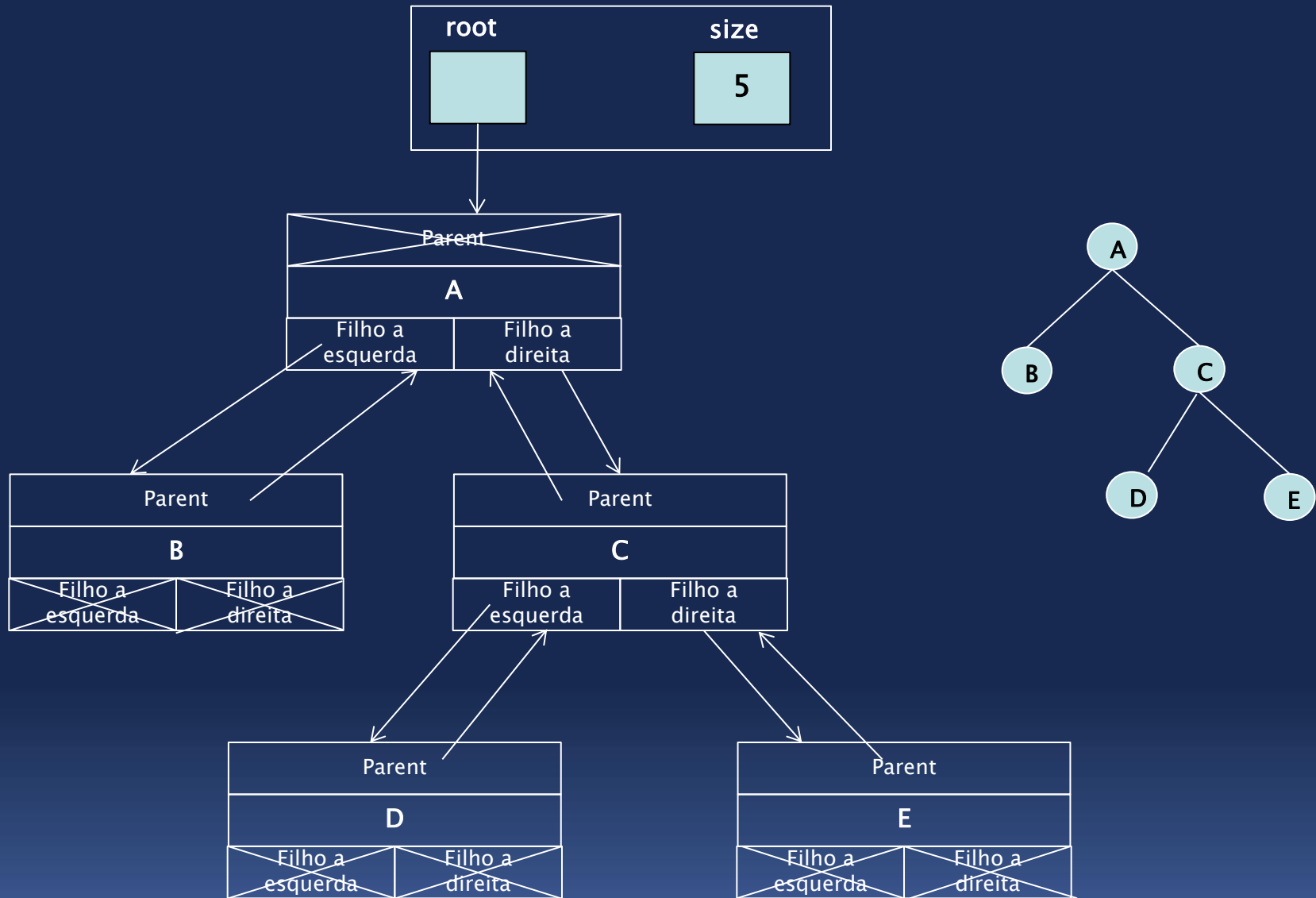
```
struct node {
    int data;
    struct node * parent;
    struct node * left;
    struct node * right;
};

struct tree {
    int size;
    struct node * root;
};
```

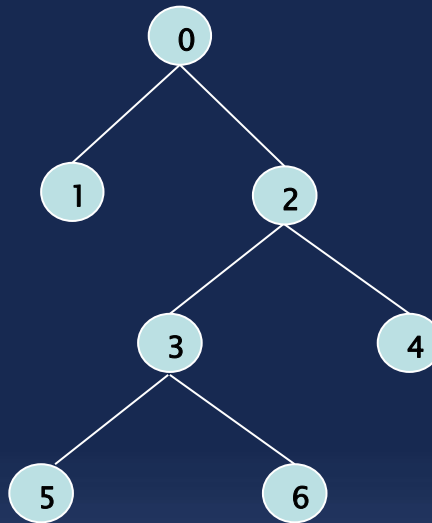
Representando Nó da árvore



Exemplo



Exemplo



```
#include <stdio.h>
#include <stdlib.h>

struct node * node_parent(struct node * );

struct node {

    int data;

    struct node * parent;

    struct node * left;

    struct node * right;

};

struct tree {

    int size;

    struct node * root;

};

enum boolean {
    true = 1, false = 0
};

typedef enum boolean bool;

struct node * root = NULL;

int size = 0;
```

```
void insert_root(int);
struct node * cria_node (int);
struct node * ret_Root();
struct node * left(struct node * );
struct node * right(struct node * );
enum boolean isLeft(struct node * );
enum boolean isRight(struct node * );
int sizeTree();
enum boolean isEmpty();

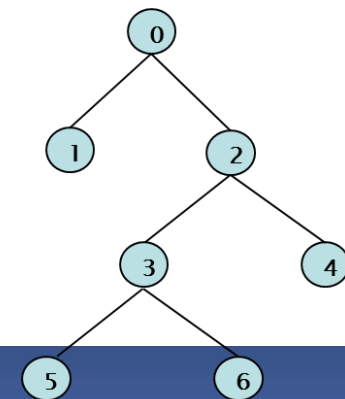
void preorder(struct node * );
void posorder(struct node * );
void inorder(struct node * );

int main() {

    printf(" **** Implementacao de arvores binarias ....\n");

    insert_root(0);

    struct node * no_1 = cria_node(1);
    struct node * no_2 = cria_node(2);
    struct node * no_3 = cria_node(3);
    struct node * no_4 = cria_node(4);
    struct node * no_5 = cria_node(5);
    struct node * no_6 = cria_node(6);
```



```
struct node * no_6 = cria_node(6);

root->left  = no_1;
root->right = no_2;

no_2->left = no_3;
no_2->right = no_4;

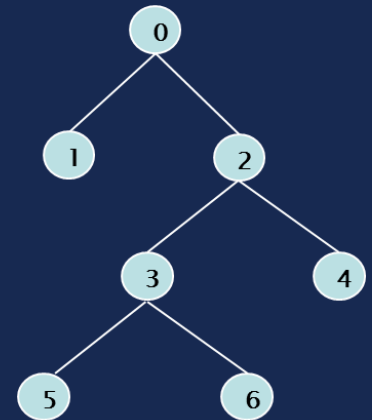
no_3->left = no_5;
no_3->right = no_6;

printf ("\n ***** preorder *****\n\n");
preorder(root);

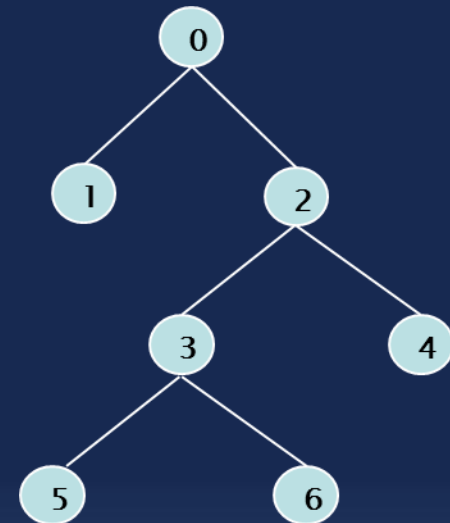
printf ("\n\n ***** posorder *****\n\n");
posorder(root);

printf ("\n\n ***** inorder *****\n\n");
inorder(root);

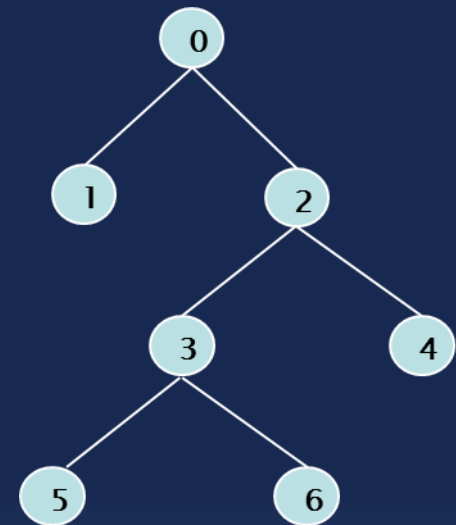
return 0;
}
```



```
void insert_root( int valor) {  
  
    struct node * novo_node = cria_node(valor);  
  
    root = novo_node;  
  
    size = 1;  
}  
  
struct node *  ret_Root () {  
  
    return root;  
}  
  
int sizeTree() {  
  
    return size;  
}  
  
enum boolean isEmpty() {  
  
    if (size == 0)  
        return true;  
    return false;  
}
```




```
struct node * left(struct node * ponteiro) {  
    if (ponteiro -> left == NULL)  
        return NULL;  
  
    return ponteiro -> left;  
}  
  
struct node * right (struct node * ponteiro) {  
    if (ponteiro -> right == NULL)  
        return NULL;  
  
    return ponteiro -> right;  
}  
  
enum boolean isLeft(struct node * ponteiro ) {  
    if (ponteiro -> left == NULL )  
        return false;  
    return true;  
}  
  
enum boolean isRight (struct node * ponteiro ) {  
    if (ponteiro -> right == NULL )  
        return false;  
    return true;  
}
```



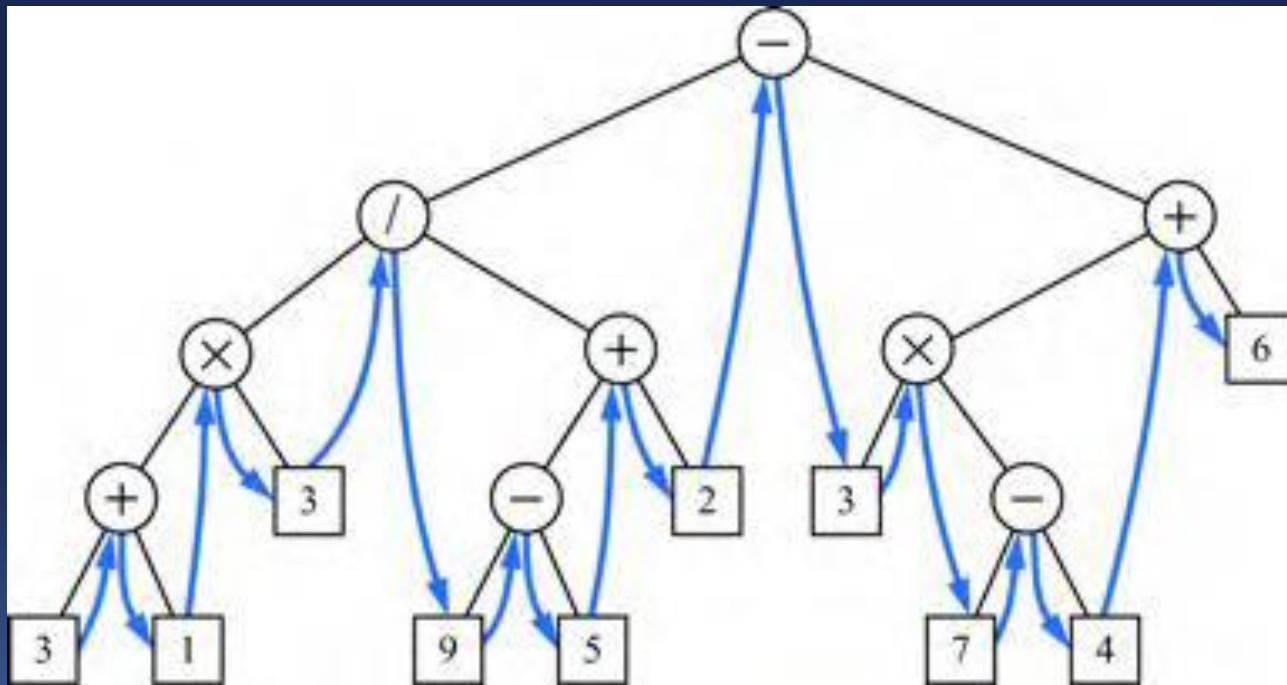
```
struct node * cria_node (int valor) {  
  
    struct node * new_node ;  
  
    new_node = (struct node *) malloc (sizeof (struct node));  
  
    new_node -> left = NULL;  
    new_node -> right = NULL;  
    new_node -> parent = NULL;  
    new_node -> data = valor;  
  
    return new_node;  
}
```

```
void preorder(struct node * ponteiro ) {  
  
    printf ("\t %d" , ponteiro -> data );  
  
    if (isLeft(ponteiro))  
        preorder(ponteiro ->left);  
  
    if (isRight(ponteiro))  
        preorder(ponteiro ->right);  
  
}
```

```
void posorder(struct node * ponteiro ) {  
  
    if (isLeft(ponteiro))  
        posorder(ponteiro ->left);  
  
    if (isRight(ponteiro))  
        posorder(ponteiro ->right);  
  
    printf ("\t %d" , ponteiro -> data );  
}
```

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.



Travessia Inorder

```
void inorder (struct node * ponteiro ) {  
  
    if (isLeft(ponteiro))  
        inorder(ponteiro ->left);  
  
    printf ("\t %d" , ponteiro -> data );  
  
    if (isRight(ponteiro))  
        inorder(ponteiro ->right);  
}
```