



Unidade 5 – Análise de Algoritmos com Estruturas de Dados Hierárquicas – Parte 3

Árvores Binárias de Pesquisa



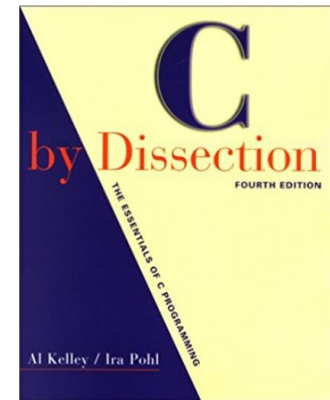
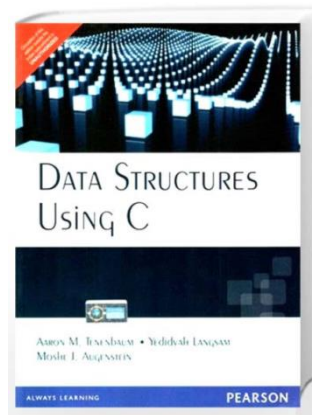
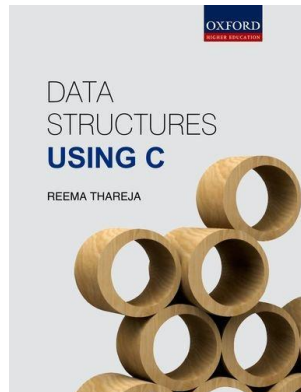
Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP
aparecidovfreitas@gmail.com





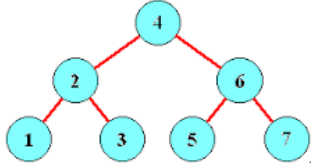
Bibliografia

- 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





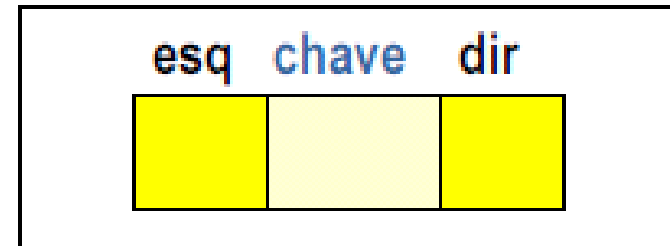
Árvore Binária de Pesquisa



- Também conhecida por:
 - Árvore Binária de Busca
 - Árvore Binária Ordenada
 - Search Tree (em inglês)



- Apresentam uma relação de ordem entre os nós.
- A ordem é definida por um campo **chave** (key).
- Não permite chaves duplicadas.



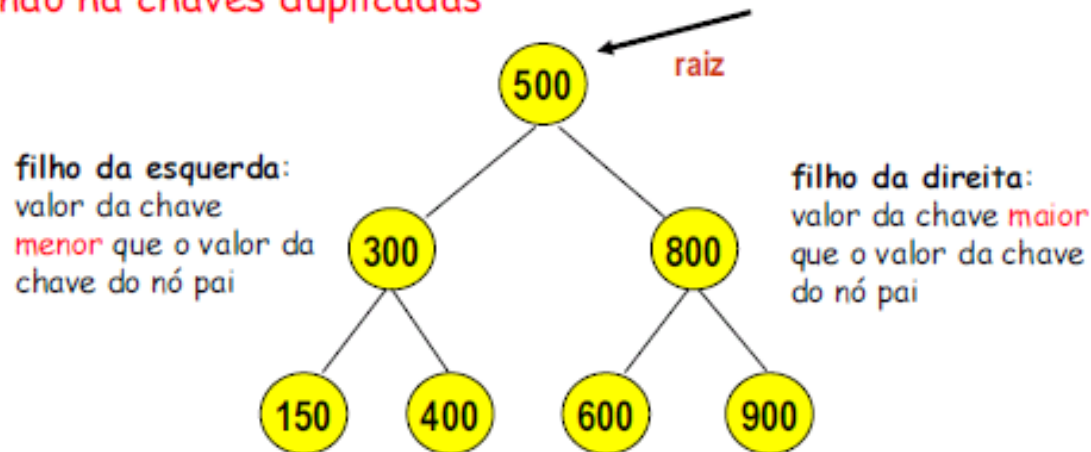


Árvore Binária de pesquisa

■ Definição de Niklaus Wirth:

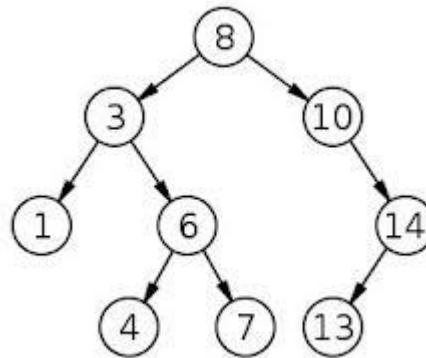
Árvore que se encontra organizada de tal forma que, para cada nó t_i , todas as chaves da sub-árvore:
à **esquerda** de t_i são **menores** que t_i e
à **direita** de t_i são **maiores** que t_i .

não há chaves duplicadas





Inserção em Árvore Binária de Pesquisa





Carga da Árvore Binária de pesquisa

```
int[] valores = { 17,49,14,23,27,15,2,1,34,10,12 } ;
```

```
String[] nomes = {  
"Paulo","Ana","José","Rui","Paula","Bia","Selma","Carlos","Silvia","Teo","Saul" } ;
```

**A partir das listas acima, implementar
a árvore binária de busca.**



```
while( n < (docum  
{  
  
    n++;  
    calc = ev  
    i++  
    i++
```





Inserção em uma árvore de busca binária

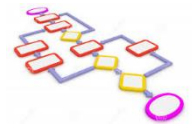
Lembrando que ...

- A sub-árvore da **direita** de um nó deve possuir chaves **maiores** que a chave do pai.
- A sub-árvore da **esquerda** de um nó deve possuir chaves **menores** que a chave do pai.

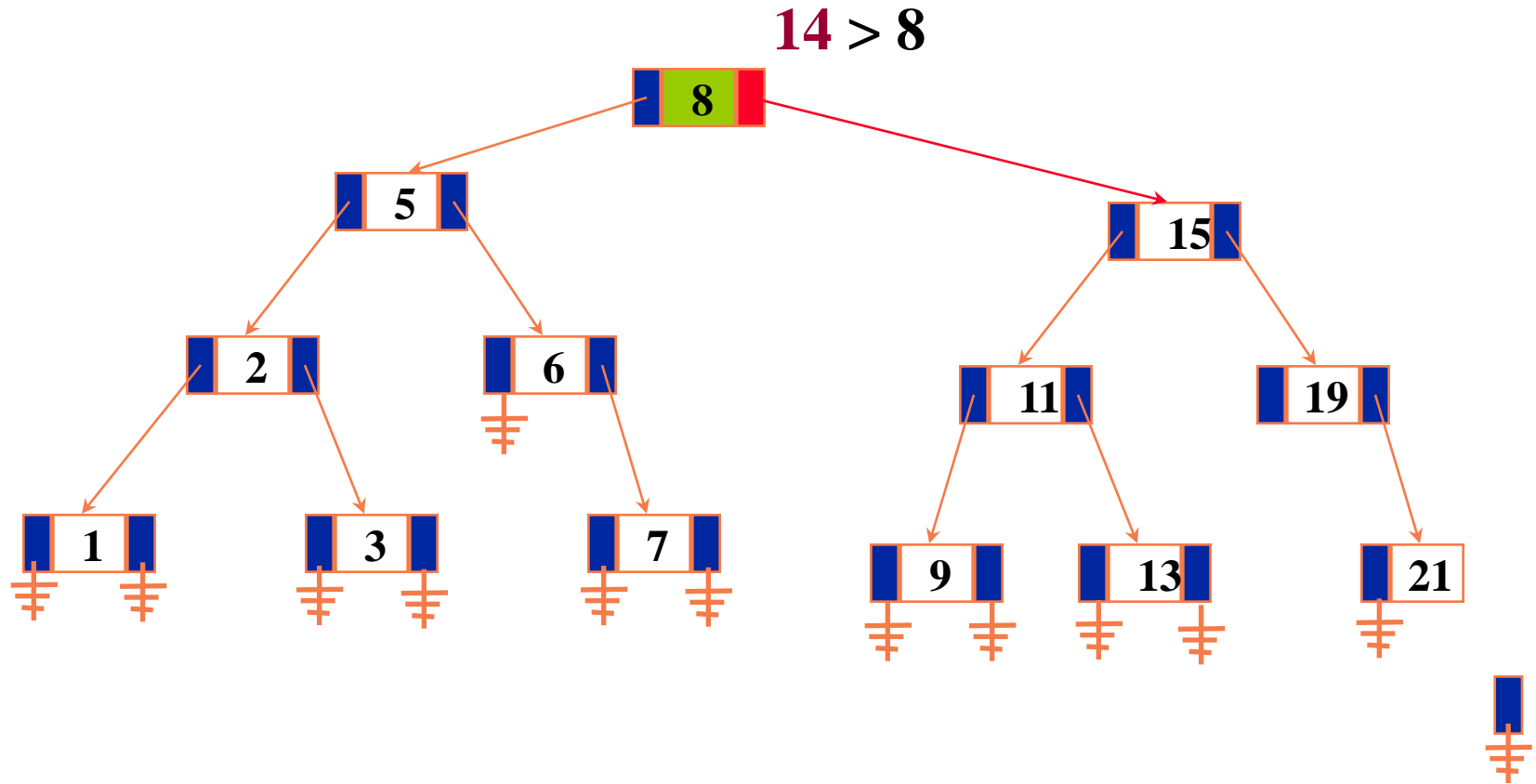
Princípio Básico

- ◆ Percorrer a árvore até encontrar um nó sem filho, de acordo com os critérios acima.



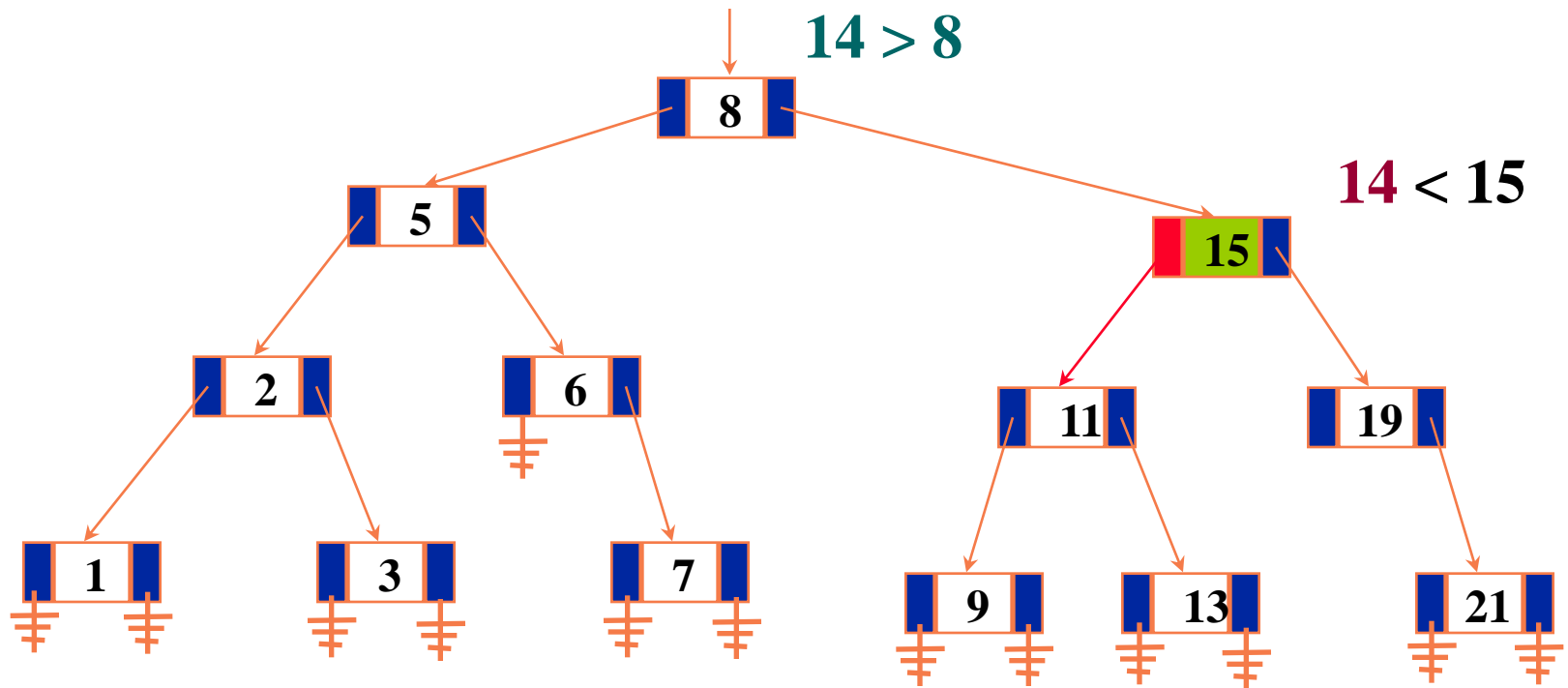


Exemplo – Inserção de elemento com chave 14

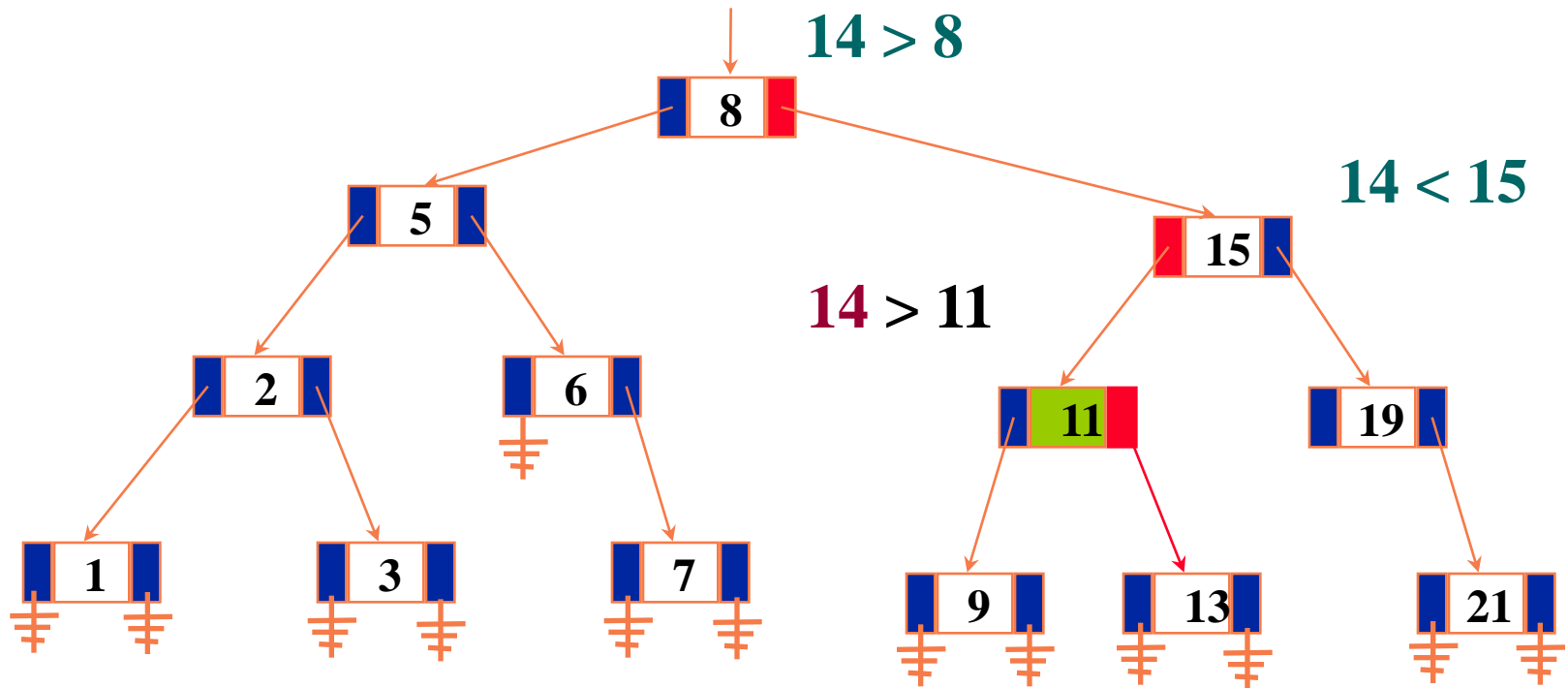


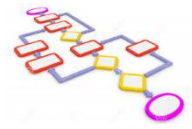


Exemplo – Inserção de elemento com chave 14

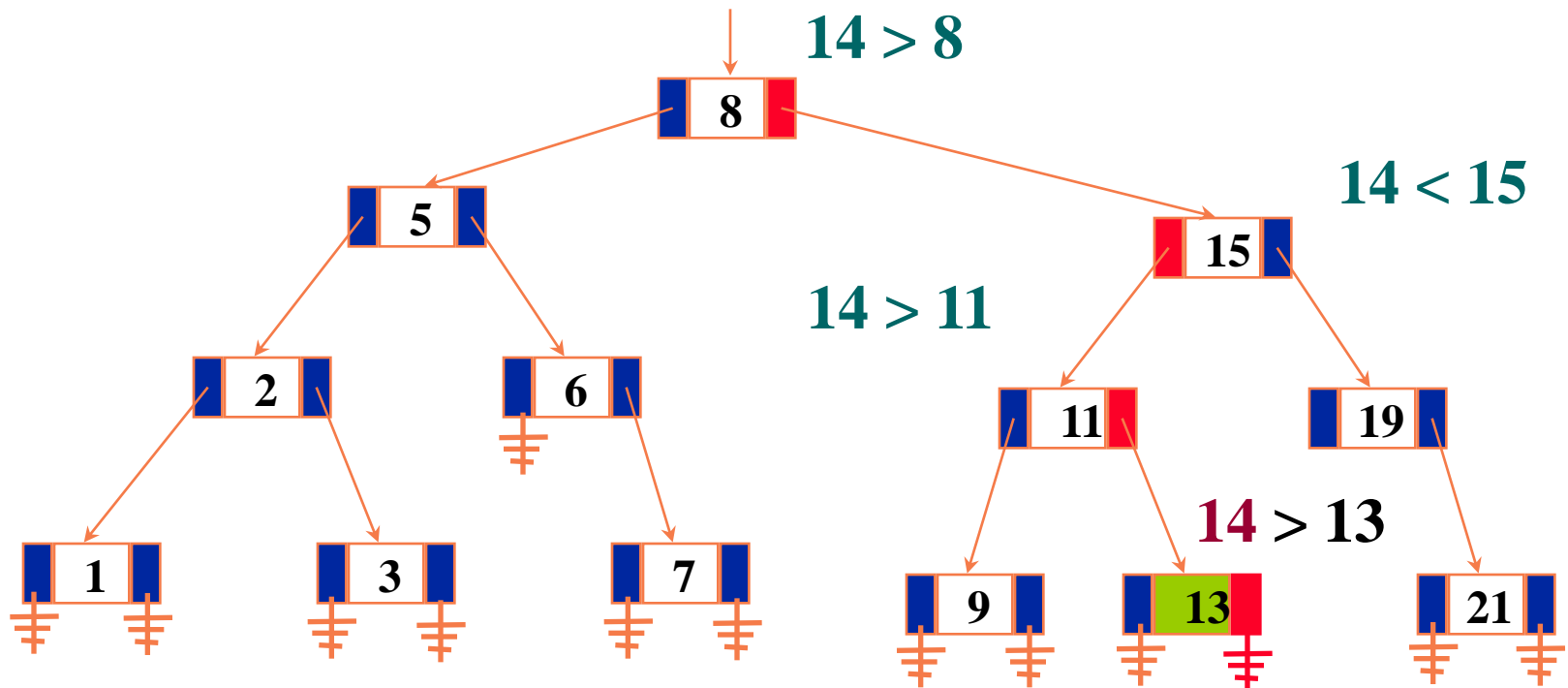


Exemplo – Inserção de elemento com chave **14**

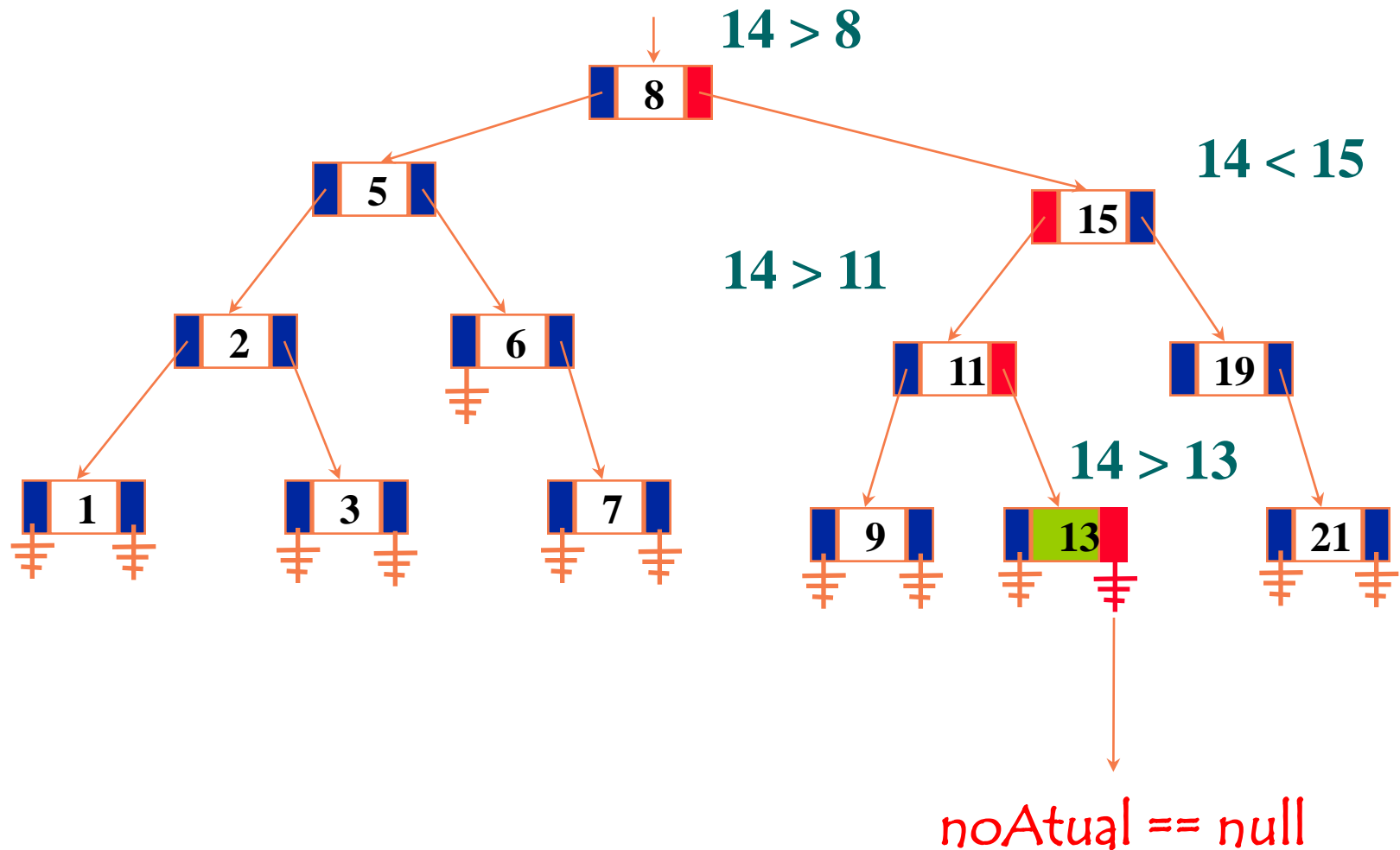




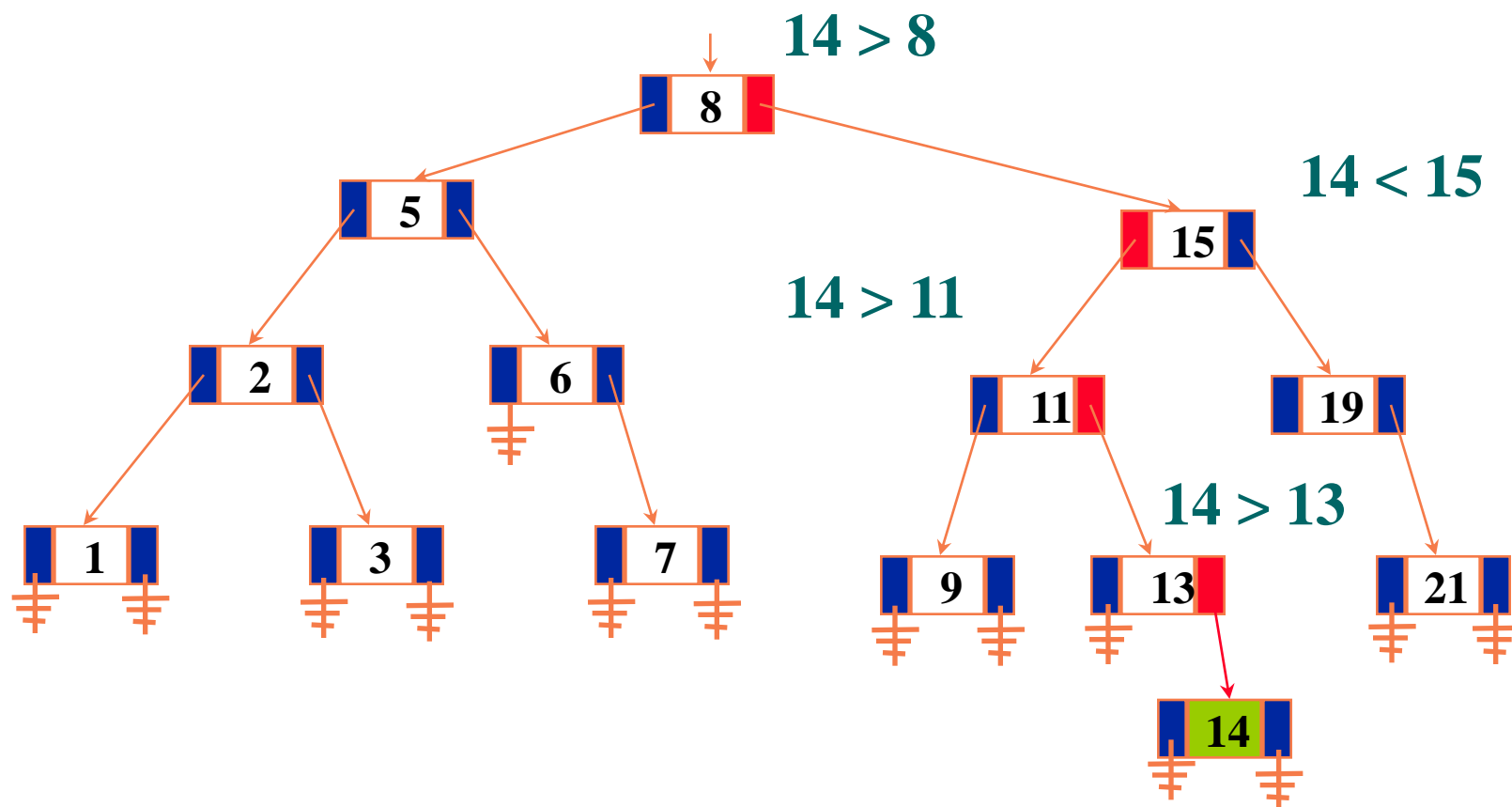
Exemplo – Inserção de elemento com chave 14



Exemplo – Inserção de elemento com chave 14



Exemplo – Inserção de elemento com chave 14



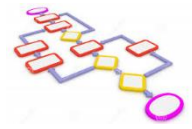
Implementação da função addnode()

```

public void addNode(int chave, String nome) {

    SearchTreeNode newNode = new SearchTreeNode(chave,nome);
    if (root == null)
        this.insert_root(newNode);
    else {
        SearchTreeNode NodeTrab = this.root;
        NodeTrab = this.root;
        while (true) {
            if (chave < NodeTrab.key) {
                if (NodeTrab.left == null) {
                    NodeTrab.left = newNode;
                    newNode.parent = NodeTrab;
                    newNode.nome = nome;
                    return;
                }
                else NodeTrab = NodeTrab.left;
            }
            else {
                if (NodeTrab.right == null) {
                    NodeTrab.right = newNode;
                    newNode.parent = NodeTrab;
                    newNode.nome = nome;
                    return;
                }
                else NodeTrab = NodeTrab.right;
            }
        }
    }
}

```



Busca em árvores de pesquisa

- ✓ Em uma árvore binária é possível encontrar qualquer chave existente **X** atravessando-se árvore:
 - ✓ sempre **à esquerda** se **X** for **menor** que a chave do nó visitado e
 - ✓ sempre **à direita** toda vez que for **maior**.
 - ✓ A escolha da direção de busca só depende de **X** e da chave que o nó atual possui.
- ✓ A busca de um elemento em uma **árvore balanceada** com **n** elementos toma tempo médio menor que $\log_2(n)$, tendo a busca então $O(\log_2 n)$.



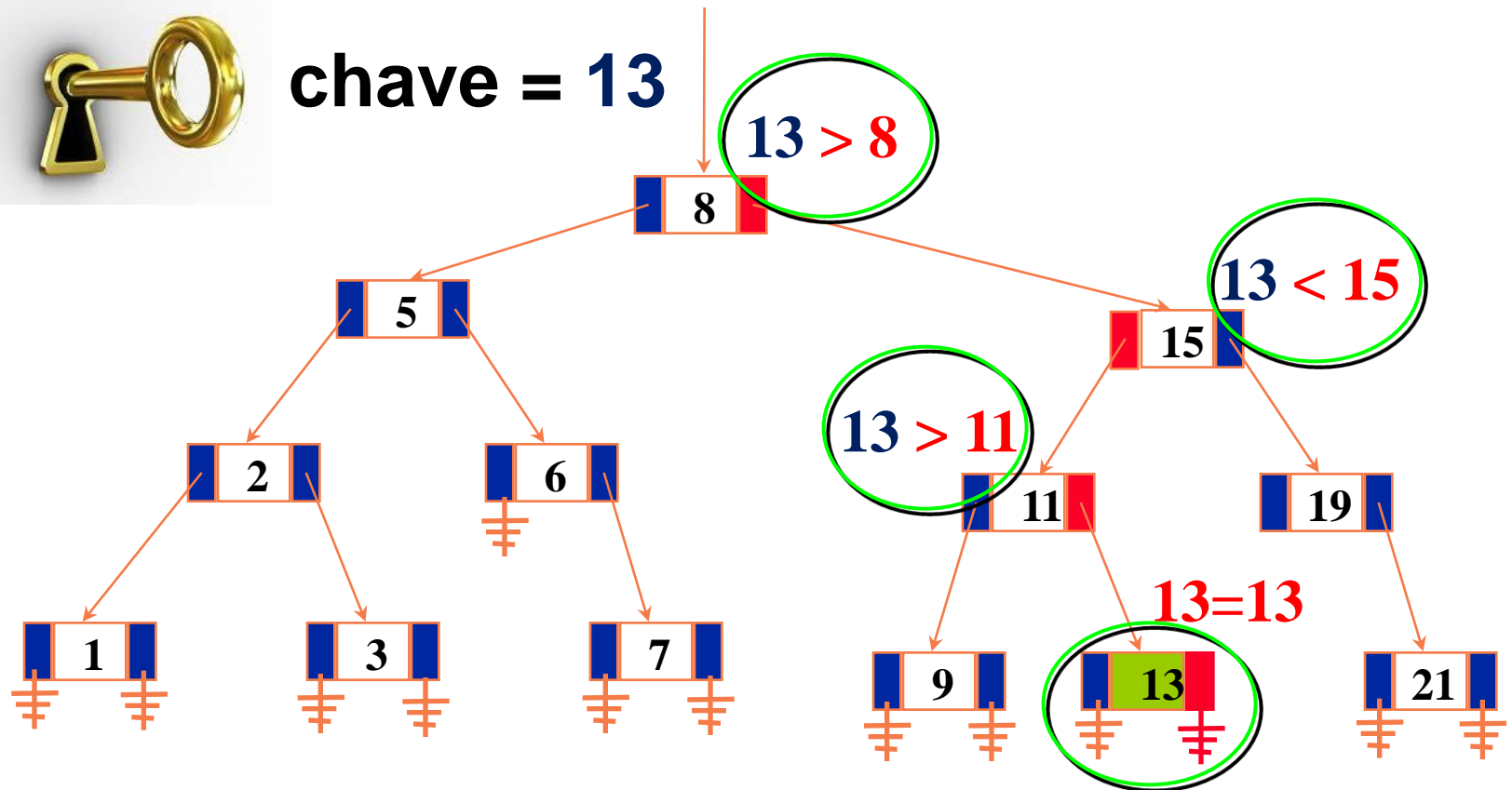
n	$\log_2(n)$
1	0,00
10	3,32
13	3,70
20	4,32
50	5,64
100	6,64
200	7,64
500	8,97



Exemplo – Busca da chave 13



chave = 13





Algoritmo iterativo de search em árvore de busca

```
Node buscaChave (int chave) {  
    Node noAtual = raiz; // inicia pela raiz  
  
    while (noAtual != null && noAtual.item != chave) {  
  
        if (chave < noAtual.key)  
            noAtual=noAtual.left ; // caminha p/esquerda  
        else  
            noAtual=noAtual.right; // caminha p/direita  
    }  
    return noAtual;  
}
```





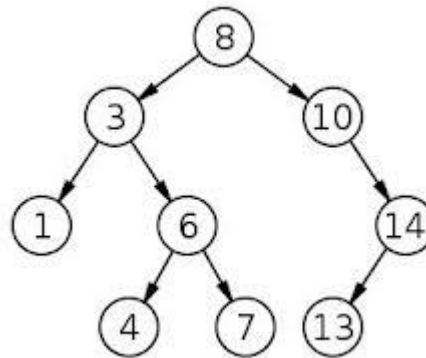
Implementação – Busca

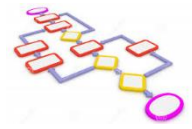
```
public SearchTreeNode Search_key(int key) {  
  
    SearchTreeNode nodeTrab = this.root; // inicia pela raiz  
  
    while (nodeTrab != null && nodeTrab.key != key) {  
  
        if (key < nodeTrab.key)  
            nodeTrab = nodeTrab.left;  
        else  
            nodeTrab = nodeTrab.right;  
    }  
    return nodeTrab;  
}
```





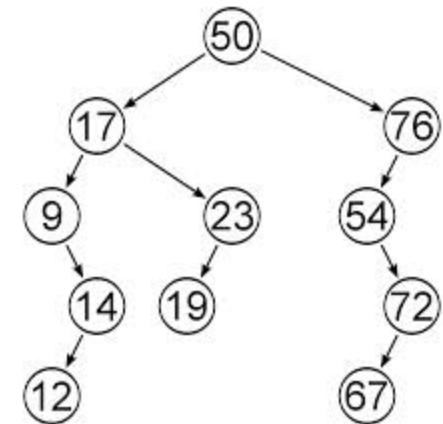
Eliminação em Árvores Binárias de Busca





Eliminação em Árvores Binárias de Busca

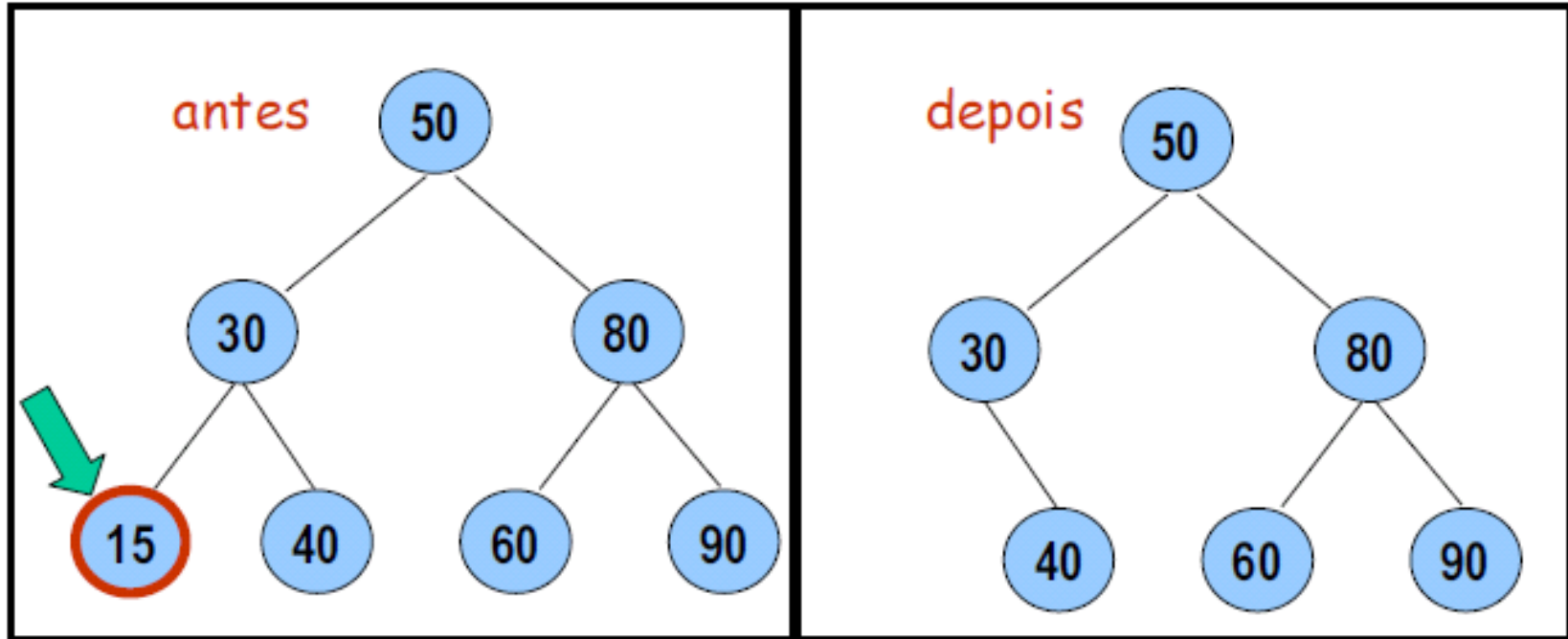
- ✓ A eliminação é mais complexa do que a inserção.
- ✓ A razão básica é que a característica organizacional da árvore não deve ser alterada.
 - A sub-árvore **direita** de um nó deve possuir chaves **maiores** que a do pai.
 - A sub-árvore **esquerda** de um nó deve possuir chaves **menores** que a do pai.
- ✓ Para garantir isto, o algoritmo deve “**remanejar**” os nós.





Caso 1 – Remoção de nó folha

Caso mais simples, basta retirá-lo da árvore

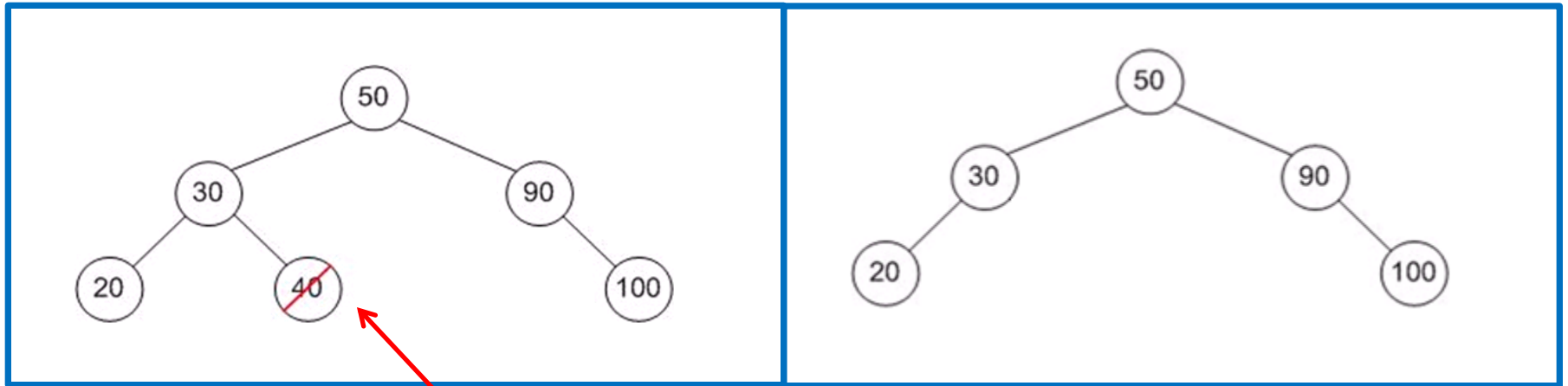


Eliminação da chave 15





Caso 1 – Outro exemplo



Eliminação da chave 40





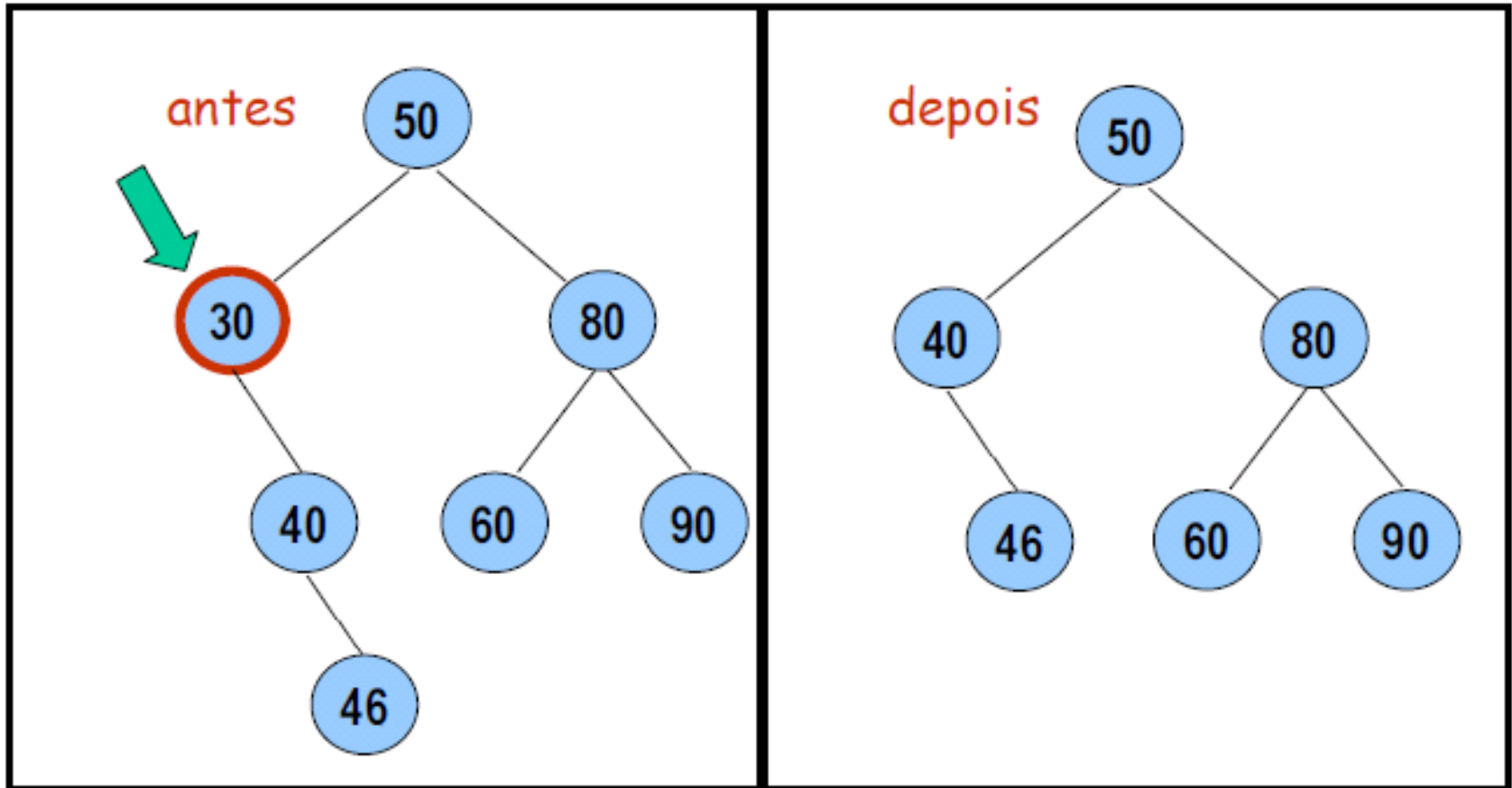
Eliminação de nó que possui uma sub-árvore filha

- ✓ Se o nó a ser removido possuir somente uma sub-árvore filha:
 - ⊕ Move-se essa sub-árvore toda para cima.
 - ⊕ Se o nó a ser excluído é filho esquerdo de seu pai, o seu filho será o novo filho esquerdo deste e vice versa.





Caso 2 – O nó tem somente uma sub-árvore



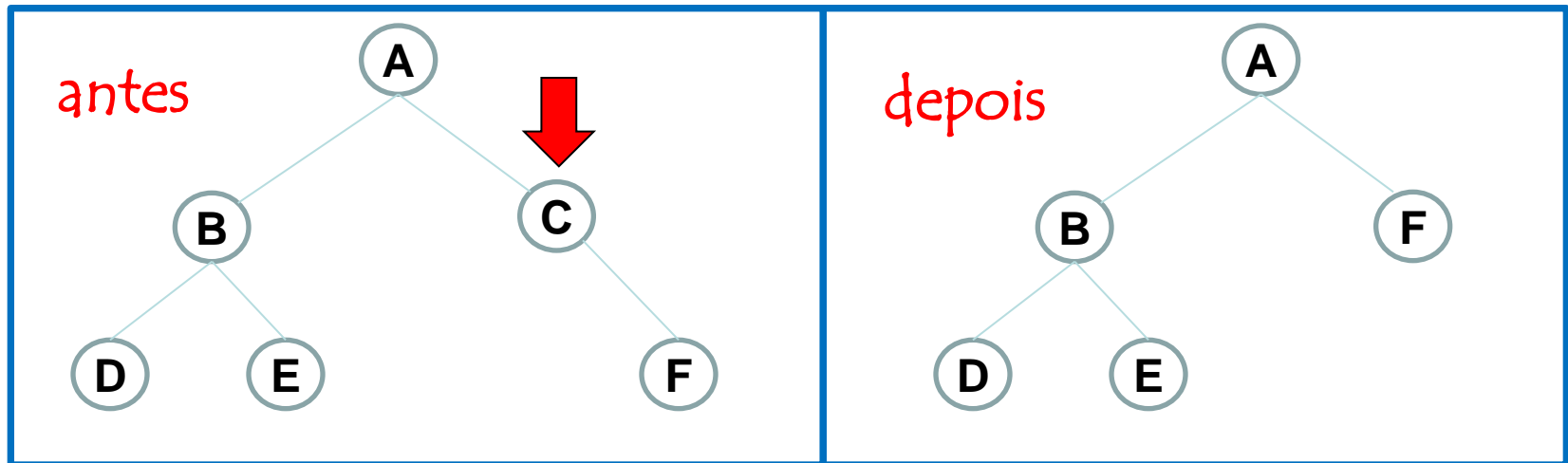
Eliminação da chave 30

O ponteiro do pai aponta para o filho deste



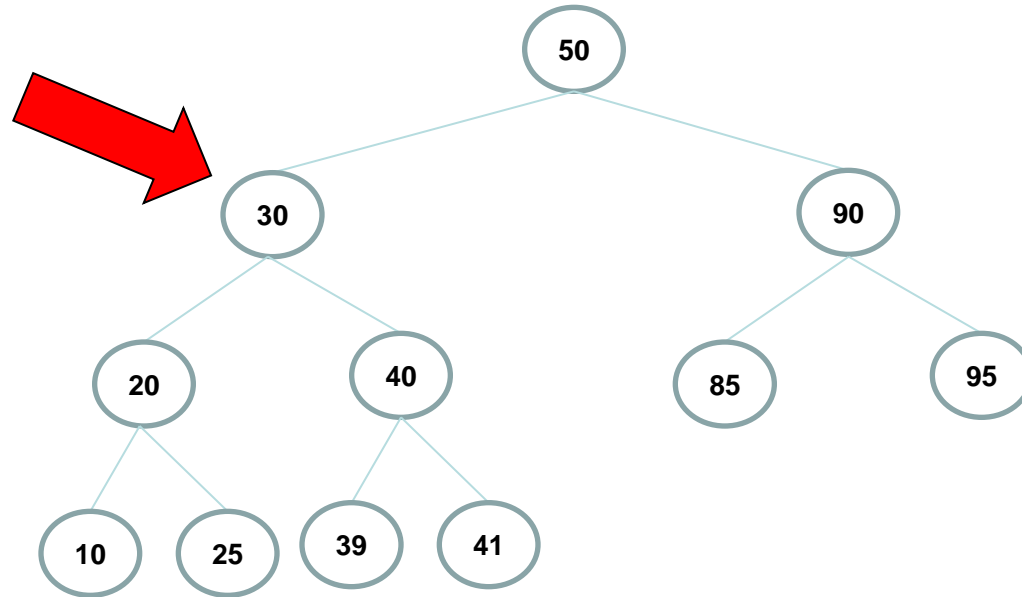


Caso 2 – Outro Exemplo





Eliminação de nó que possui duas sub-árvores filhas

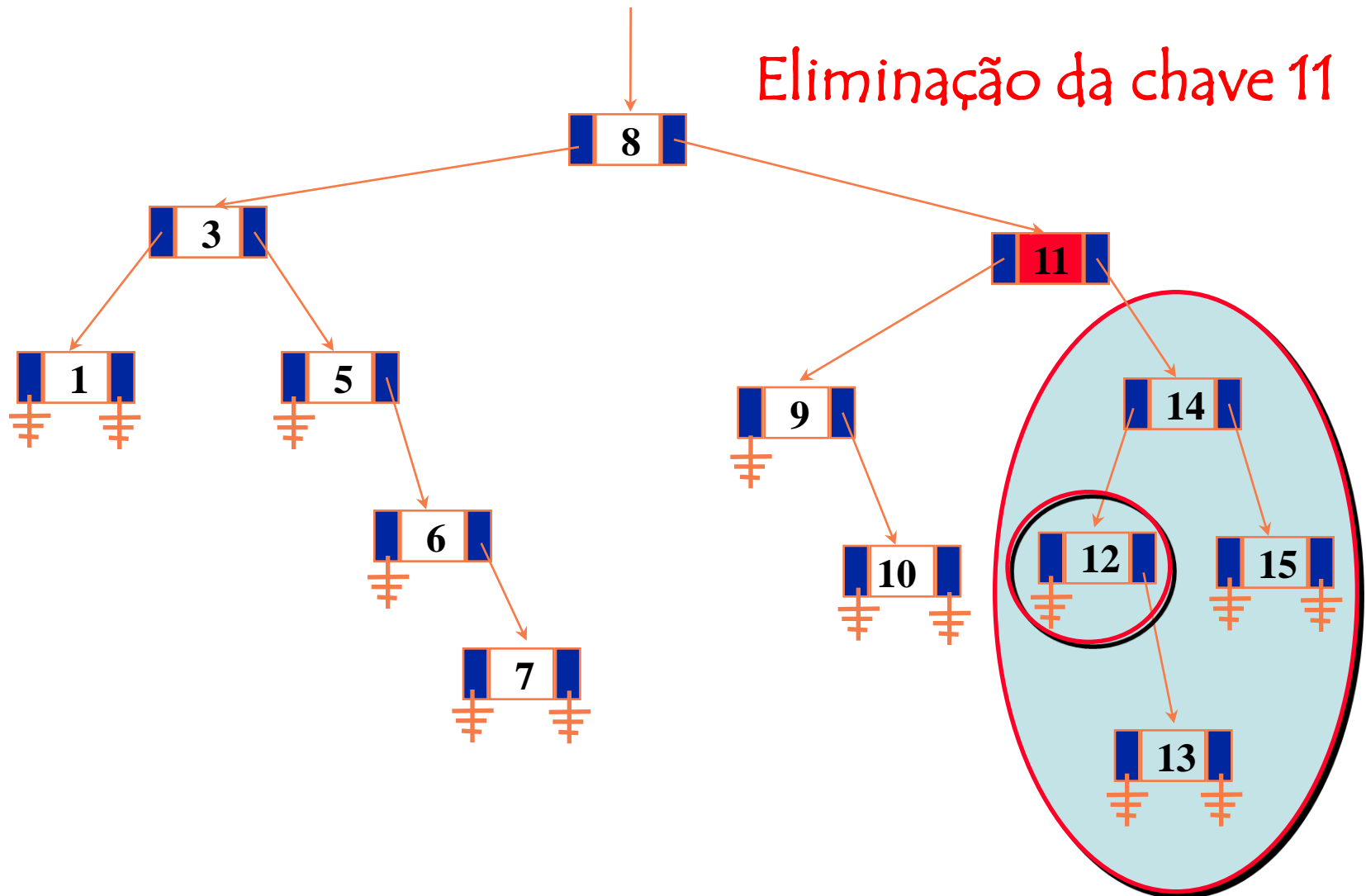


A estratégia geral (Mark Allen Weiss) é:

Substituir a chave retirada pela menor chave da sub-árvore direita

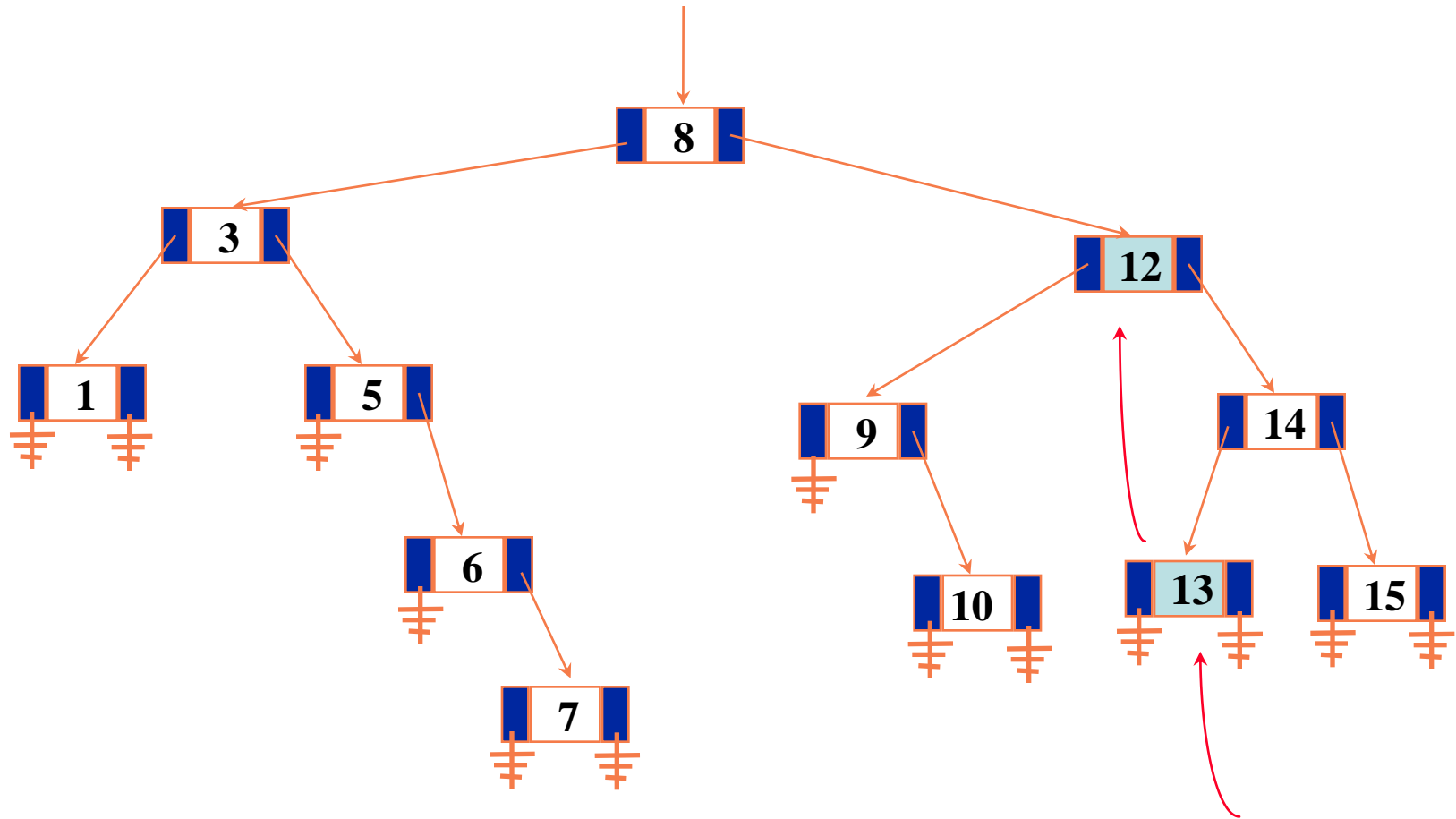


Caso 3 – Nó tem duas sub-árvores



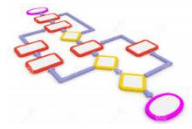
Substituir a chave retirada pela menor chave da sub-árvore direita

Caso 3 – Nó tem duas sub-árvores

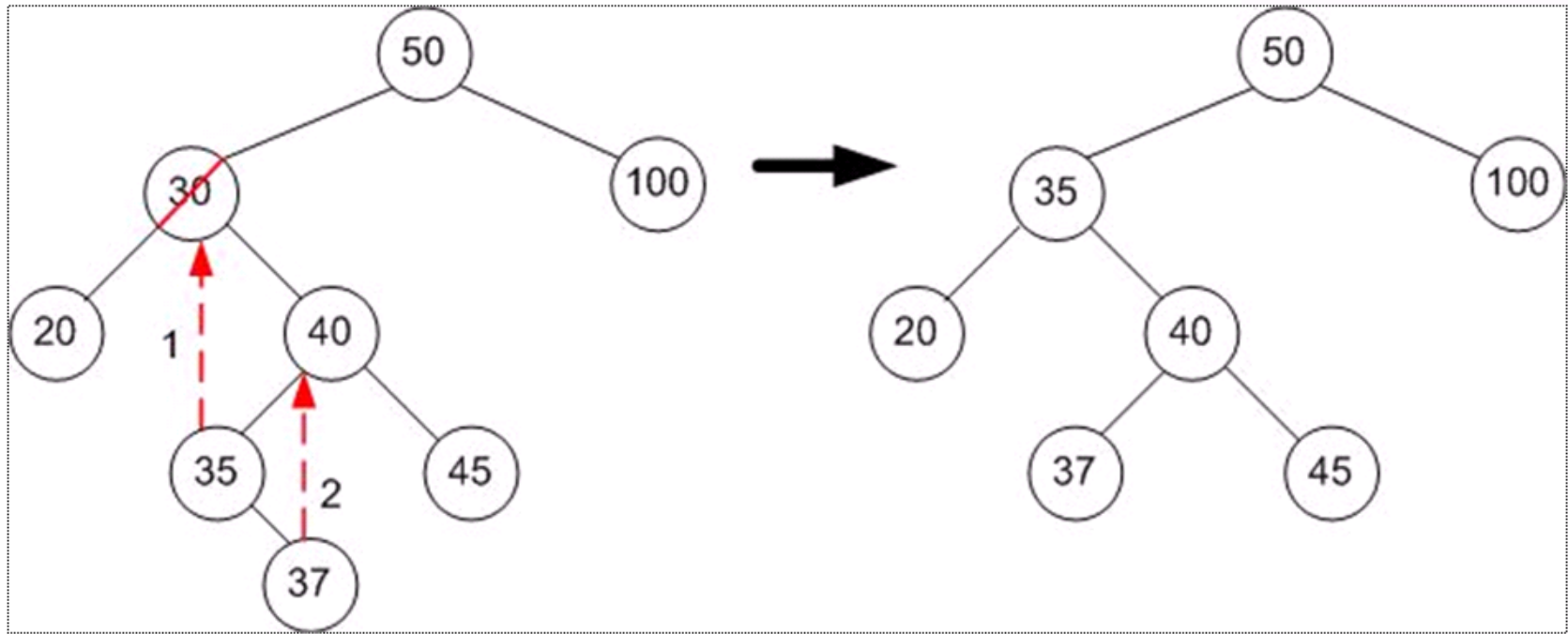


Eliminação da chave 11





Caso 3 – Outro exemplo

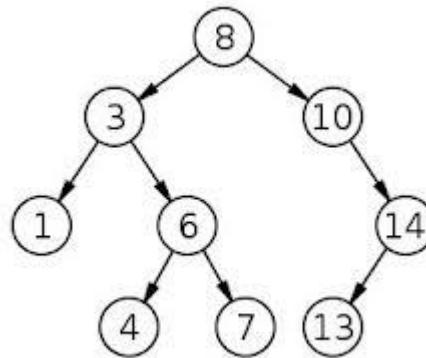


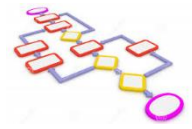
Eliminação da chave 30





Implementação de Árvores Binárias de Busca

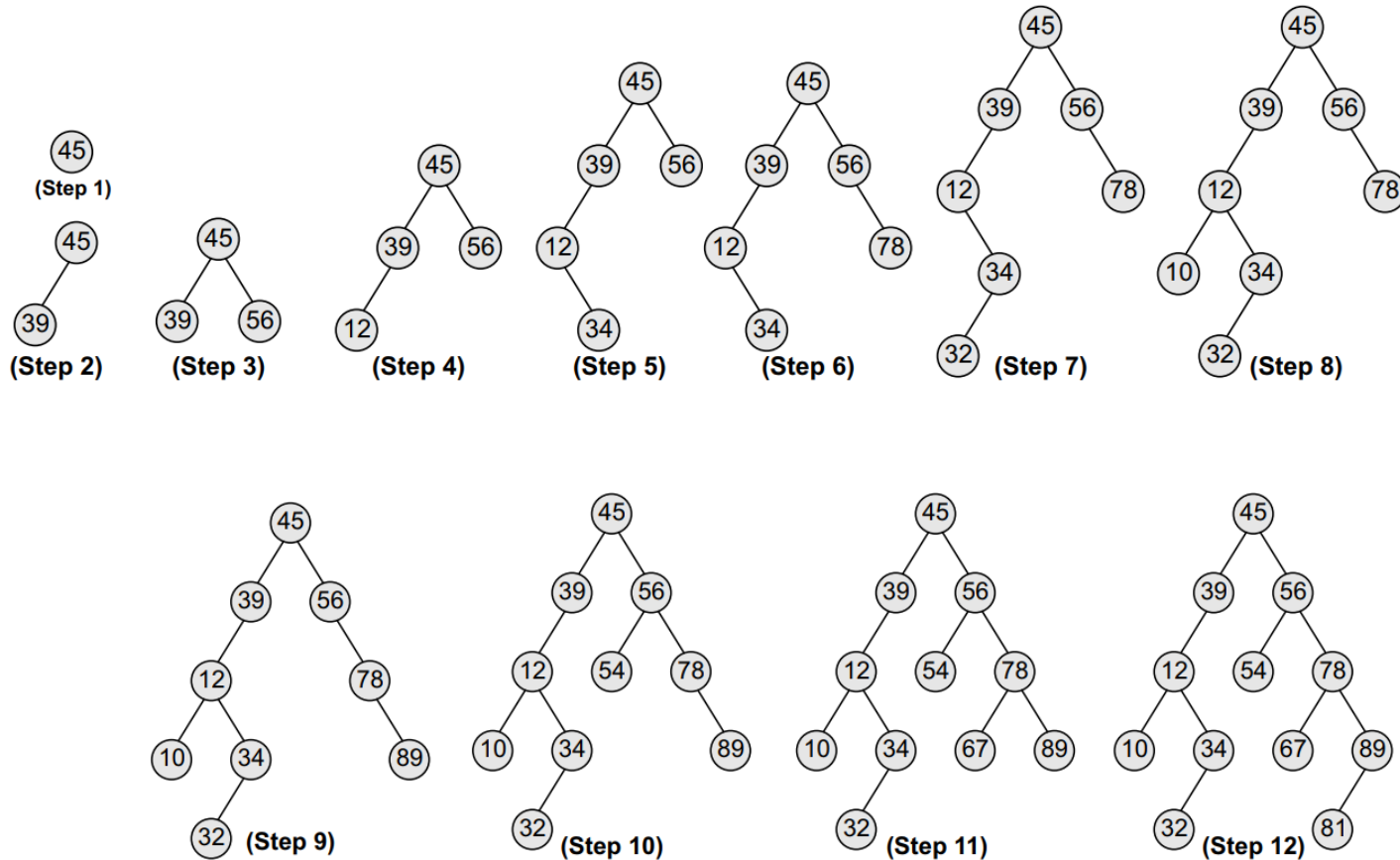




Busca em árvores de pesquisa

- ✓ Criação de uma árvore binária de busca com os seguintes elementos de dados:

45, 39, 56, 12, 34, 78, 32, 10, 89, 54, 67, 81



Criação de Árvore Binária de Busca

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

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

struct node * tree;

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

typedef enum boolean bool;

void insert_node (struct node *);
struct node * cria_node (int);

struct node * left(struct node * );
struct node * right(struct node *);
enum boolean isLeft(struct node *);
enum boolean isRight(struct node *);
enum boolean isEmpty();
void preorder(struct node *);
void posorder(struct node *);
void inorder(struct node *);
```


AGOSTINO
Esempi di finalit 
1984 Lucio - 1990
1984 Agostino - 1990
1984 - 1990
1984 - 1990
1984 - 1990

Criação de Árvore Binária de Busca

```
void insert_node( struct node * ponteiro) {  
  
    if (tree == NULL)  
        tree = ponteiro;  
    else {  
  
        struct node * nodeTrab = tree;  
        struct node * saveNode = NULL;  
  
        while ( nodeTrab != NULL ) {  
            saveNode = nodeTrab;  
            if ( ponteiro -> data > nodeTrab->data )  
                nodeTrab = nodeTrab->right;  
            else  
                nodeTrab = nodeTrab->left;  
        }  
  
        if (ponteiro -> data > saveNode->data )  
            saveNode->right = ponteiro;  
        else  
            saveNode->left = ponteiro;  
  
        ponteiro->parent = saveNode;  
    }  
}
```



Criação de Árvore Binária de Busca

```
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;  
}
```



```
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;  
}
```



Criação de Árvore Binária de Busca

```
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;  
}
```



Criação de Árvore Binária de Busca

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



Criação de Árvore Binária de Busca

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





Criação de Árvore Binária de Busca

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



Busca em árvores de pesquisa

```
int main() {

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

    struct node * Nodetrab;

    int dados[] = { 45, 39, 56, 12, 34, 78, 32, 10, 89, 54, 67, 81 };

    for (int i = 0; i < 12 ; i++) {
        Nodetrab = cria_node(dados[i]);
        insert_node(Nodetrab);
    }

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

    int key;
    printf ("\n\n Entre com uma chave de pesquisa: ");
    scanf ("%d", &key);

    printf ("\n\n chave a ser consultada: %d", key);
    search_key(key);

    return 0;
}
```



Busca em árvores de pesquisa

```
enum boolean search_key(int key) {

    if (tree == NULL) {
        printf("\n\ narvore nula, chave nao existente...");
        return false;
    }
    else {

        struct node * nodeTrab = tree;

        while ( nodeTrab != NULL ) {
            if ( nodeTrab->data == key ) {

                printf("\n\ nkey %d encontrada na arvore...", key);
                return true;
            }
            else {
                if (key > nodeTrab->data)
                    nodeTrab = nodeTrab->right;
                else nodeTrab = nodeTrab->left;
            }
        }

        printf("\n\ nkey %d NAO encontrada na arvore...", key);
        return false;
    }
}
```



Busca do menor elemento em uma árvore de pesquisa

```

int main() {

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

    struct node * Nodetrab;

    int dados[] = { -5, 39, 56, 9, 34, 78, 32, 10, -999, 54, 67, 81 };

    for (int i = 0; i < 12 ; i++) {
        Nodetrab = cria_node(dados[i]);
        insert_node(Nodetrab);
    }

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

    if (tree == NULL)
        printf("\n\narvore vazia. nao ha menor elemento...");
    else
        printf ("\n\nmenor elemento: %d " , findMenorElemento(tree) ->data) ;
    return 0;

}

```





Busca do menor elemento em uma árvore de pesquisa

```
struct node * findMenorElemento(struct node * ponteiro) {  
    if ( (ponteiro == NULL) || (ponteiro->left == NULL))  
        return ponteiro;  
    else  
        return findMenorElemento(ponteiro->left);  
}
```

