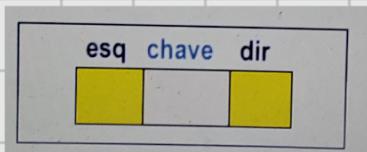


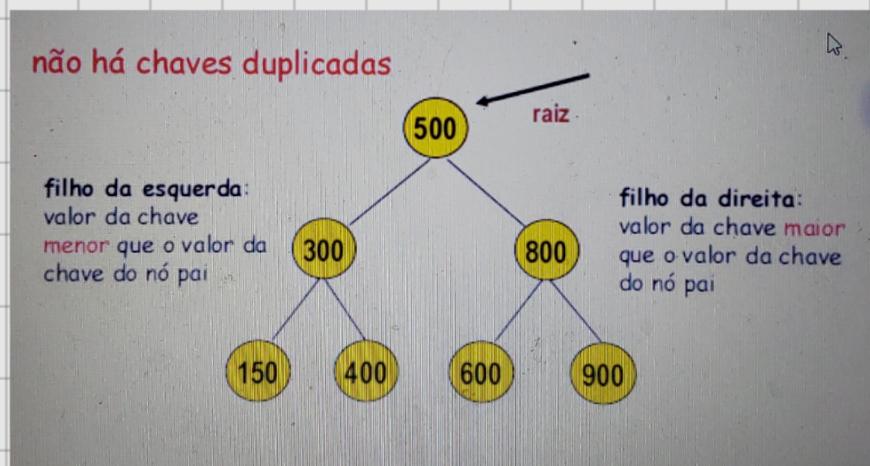
# Árvore Binária de Pesquisa

- Ordem definida por um campo chamado chave
- Não permite chaves duplicadas.



Para cada nó de  $t_i$ , todas as chaves da sub-árvore:

- Na Esquerda de  $t_i$  são menores que  $t_i$
- Na Direita de  $t_i$  são maiores que  $t_i$

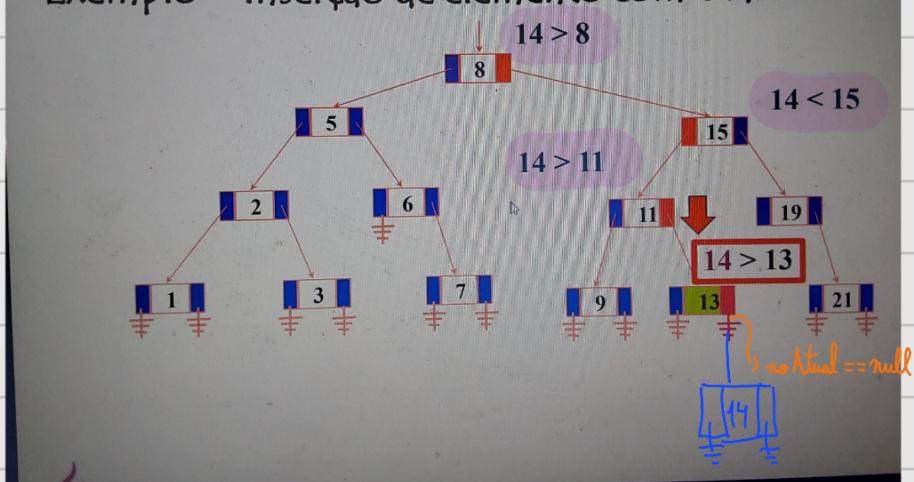


## Inserção em Árvore de Busca Binária

### Princípio Básico

Percorrer a árvore até encontrar um nó sem filho

Exemplo – Inserção de elemento com chave 14



# Código de 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;  
        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;  
            }  
        }  
    }  
}
```

Topicos Avançados de Estrutura de Dados - Unidade 13 - Árvores Binárias de Pesquisa

## Busca em Árvore de Pesquisa

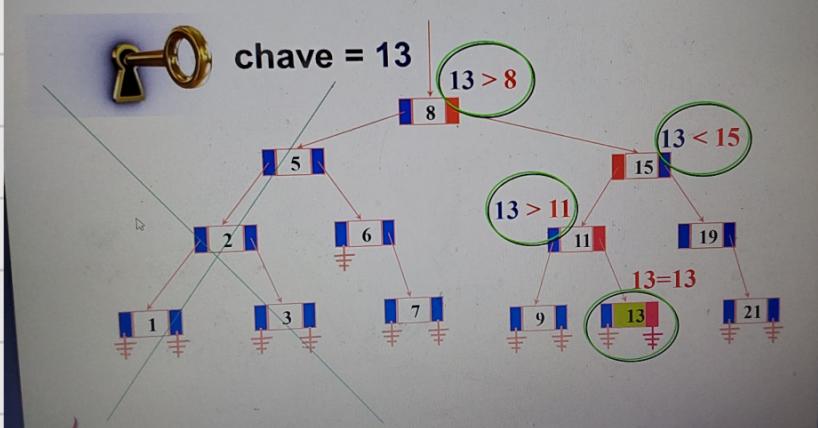
• É possível encontrar qualquer chave existente ✕ passando a árvore.

\* Sempre à esquerda se ✕ for menor que a chave do nó visitado

\* Sempre à direita toda vez que for maior

\* A escolha da direção de busca só depende de ✕ e da chave que o nó atual possui

### Exemplo – Busca da 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;
}

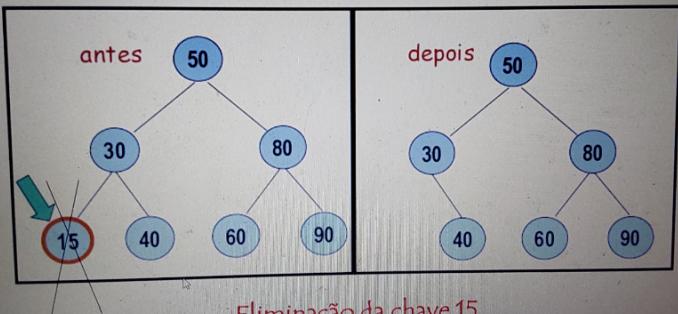
```

## Eliminação em Árvore Binária de Busca

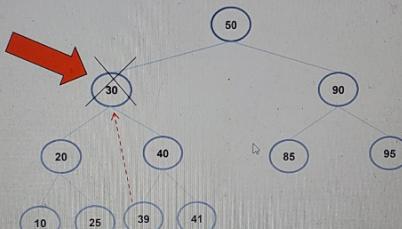
- Mais complexa que a inserção
- Regras básicas é característica organizacional da árvore não deve ser alterada.
  - \* A sub-árvore direita de um nó deve possuir chaves maiores que a do pai
  - \* A sub-árvore esquerda de um nó deve possuir chaves menores que a do pai
- Para garantir isto, o algoritmo deve "remanejar" os nós.

### Caso 1 – Remoção de nó folha

Caso mais simples, basta retirá-lo da árvore



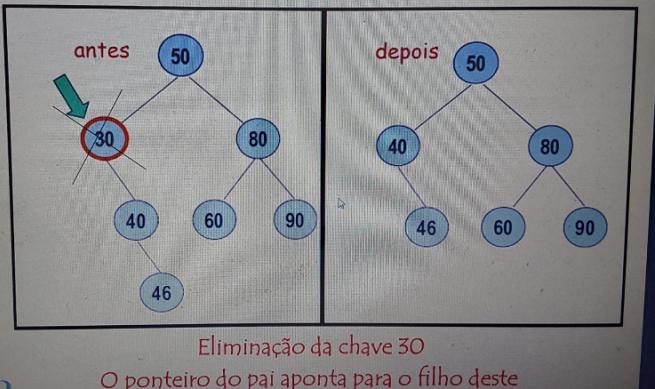
### Eliminação de nó que possui duas sub-árvores filhas



A estratégia geral (Mark Allen Weiss) é:

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

### Caso 2 – O nó tem somente uma sub-árvore



### Caso 3 – Outro exemplo

