

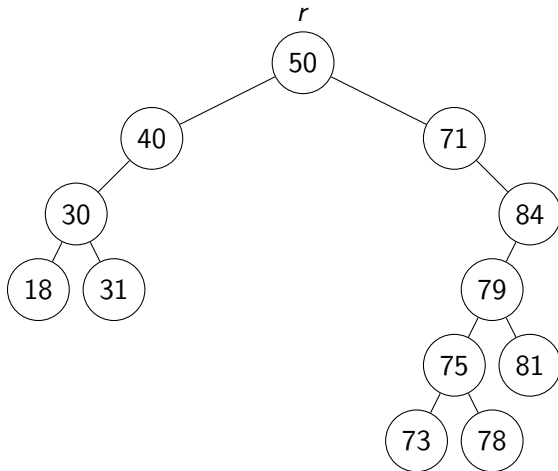
Estruturas de Dados - Árvore Binária de Busca

Árvore binária de busca: definição

- Uma árvore binária de busca (ou BST - *binary search tree*) é uma árvore na qual os elementos estão dispostos segundo a seguinte propriedade:
 - se u e v são nós e u pertence à sub-árvore **esquerda** de v , então $u \rightarrow chave \leq v \rightarrow chave$
 - se u e v são nós e u pertence à sub-árvore **direita** de v , então $u \rightarrow chave > v \rightarrow chave$
- Em outras palavras, dado um nó v :
 - os nós na sub-árvore esquerda de v tem chave menor ou igual à chave de v
 - os nós na sub-árvore direita de v tem chave maior que a chave de v

Árvore binária de busca

- Exemplo de uma BST:



Árvore binária de busca: estrutura de um nó

- Assim como em lista, pilha e fila, uma árvore binária de busca também pode ser representada como um conjunto de nós
- Estrutura de um nó v :
 - *chave*: valor armazenado em v
 - *esq*: ponteiro para a raiz da sub-árvore esquerda de v
 - *dir*: ponteiro para a raiz da sub-árvore direita de v
 - *p*: ponteiro para o nó acima de v (será utilizado posteriormente)

Árvore binária de busca: métodos

- Métodos que vamos analisar:
 - criar BST inicialmente vazia
 - buscar em uma BST
 - incluir em uma BST
 - mínimo e máximo de uma BST
 - percorrer os elementos de uma BST
 - altura de uma BST
 - sucessor e antecessor de um nó em uma BST
 - remover de uma BST
 -

Árvore binária de busca: criar

Algoritmo: CriaBST()

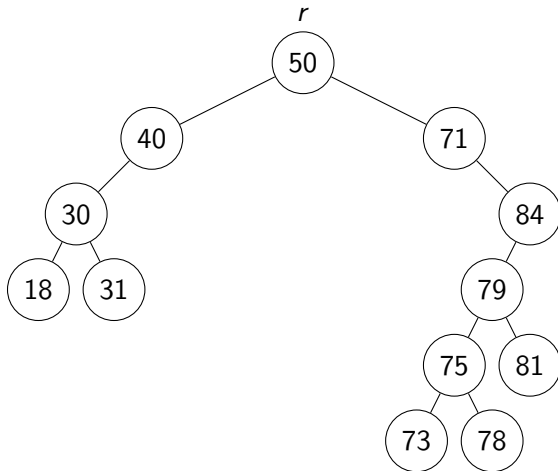
Saída: BST inicialmente vazia

- 1 criar novo nó r
 - 2 $r = \lambda$
 - 3 **retorne** r
-

Complexidade: $O(1)$

Árvore binária de busca: buscar

- Exemplo: buscar por elementos nesta BST



Árvore binária de busca: buscar (recursivo)

Algoritmo: BuscaBST(r, x)

Entrada: nó raiz r da BST, valor x

Saída: nó cuja chave é x , ou λ caso o valor x não esteja na BST

```
1 se  $r \neq \lambda$  então
2   | se  $x == r \rightarrow chave$  então
3   |   | retorne  $r$ 
4   | se  $x < r \rightarrow chave$  então
5   |   | retorne BuscaBST( $r \rightarrow esq, x$ )
6   |   | retorne BuscaBST( $r \rightarrow dir, x$ )
7 retorne  $\lambda$ 
```

Complexidade: proporcional à altura da BST - $O(h)$

Árvore binária de busca: buscar (iterativo)

Algoritmo: BuscaBST(r, x)

Entrada: nó raiz r da BST, valor x

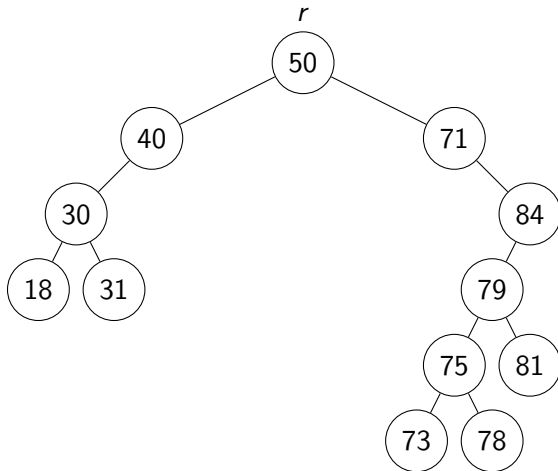
Saída: nó cuja chave é x , ou λ caso o valor x não esteja na BST

```
1 enquanto  $r \neq \lambda$  faça
2   se  $x == r \rightarrow chave$  então
3     |   retorne  $r$ 
4   se  $x < r \rightarrow chave$  então
5     |    $r = r \rightarrow esq$ 
6   senão
7     |    $r = r \rightarrow dir$ 
8 retorne  $\lambda$ 
```

Complexidade: proporcional à altura da BST - $O(h)$

Árvore binária de busca: incluir

- Exemplo: incluir novos valores nesta BST



Árvore binária de busca: incluir (recursivo)

Algoritmo: IncluiBST(r, v)

Entrada: nó raiz r da BST, nó v a ser incluído

Saída: raiz da árvore resultante

```
1 se  $r == \lambda$  então
2   |   retorne  $v$ 
3 se  $v \rightarrow chave \leq r \rightarrow chave$  então
4   |    $r \rightarrow esq = \text{IncluiBST}(r \rightarrow esq, v)$ 
5 senão
6   |    $r \rightarrow dir = \text{IncluiBST}(r \rightarrow dir, v)$ 
7 retorne  $r$ 
```

Complexidade: proporcional à altura da BST - $O(h)$

Árvore binária de busca: incluir (iterativo)

Algoritmo: IncluiBST(r, v)

Entrada: nó raiz r da BST, nó v a ser incluído

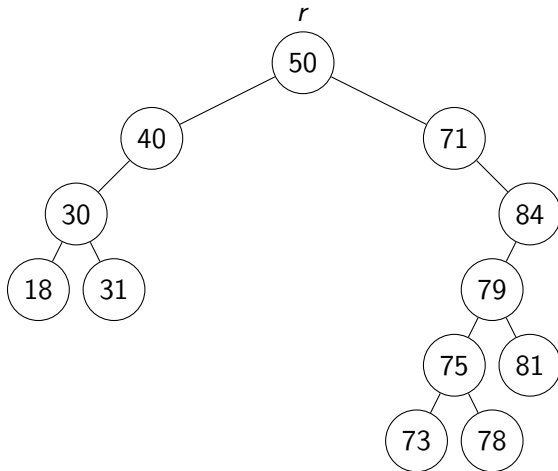
Saída: raiz da árvore resultante

```
1 se  $r == \lambda$  então
2   | retorne  $v$ 
3  $t = r$ 
4 criar novo ponteiro  $u$ 
5 enquanto  $r \neq \lambda$  faça
6   |  $u = r$ 
7   | se  $v \rightarrow chave \leq r \rightarrow chave$  então
8   |   |  $r = r \rightarrow esq$ 
9   | senão
10  |   |  $r = r \rightarrow dir$ 
11 se  $v \rightarrow chave \leq u \rightarrow chave$  então
12 |    $u \rightarrow esq = v$ 
13 senão
14 |    $u \rightarrow dir = v$ 
15 retorne  $t$ 
```

Complexidade: proporcional à altura da BST - $O(h)$

Árvore binária de busca: mínimo e máximo

- Exemplo: determinar o mínimo e o máximo nesta BST



Árvore binária de busca: mínimo (recursivo)

Algoritmo: $\text{MinimoBST}(r)$

Entrada: nó raiz r da BST

Saída: nó com a menor chave

```
1 se  $r \rightarrow \text{esq} \neq \lambda$  então
2   |   retorne  $\text{MinimoBST}(r \rightarrow \text{esq})$ 
3 retorne  $r$ 
```

Complexidade: proporcional à altura da BST - $O(h)$

Árvore binária de busca: mínimo (iterativo)

Algoritmo: MinimoBST(r)

Entrada: nó raiz r da BST

Saída: nó com a menor chave

```
1 enquanto  $r \rightarrow \text{esq} \neq \lambda$  faça
2   |    $r = r \rightarrow \text{esq}$ 
3 retorne  $r$ 
```

Complexidade: proporcional à altura da BST - $O(h)$

Árvore binária de busca: máximo (recursivo)

Algoritmo: $\text{MaximoBST}(r)$

Entrada: nó raiz r da BST

Saída: nó com a maior chave

```
1 se  $r \rightarrow \text{dir} \neq \lambda$  então
2   |   retorne  $\text{MaximoBST}(r \rightarrow \text{dir})$ 
3 retorne  $r$ 
```

Complexidade: proporcional à altura da BST - $O(h)$

Árvore binária de busca: máximo (iterativo)

Algoritmo: MaximoBST(r)

Entrada: nó raiz r da BST

Saída: nó com a maior chave

1 **enquanto** $r \rightarrow dir \neq \lambda$ **faça**

2 $r = r \rightarrow dir$

3 **retorne** r

Complexidade: proporcional à altura da BST - $O(h)$