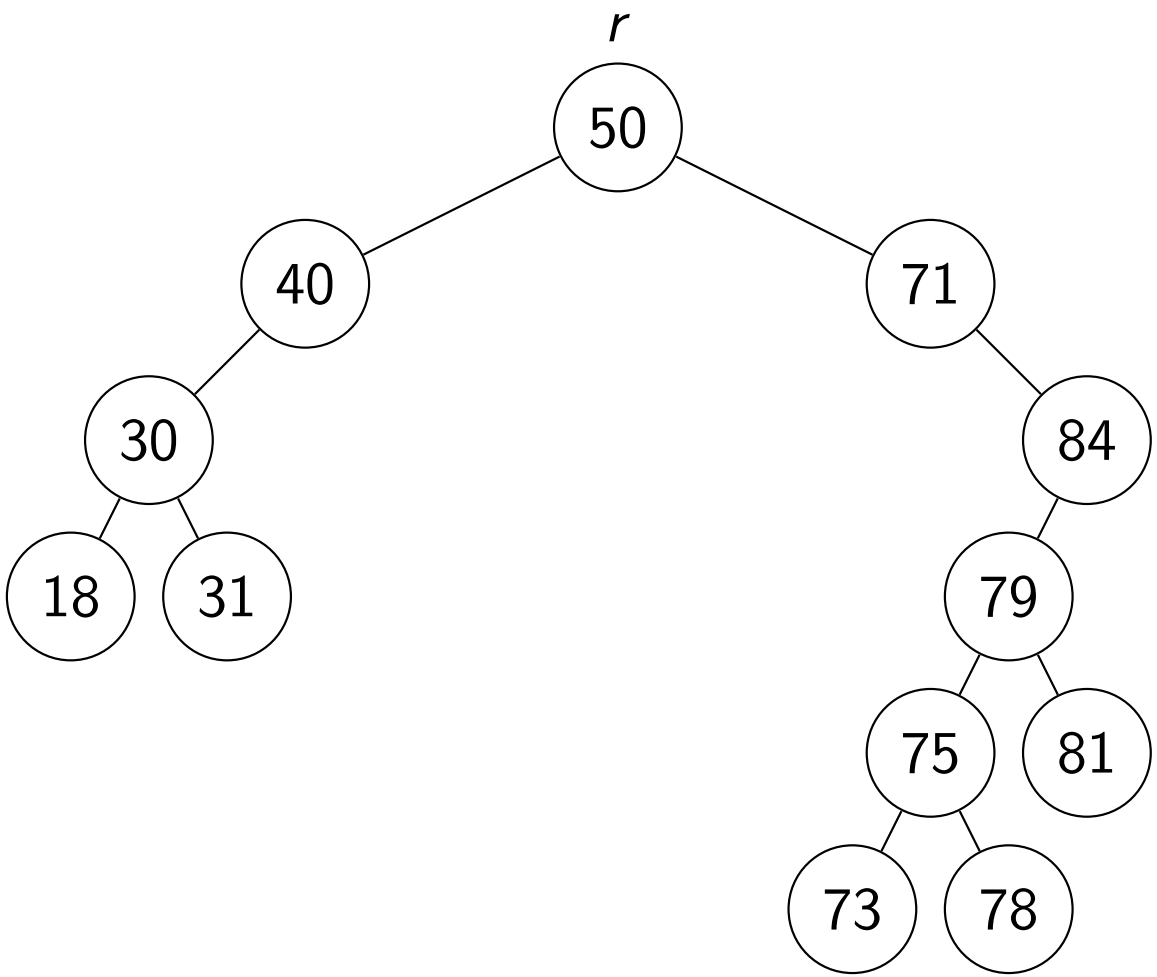
Estruturas de Dados - Árvore Binária de Busca

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
  - *pai*: 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()

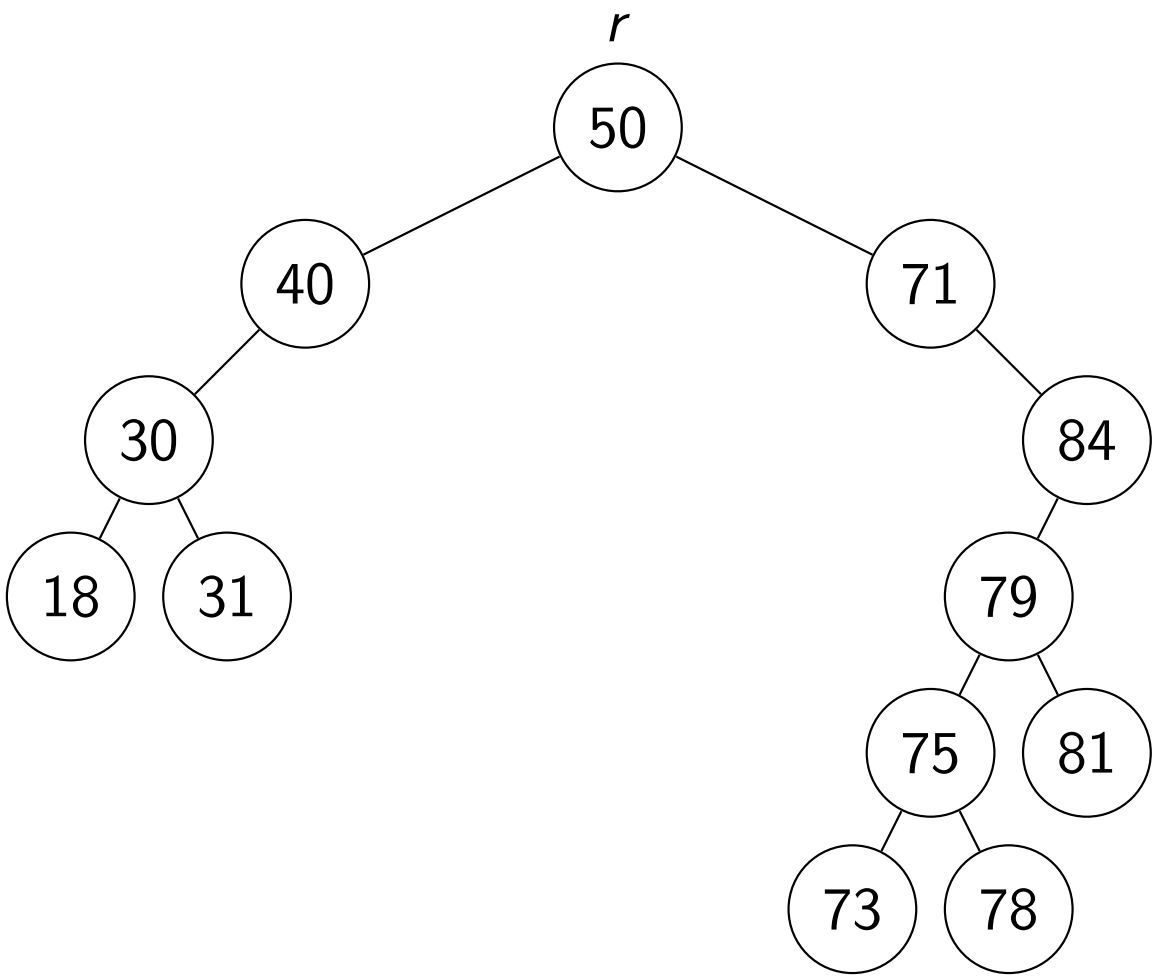
Saída: BST inicialmente vazia

1 **retorne** λ

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 ≠ λ então
2 |   se x == r→chave então
3 |   |   retorne r
4 |   se x < r→chave então
5 |   |   retorne BuscaBST(r→esq, x)
6 |   retorne BuscaBST(r→dir, x)
7 retorne λ
```

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

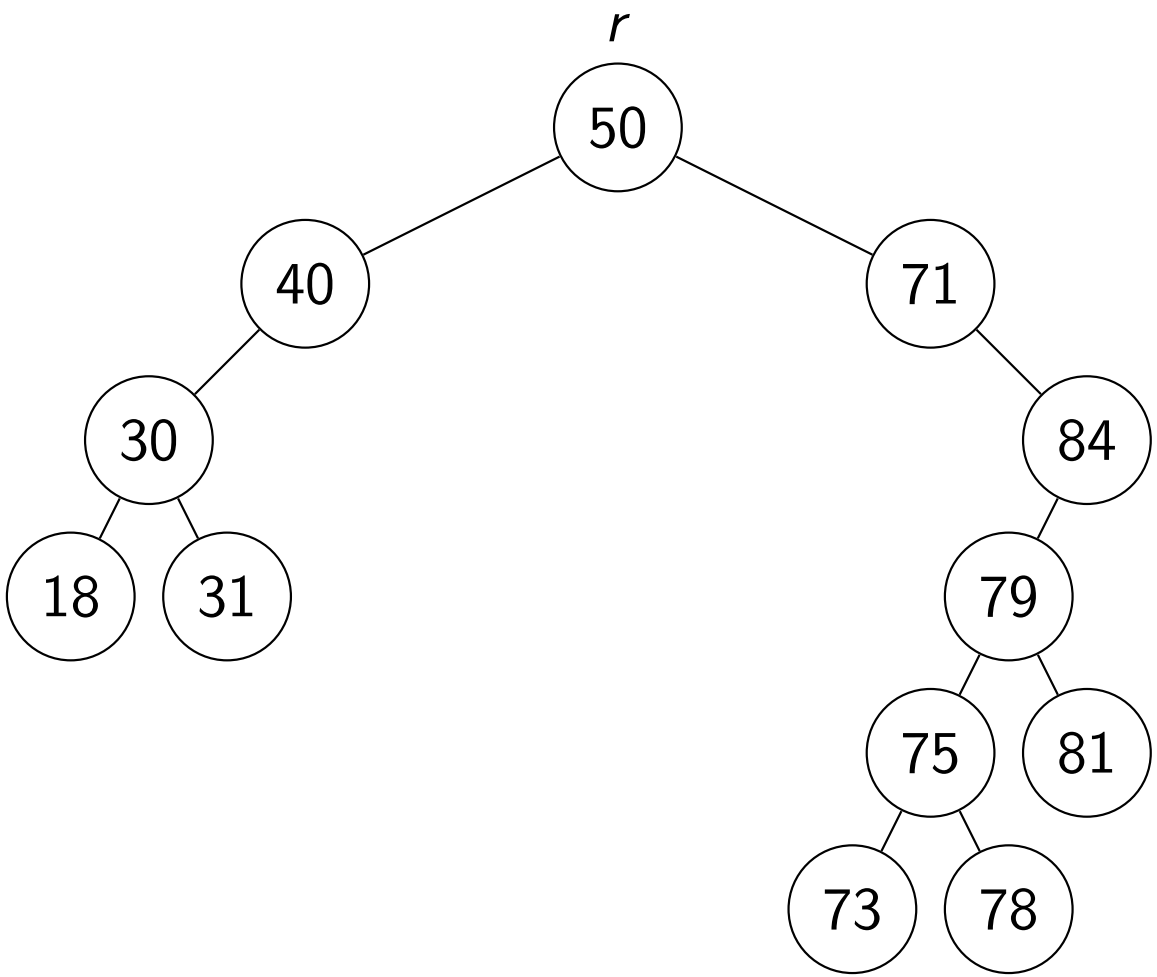
Saída: nó cuja chave é x, ou λ caso o valor x não esteja na BST

```
1 enquanto r ≠ λ faça
2 |   se x == r→chave então
3 |   |   retorne r
4 |   se x < r→chave então
5 |   |   r = r→esq
6 |   senão
7 |   |   r = r→dir
8 retorne λ
```

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, x)

Entrada: nó raiz r da BST, elemento x a ser incluído

Saída: raiz da árvore resultante

```
1  se r == λ então
2  |   criar novo nó v; v→chave = x; v→esq = λ; v→dir = λ; v→pai = λ
3  |   retorne v
4  se x ≤ r→chave então
5  |   r→esq = IncluiBST(r→esq, x)
6  |   r→esq→pai = r
7  senão
8  |   r→dir = IncluiBST(r→dir, x)
9  |   r→dir→pai = r
10 retorne r
```

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

Árvore binária de busca: incluir (iterativo)

Algoritmo: IncluiBST(r, v)

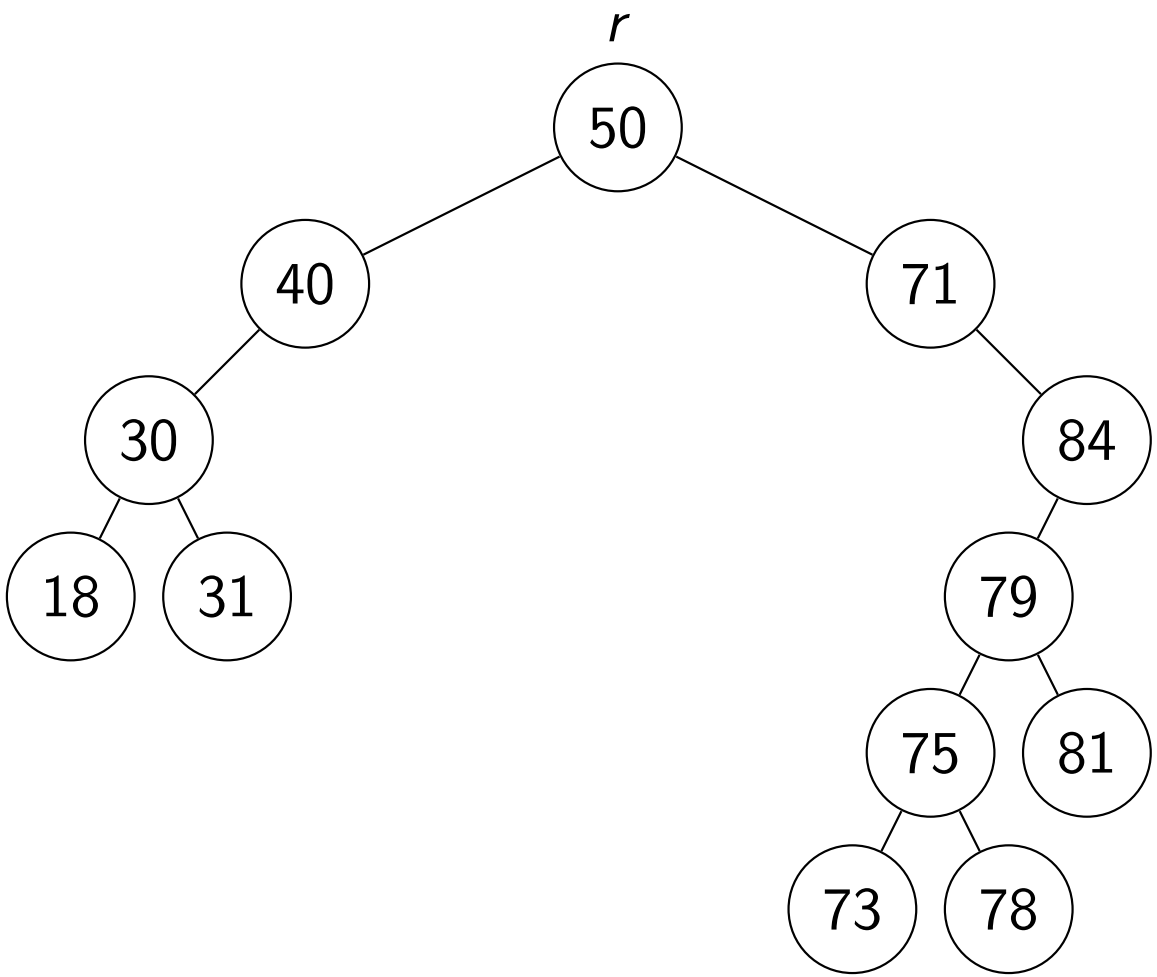
Entrada: nó raiz r da BST, elemento x a ser incluído
Saída: raiz da árvore resultante

```
1  criar novo nó v
2  v→chave = x; v→esq = λ; v→dir = λ; v→pai = λ
3  se r == λ então
4  |  retorne v
5  t = r
6  criar novo ponteiro u
7  enquanto r ≠ λ faça
8  |  u = r
9  |  se v→chave ≤ r→chave então
10 |  |  r = r→esq
11 |  senão
12 |  |  r = r→dir
13 se v→chave ≤ u→chave então
14 |  u→esq = v
15 senão
16 |  u→dir = v
17 v→pai = u
18 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: MinimoBST(r)

Entrada: nó raiz r da BST

Saída: nó com a menor chave

```
1 se r→esq ≠ λ então
2 |   retorne MinimoBST(r→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→esq ≠ λ faça
2 |   r = r→esq
3 retorne r
```

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

Árvore binária de busca: máximo (recursivo)

Algoritmo: MaximoBST(r)

Entrada: nó raiz r da BST

Saída: nó com a maior chave

```
1 se r→dir ≠ λ então
2 |   retorne MaximoBST(r→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→dir ≠ λ faça
2 |   r = r→dir
3 retorne r
```

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