Árvores de Busca

Algoritmos e Estruturas de Dados

Prof. Dr. Luciano Demétrio Santos Pacífico
{luciano.pacifico@ufrpe.br}

UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO

Conteúdo

- Introdução
- Árvores de Busca
- Árvores Binárias de Busca
- Busca e Inserção
- Remoção

Introdução

UNIVERSIDADE
FEDERAL RURAL
DE PERNAMBUCO

Estruturas Lineares

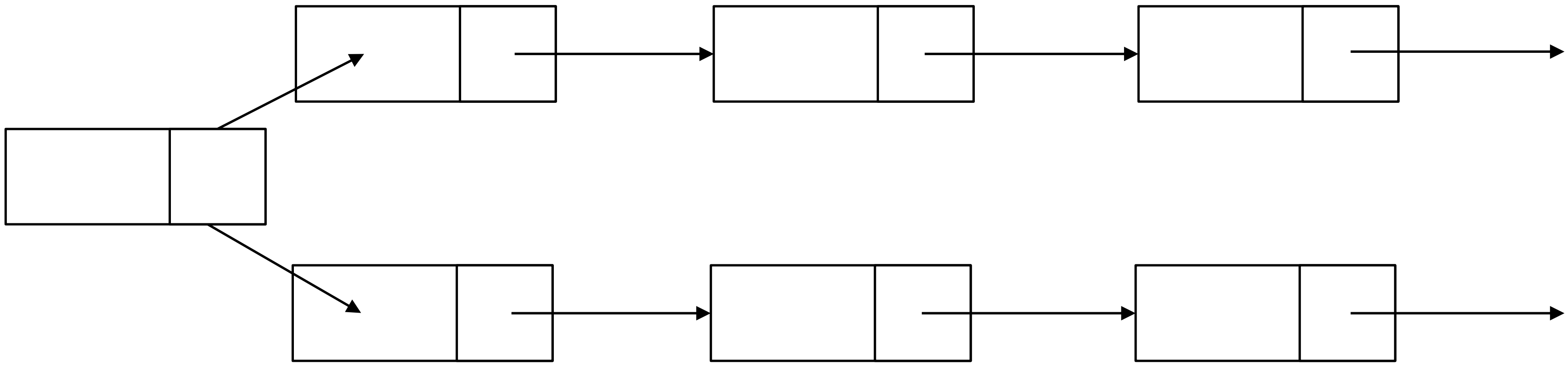
- Estruturas lineares, como listas ligadas, apresentam como um grande fator limitante ao seu uso o acesso puramente sequencial aos elementos da estrutura.

- Em diversas aplicações práticas, o uso de estruturas mais complexas faz-se necessário.

- Estruturas dinâmicas melhoradas podem ser facilmente encontradas.

Estruturas Não Lineares

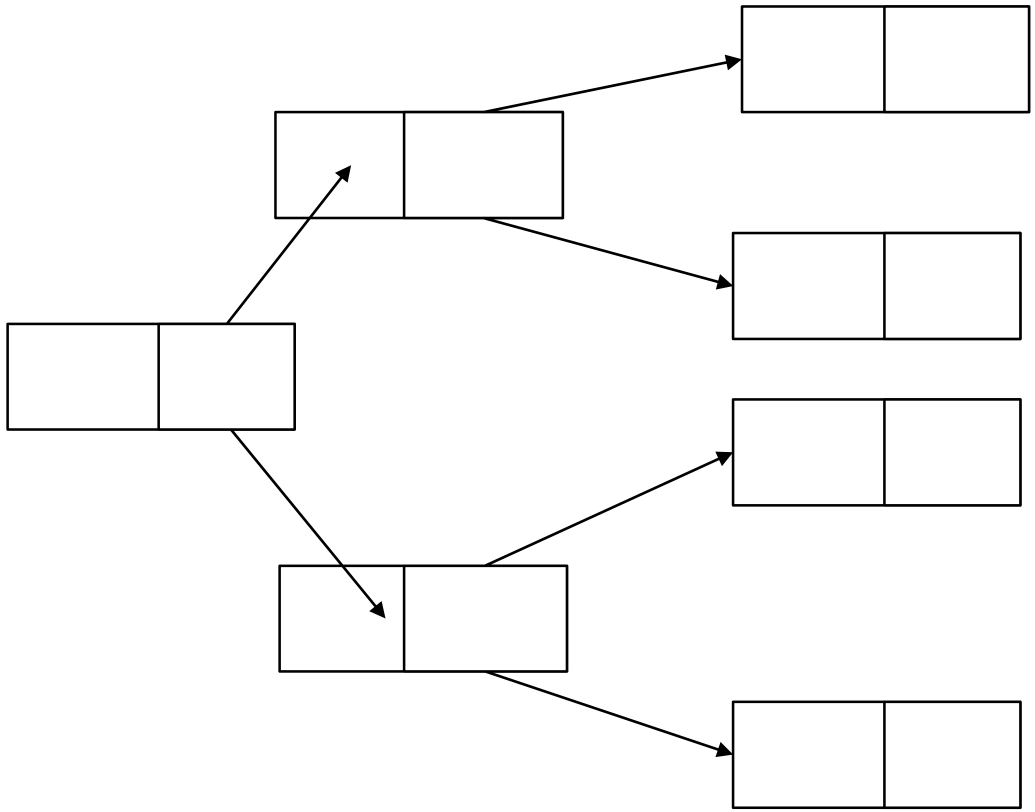
- Uma melhoria simples pode ser a divisão da lista em dois grupos:



- Porém esta solução apenas divide o problema na metade, não afetando seu custo assintótico.

Estruturas Não Lineares

- Outra solução seria a divisão sucessiva a cada elemento:



Estruturas Não Lineares

- As estruturas que fazem a divisão dos nós a cada elemento são conhecidas como **Árvores de Busca**.

- Veremos que, de modo geral, as árvores de busca são de fácil representação computacional, e além disso, são úteis para a solução de diversos problemas práticos.

Árvores de Busca

UNIVERSIDADE
FEDERAL RURAL
DE PERNAMBUCO

Árvores de Busca

- Uma **Árvore Enraizada** *T*, ou simplesmente **Árvore**, é um conjunto finito de elementos (*nós*) tais que:

  - $T = \emptyset$, e a árvore é dita vazia, ou

  - Existe um nó especial, *r*, chamado **raiz** de *T*; os nós restantes constituem um único conjunto vazio ou são divididos em $m \geq 1$ conjuntos disjuntos não vazios, as subárvores de *r*, ou simplesmente subárvores, cada qual, por sua vez, uma árvore.

Árvores de Busca

- A forma mais comum de representação de uma árvore e de seus elementos é através de um gráfico hierarquizado.

- Um conjunto de árvores é chamado de **Floresta**.

- Se v é um nó de T, a notação T_v indica a subárvore de T com raiz em v.

Árvores de Busca

- Seja *v* o nó raiz da subárvore T_v de *T*.

- Os nós $w_1, w_2, \dots, w_j$ das subárvores de T_v são chamados **filhos** de *v*, *v* é chamado **pai** dos nós $w_1, w_2, \dots, w_j$ e os nós $w_1, w_2, \dots, w_j$ são irmãos entre si.

- O **número de filhos de um nó** é chamado **grau de saída**, ou apenas **grau**, desse nó.

Árvores de Busca

- Se x pertence à subárvore de T_v, x é descendente de v, e v é ancestral de x.
- Nesse caso, se x é diferente de v, x é *descendente próprio* de v, e v é *ancestral próprio* de x.
- Um nó que não possui descendentes próprios é chamado de **folha**.
- Toda árvore com $n > 1$ nós possui no mínimo 1 e no máximo $n - 1$ folhas.
- Um nó não folha é dito **nó interior**.

Árvores de Busca

- Uma sequência de nós distintos $v_1, v_2, \dots, v_k$, tal que existe sempre entre nós consecutivos (v_1 e v_2, v_2 e v_3, …, v_{k-1} e v_k) a relação "é filho de" ou "é pai de", é denominado **caminho da árvore**.

- O valor k-1 é dito o **tamanho do caminho**.

- Diz-se que v_1 alcança v_k, e vice-versa.

- O **nível** de um nó é definido como **o número de nós no caminho entre o nó em questão e a raiz da árvore, contando o nó em questão**.

Árvores de Busca

- O nível da raiz é igual a 1.

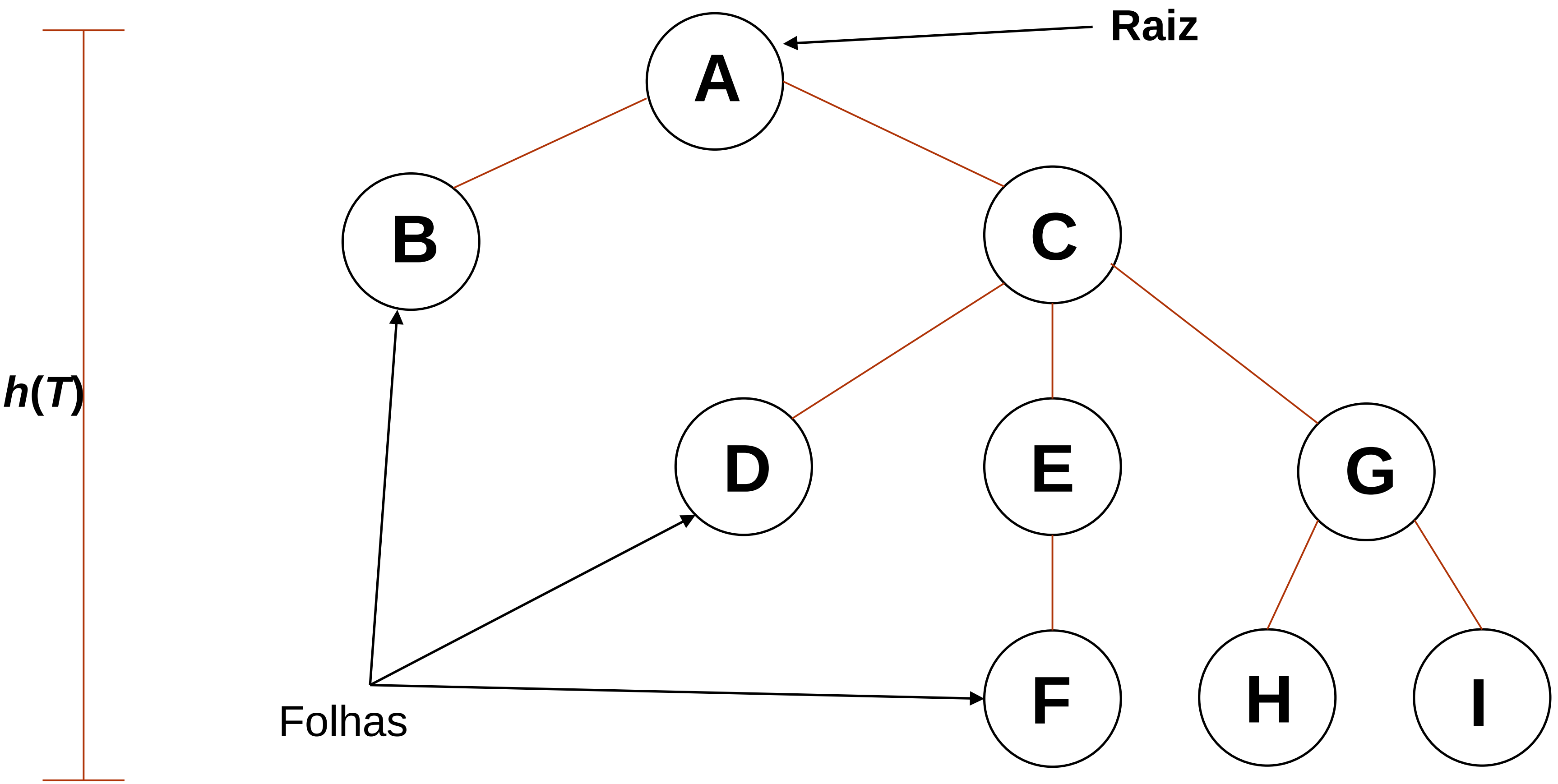
- A **altura** de um nó é **o número de nós do maior caminho do nó em questão até um de seus descendentes**.

- As **folhas** de uma árvore têm **altura 1**.

- A altura $h(T)$ da árvore T é igual ao nível máximo de seus nós.

- Uma *árvore ordenada* é aquela na qual os filhos de cada nó estão ordenados.

Árvores de Busca



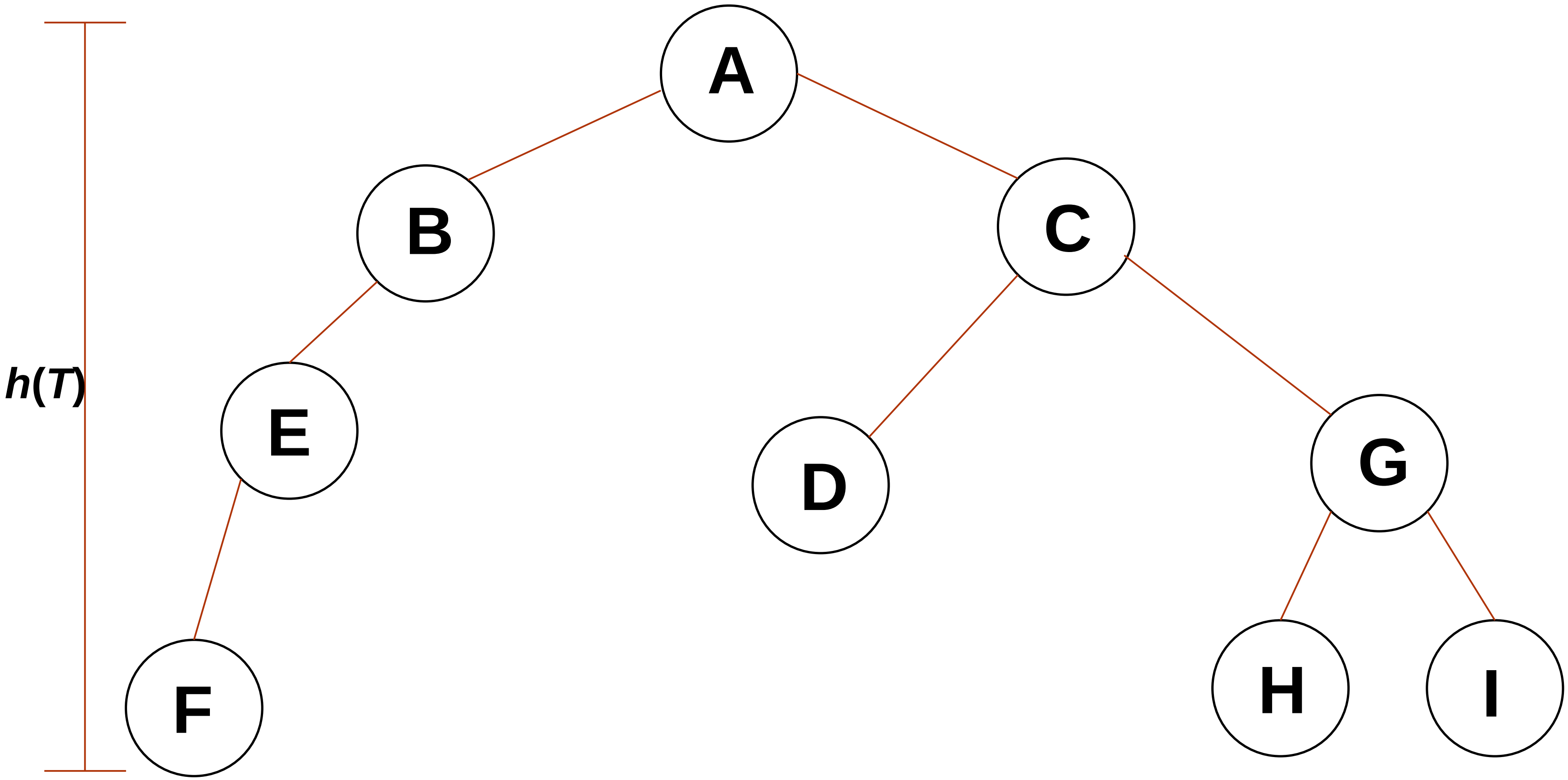
Árvores Binárias de Busca

UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO

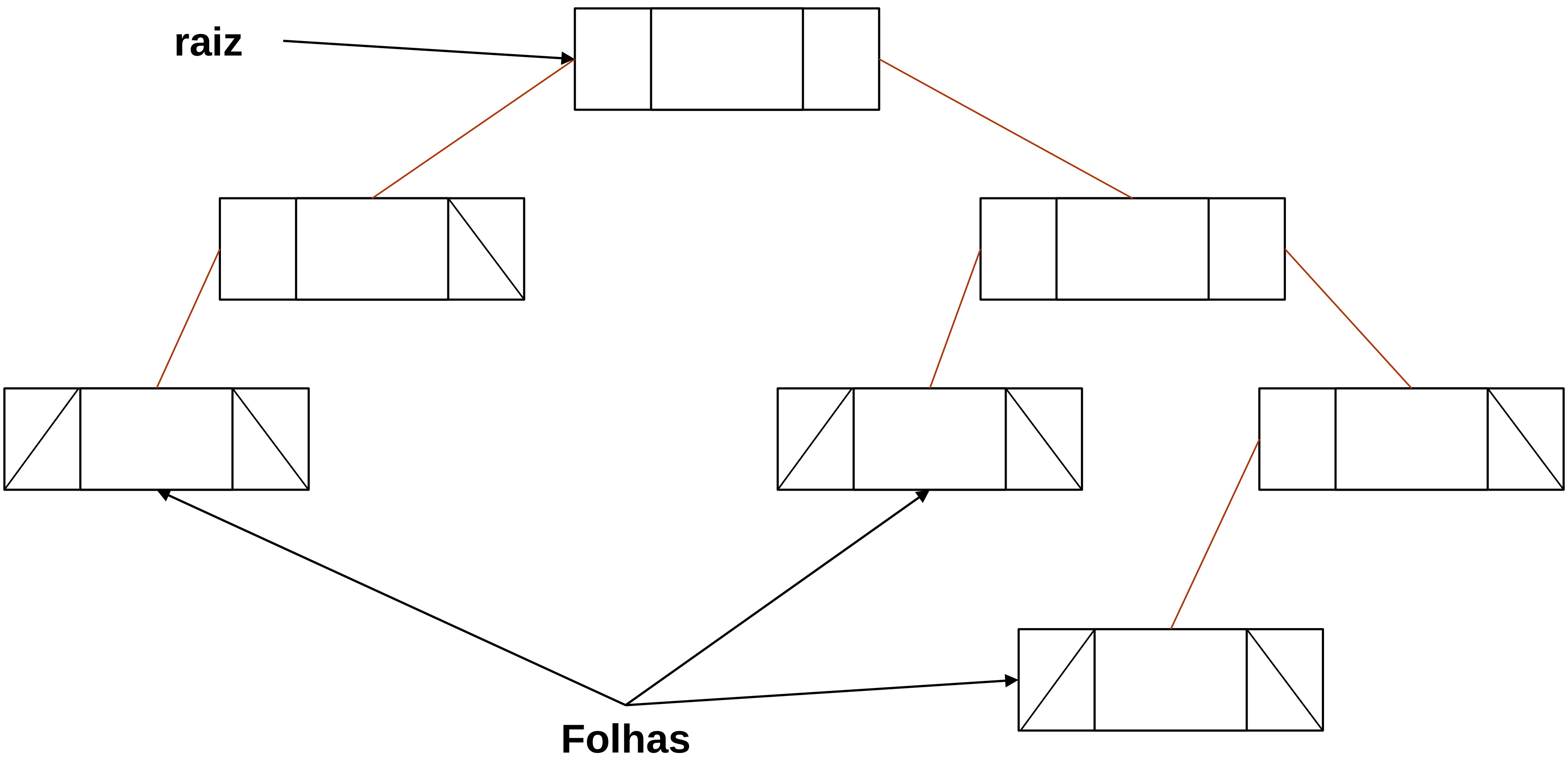
Árvores Binárias de Busca

- Dentre as árvores de busca, as *binárias* são as mais comuns.
- Uma árvore binária *T* é um conjunto finito de nós tais que:
  - *T* = Ø, e a árvore é dita vazia, ou
  - Existe um nó especial, *r*, chamado *raiz* de *T*; os nós restantes podem ser divididos em dois subconjuntos disjuntos, T_r^E e T_r^D, que são as subárvores esquerda e direita de *r*, respectivamente, e que também são árvores binárias.

Árvores Binárias de Busca - Representação



Árvore Binária de Busca – Alocação Dinâmica



Árvores Binárias de Busca

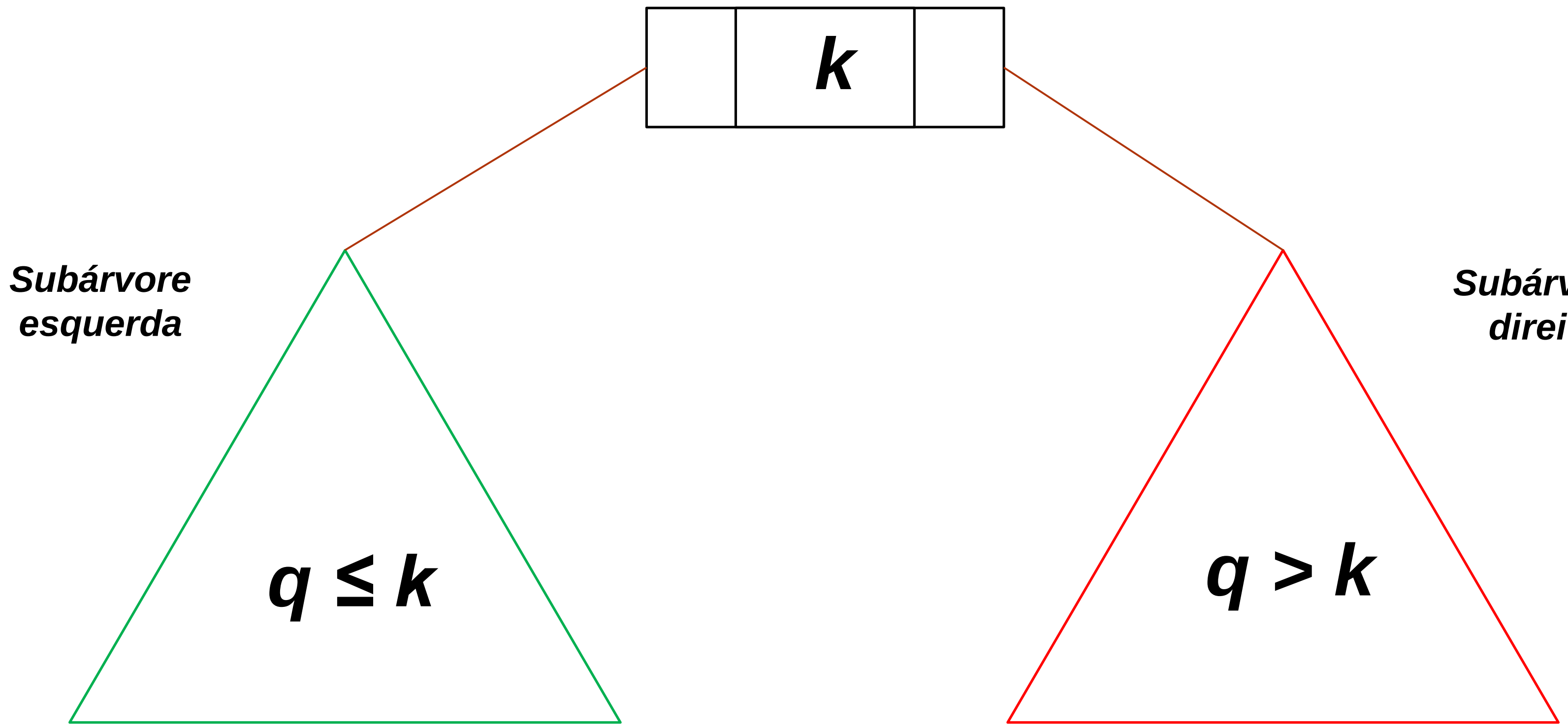
- Em uma árvore binária de busca, os elementos são organizados de tal forma que:

  - Todos os **elementos na subárvore da esquerda** de cada nó *k* têm valor de chave **menor ou igual** à chave do nó *k*;

  - Todos os **elementos na subárvore da direita** de cada nó *k* têm valor de chave **maior** que a chave do nó *k*;

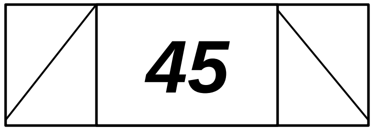
Árvores Binárias de Busca

- Os elementos são organizados de forma que:



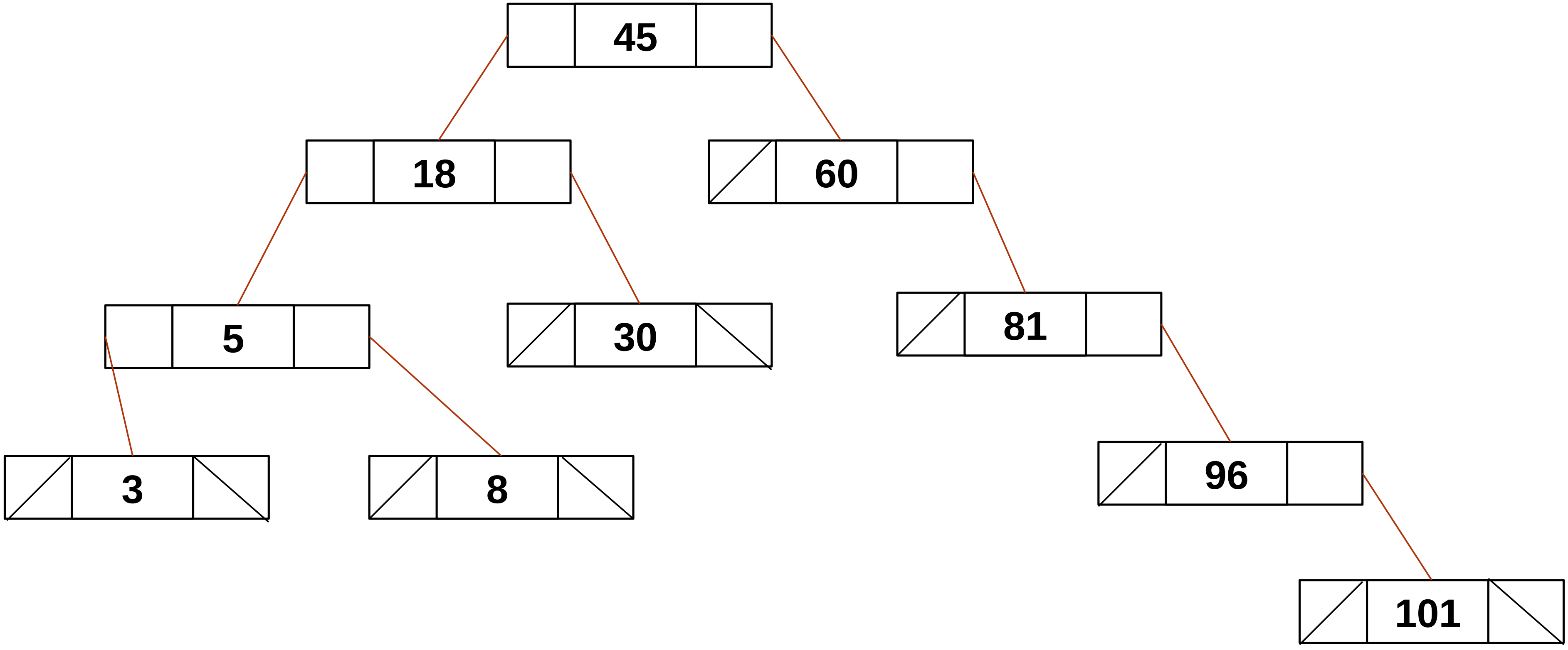
Exemplo

- Inserir: 45, 18, 30, 60, 81, 96, 101, 5, 8, 3



Exemplo

- Inserir: 45, 18, 30, 60, 81, 96, 101, 5, 8, 3



Busca e Inserção

UNIVERSIDADE
FEDERAL RURAL
DE PERNAMBUCO

Árvores Binárias - Estruturas Básicas

- Para os algoritmos exemplificados, as Estruturas de Dados básicas necessárias são representadas abaixo.

```
1. registro NoArvoreBin
2.      chave:inteiro,
3.      pai:NoArvoreBin,
4.      esquerda:NoArvoreBin,
5.      direita:NoArvoreBin
```

```
1. registro ArvoreBin
2.      raiz:NoArvoreBin
```

Percursos em Árvores Binárias

- Um **percurso** pode ser entendido como uma visita simétrica a cada nó da árvore.

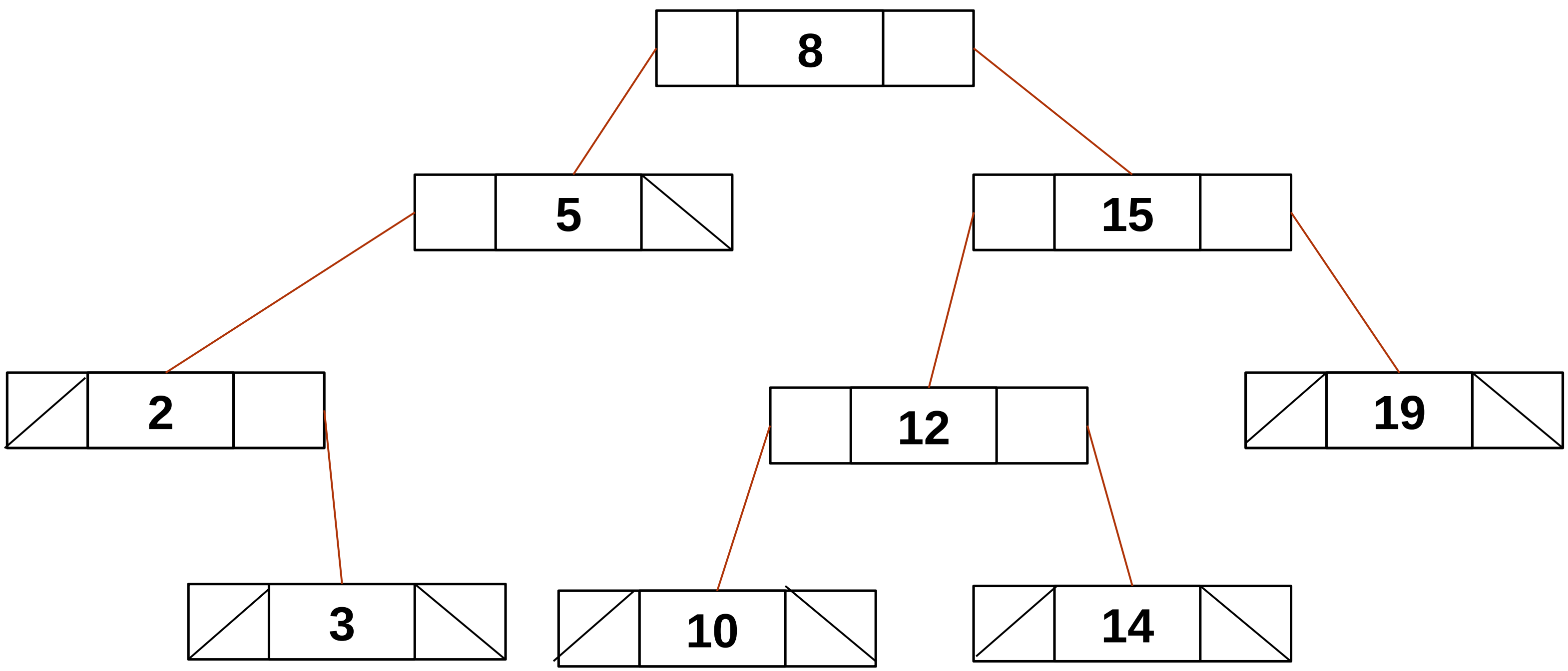
- A visita a um nó pode ser vista como uma operação que faça uso da informação contida no mesmo para algum fim.

- Existem três tipos básicos de percursos em uma árvore binária:

  - Percurso em pré-ordem;
  - Percurso em ordem;
  - Percurso em pós-ordem.

Percursos em Árvores Binárias

- Árvore Binária exemplo:



Percursos em Árvores Binárias

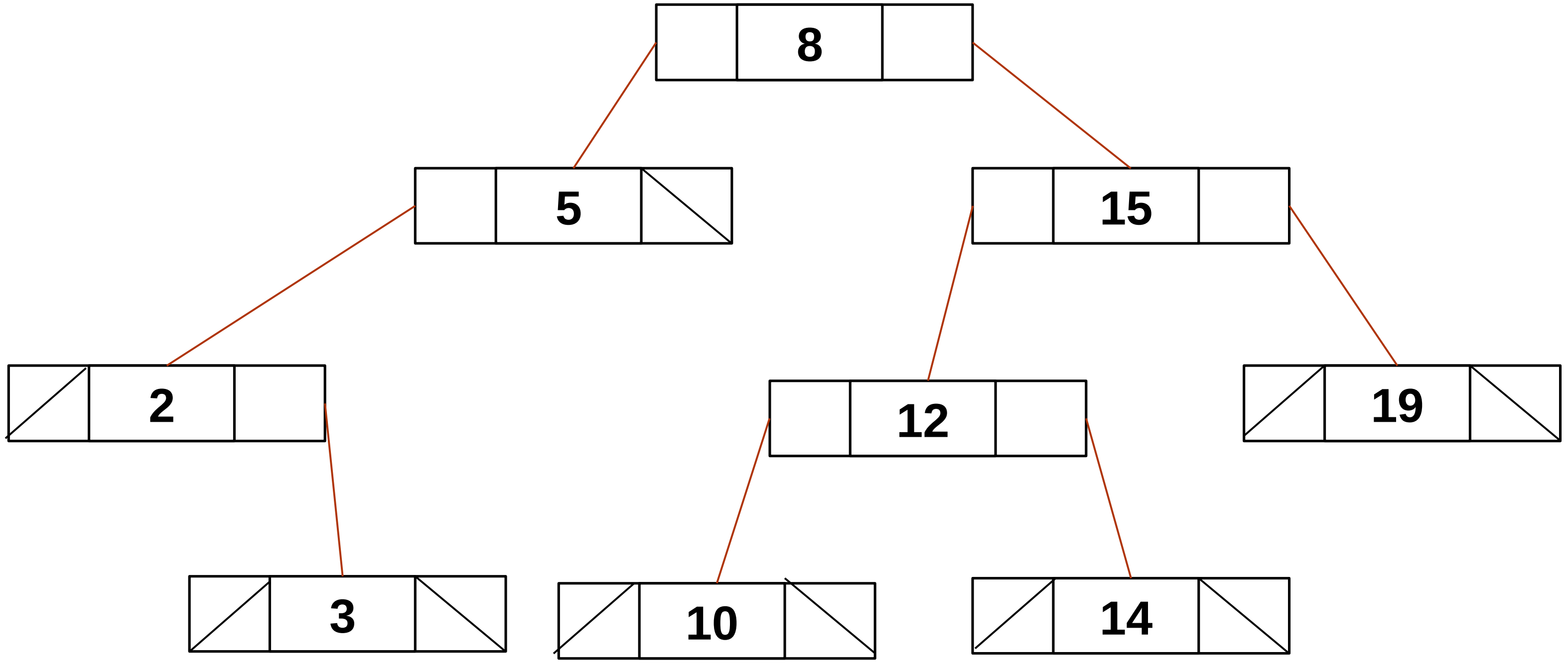
- Para todos os percursos que serão apresentados, a chamada inicial ao procedimento deve ser feita pelo fornecimento do **nó raiz da árvore**.

- Por questão de otimização, a checagem da nulidade do nó raiz deve ser feita fora da chamada do procedimento do percurso.

```
//pt -> é um NoArvoreBin
procedimento executarPercursoPreOrdem(pt)
    imprimir(pt.chave)
    se pt.esquerda != NIL
        executarPercursoPreOrdem(pt.esquerda)
    se pt.direita != NIL
        executarPercursoPreOrdem(pt.direita)
```

Percurso em Pré-Ordem

- Na pré-ordem, raiz das subárvores é visitada primeiro.
- Resultado: **8 5 2 3 15 12 10 14 19**



Percursos em Árvores Binárias

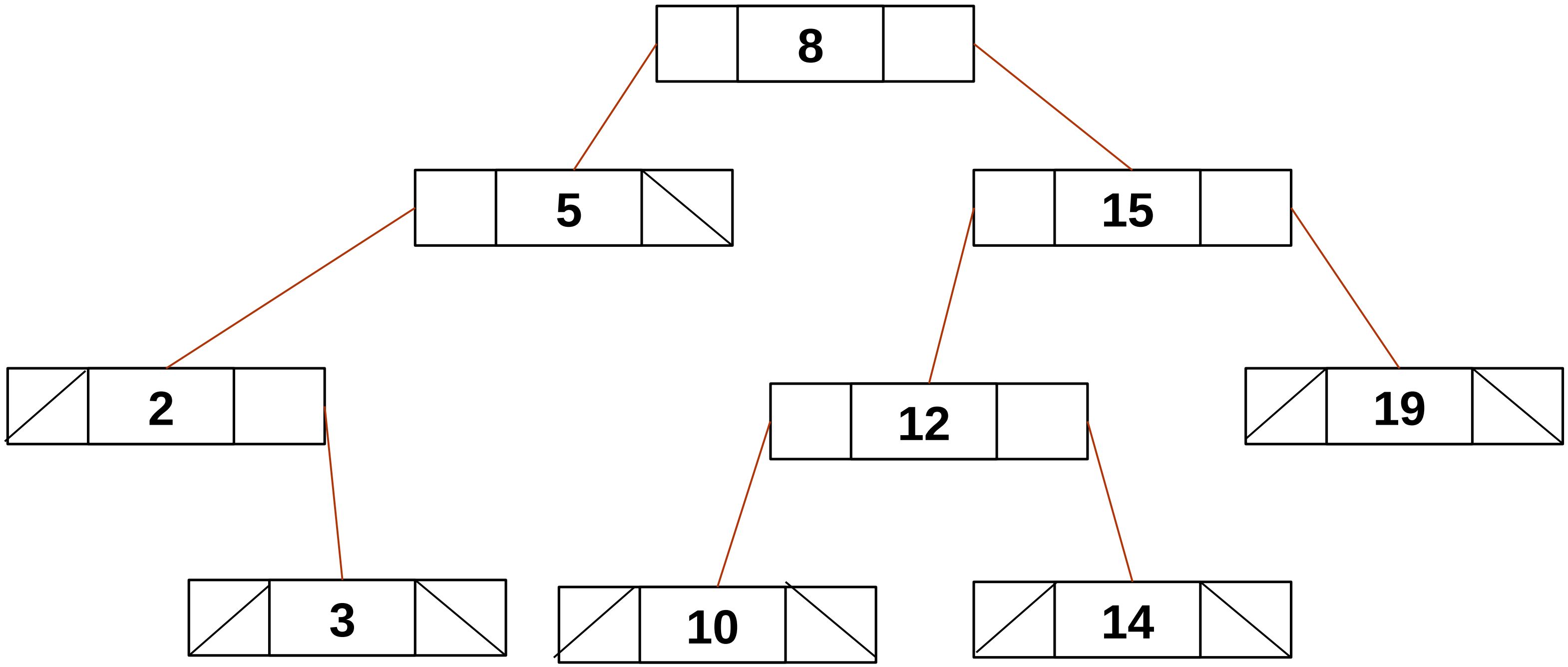
- No percurso em ordem, a raiz de cada subárvore é visitada entre a visita aos seus filhos.

- Esse procedimento pode ser usado para a visita dos nós da árvore **em ordem crescente**.

```
//pt -> é um NoArvoreBin
procedimento executarPercursoEmOrdem(pt)
    se pt.esquerda != NIL
        executarPercursoEmOrdem(pt.esquerda)
    imprimir(pt.chave)
    se pt.direita != NIL
        executarPercursoEmOrdem(pt.direita)
```

Percurso em Ordem

- Resultado: **2 3 5 8 10 12 14 15 19**



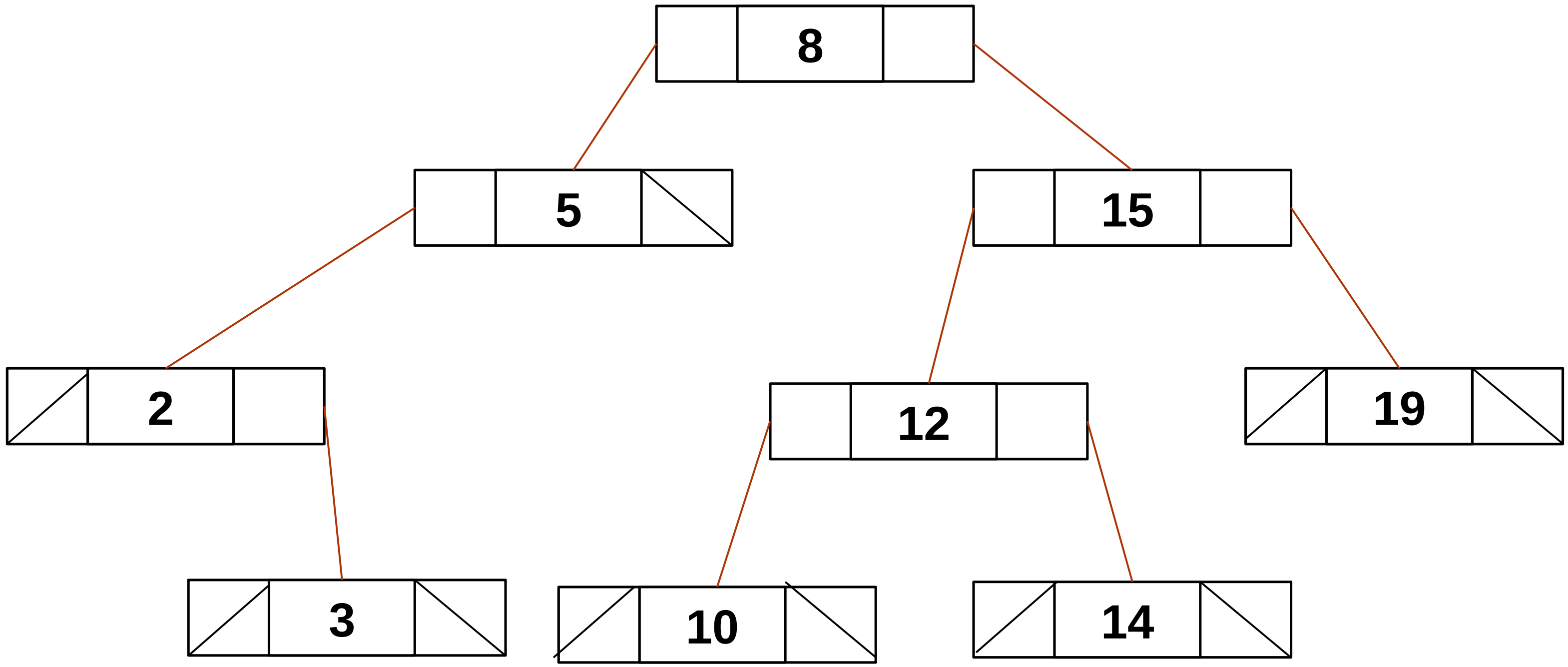
Percursos em Árvores Binárias

- No percurso em pós-ordem, a visita à raiz da subárvore ocorrerá apenas após a visita aos seus filhos.

```
//pt -> é um NoArvoreBin
procedimento executarPercursoPosOrdem(pt)
    se pt.esquerda != NIL
        executarPercursoPosOrdem(pt.esquerda)
    se pt.direita != NIL
        executarPercursoPosOrdem(pt.direita)
    imprimir(pt.chave)
```

Percurso em Pós-Ordem

- Resultado: **3 2 5 10 14 12 19 15 8**



Busca em Árvores Binárias

- Embora os procedimentos de percurso tenham sido apresentados em forma **recursiva**, podemos facilmente adaptar um algoritmo recursivo em um algoritmo **iterativo**.

- Exemplificaremos esse processo com os procedimentos de busca em Árvores Binárias.

- Tantos os procedimentos de percurso, quanto os de busca, inserção e remoção, são iniciados pela raiz da árvore na qual deseja-se executar tal procedimento.
  - A Árvore Binária não mantém o controle do número exato de nós que estão contidos na mesma (como todas as Estruturas de Dados Dinâmicas - Encadeadas), tendo informação apenas sobre o nó raiz.

Busca em Árvores Binárias

```
//x -> chave do nó buscado
//pt -> é um NoArvoreBin
procedimento buscarArvoreBinRec(x, pt)
    se (pt == NIL) ou (pt.chave == x)
        retorne pt
    senão se x < pt.chave
        retorne buscarArvoreBinRec(x, pt.esquerda)
    senão
        retorne buscarArvoreBinRec(x, pt.direita)
```

```
procedimento buscarArvoreBinIter(x, pt)
    enquanto (pt != NIL) e (pt.chave != x)
        se x < pt.chave
            pt = pt.esquerda
        senão
            pt = pt.direita
    retorne pt
```

Inserção em Árvores Binárias

```
//X -> é o NoArvoreBin a ser inserido
//T -> é a árvore (ArvoreBin)
procedimento inserirArvoreBin(X, T)
    pai = NIL
    pt = T.raiz
    enquanto pt != NIL  //encontra a posição do nó X
        pai = pt
        se X.chave <= pt.chave
            pt = pt.esquerda
        senão
            pt = pt.direita
    X.pai = pai
    se pai == NIL //árvore estava vazia, então X é a raiz
        T.raiz = X
    senão se X.chave <= pai.chave
        pai.esquerda = X
    senão
        pai.direita = X
```

Custo Computacional

- Os procedimentos de percurso (em pré-ordem, ordem e pós-ordem) possuem custo computacional da ordem de $\theta(n)$, pois todos os nós precisarão ser percorridos ao menos uma vez.

- Os procedimentos de inserção e busca possuem complexidade da ordem de $O(h(T))$, ou seja, $O(\log_2 n)$, onde n é o número de nós na árvore, quando a árvore é **balanceada**.

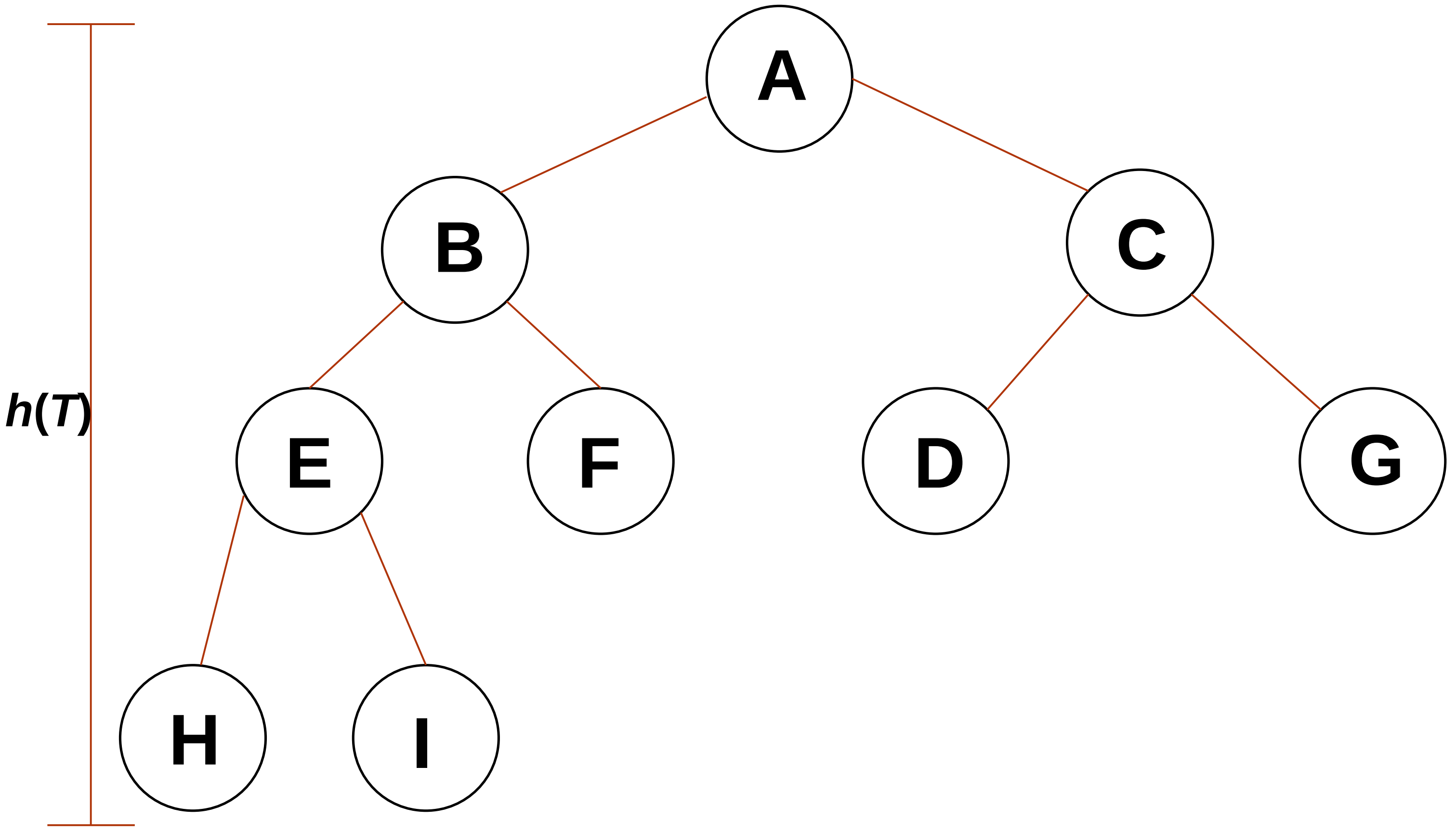
Custo Computacional

- Porém, dependendo da **ordem de inserção dos elementos na árvore binária de busca**, a mesma pode **degenerar**, tornando-se uma estrutura próxima de uma **lista linear**.

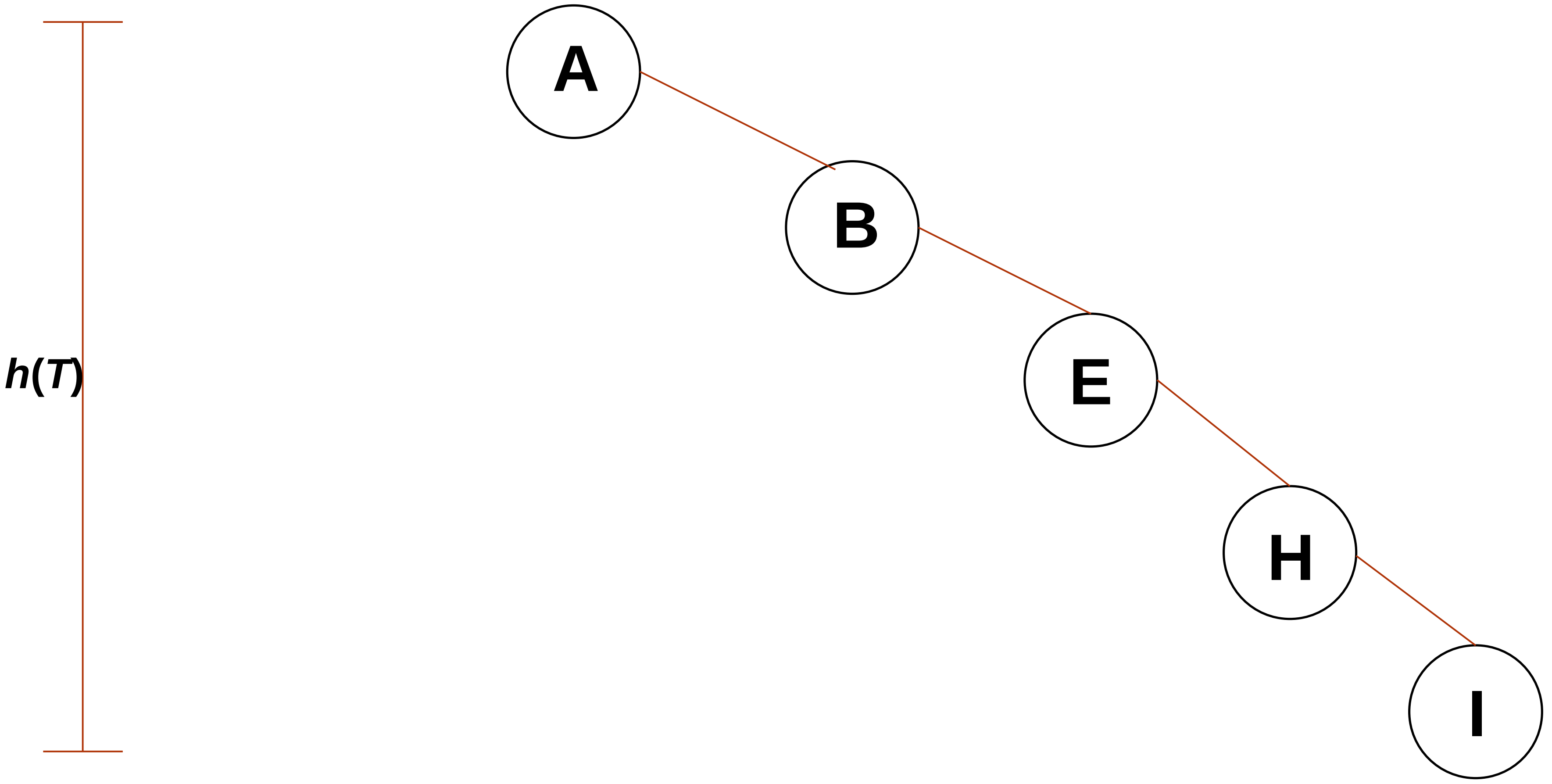
  - Se nós forem inseridos em ordem decrescente de chaves, cada nó estaria à esquerda de seu nó-pai;

  - Se nós forem inseridos em ordem crescente, cada nós estará à direita de seu nó-pai.

Árvores Binárias de Busca - Representação



Árvores Binárias de Busca - Representação



Remoção

Remoção – Nó com Zero ou Um Filho

- A estratégia geral para a remoção de um nó em uma árvore binária *T* possui três casos básicos.

- Remoção de um nó *x* que não possui filhos (folha): basta apenas modificar o nó-pai de *x* substituindo *x* por **`NIL`**.

- Remoção de um nó *x* com apenas um filho: fazemos com que esse filho de *x* substitua seu lugar em relação ao nó-pai de *x*.

Remoção - Nó com Dois Filhos

- O sucessor de *x* (que deve estar em sua subárvore direita) deve ser usado para assumir a posição de *x* em relação ao seu nó-pai.

- O restante da subárvore direita de *x* torna-se a subárvore direita de seu nó sucessor, e a subárvore esquerda de *x* torna-se a nova subárvore esquerda de seu sucessor.

- A análise é diferenciada levando em consideração quando o sucessor é o filho direto de *x* ou não.

- OBS.: Para o caso de nossa **remoção**, o sucessor **obrigatoriamente** estará na **subárvore direita** de x, pois chamaremos o procedimento do sucessor **apenas se x tiver dois filhos**!

Procedimentos Úteis

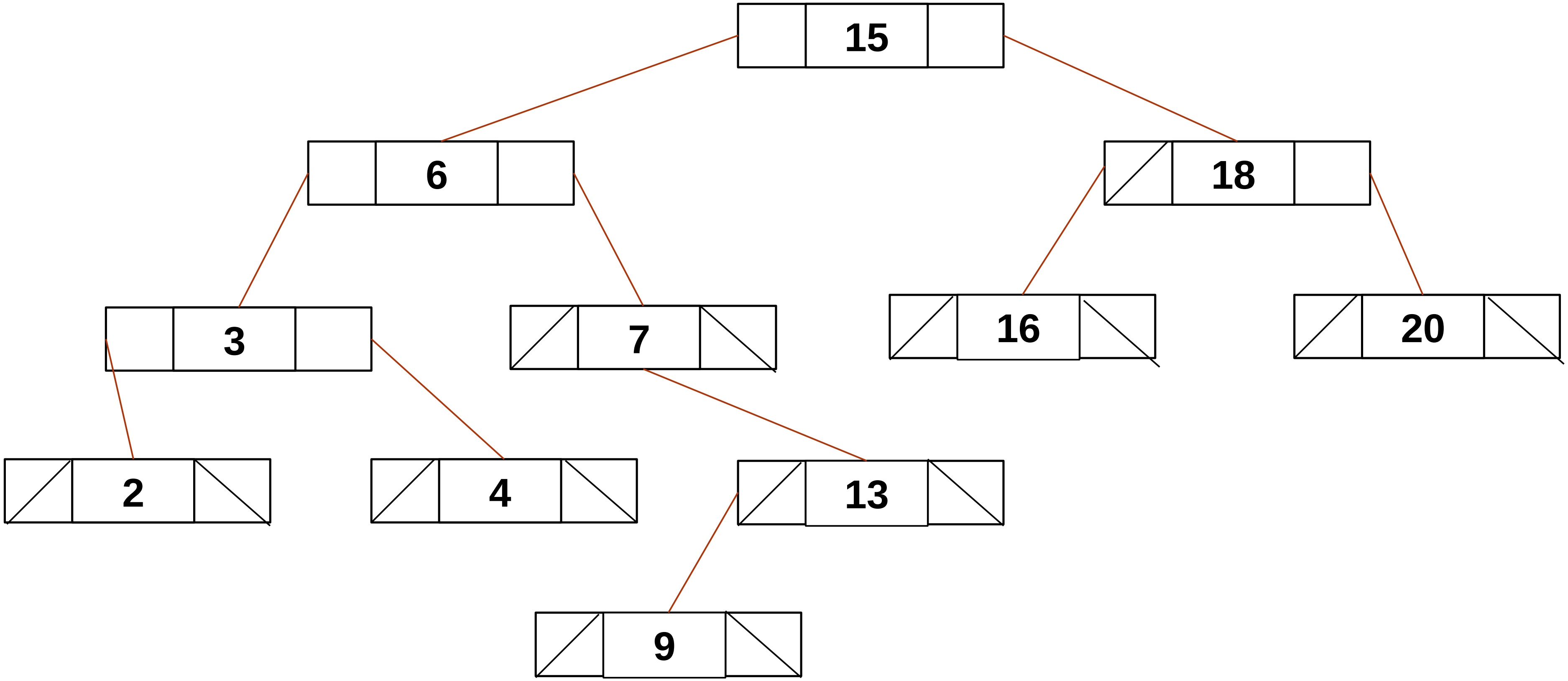
```
//pt -> é um NoArvoreBin – checar nulidade antes
procedimento encontrarMinimo(pt)
    enquanto pt.esquerda != NIL
        pt = pt.esquerda
    retorne pt
```

```
procedimento encontrarMaximo(pt)
    enquanto pt.direita != NIL
        pt = pt.direita
    retorne pt
```

```
procedimento encontrarSucessor(pt)
    se pt.direita != NIL
        retorne encontrarMinimo(pt.direita)
    senão
        pai = pt.pai
        enquanto (pai != NIL) e (pt.chave == pai.direita.chave)
            pt = pai
            pai = pai.pai
        retorne pai
```

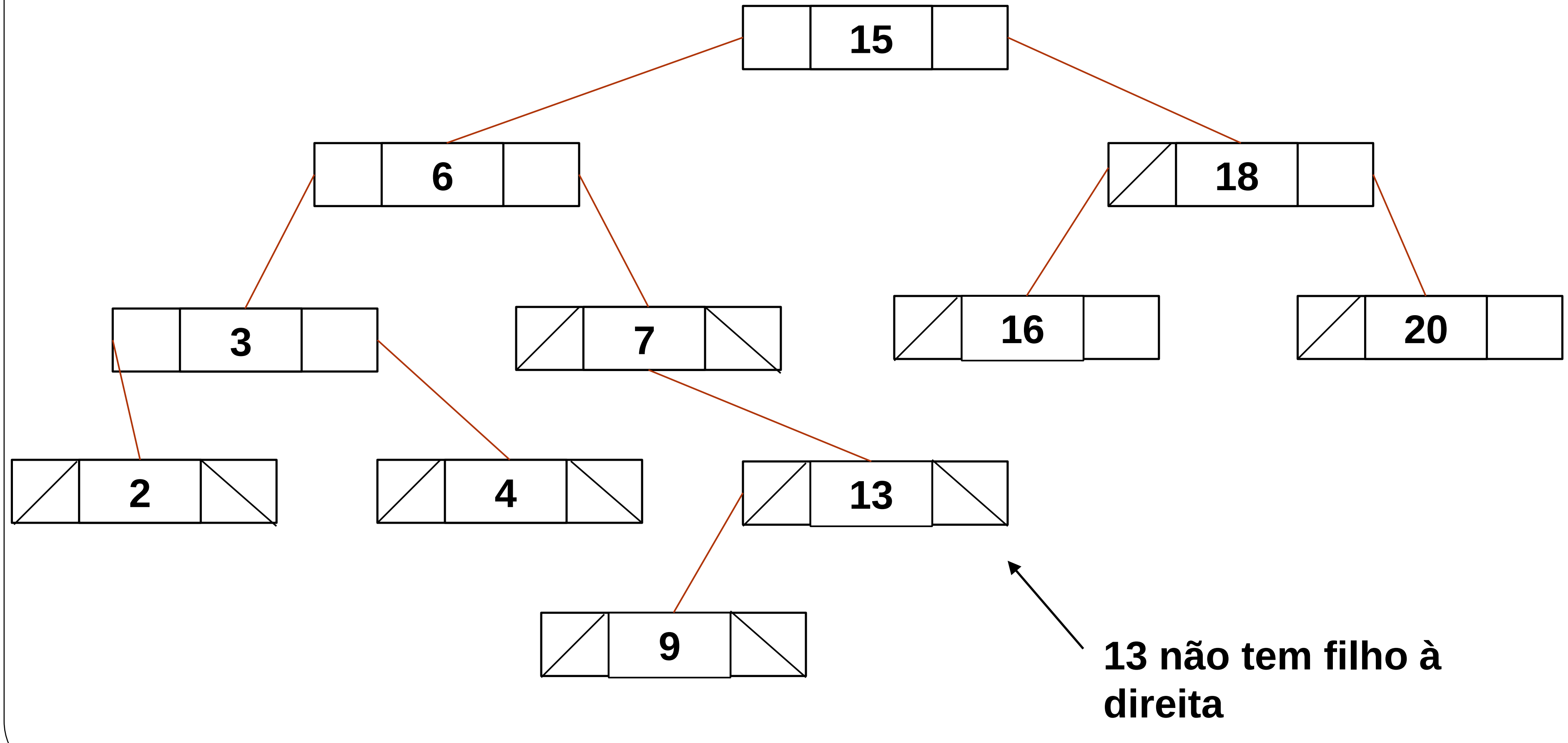
Exemplo

- Sucessor do nó 13:



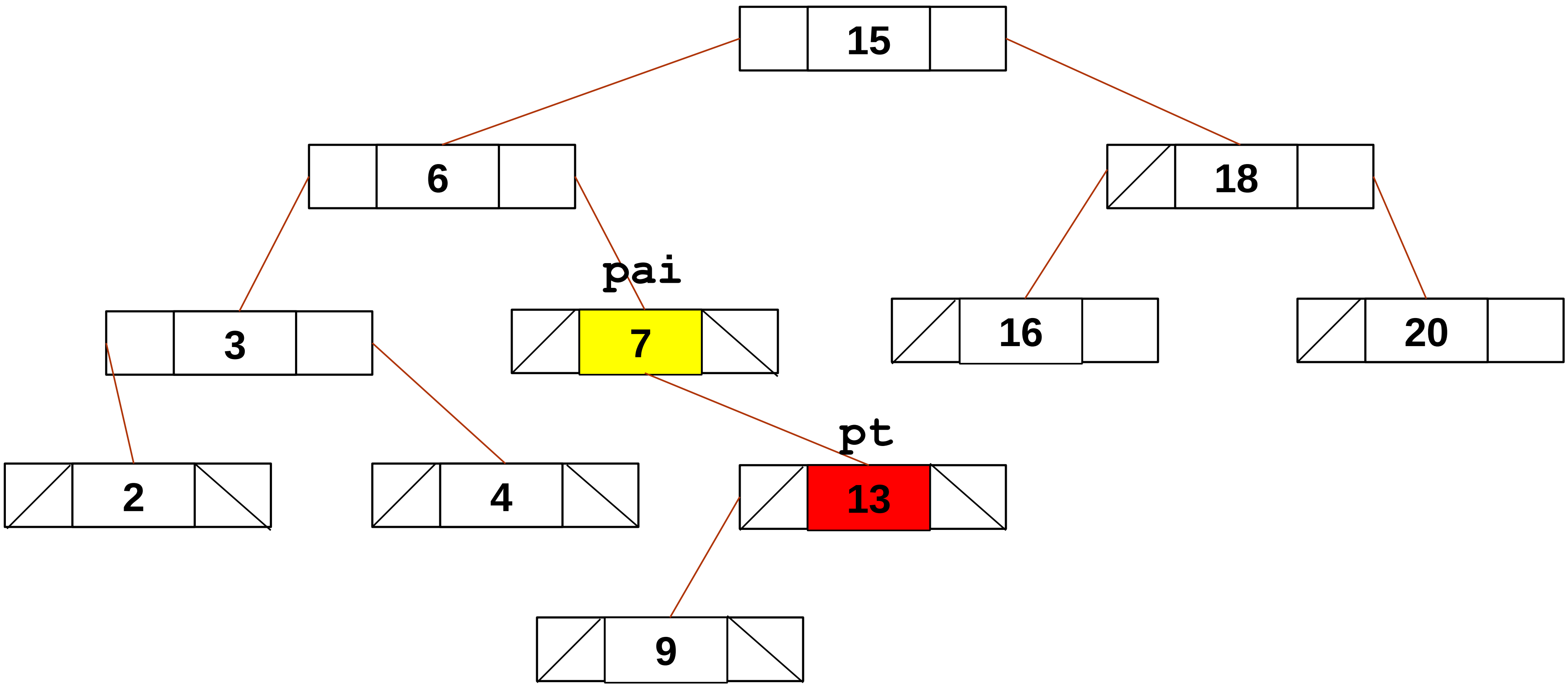
Exemplo

- Sucessor do nó 13:



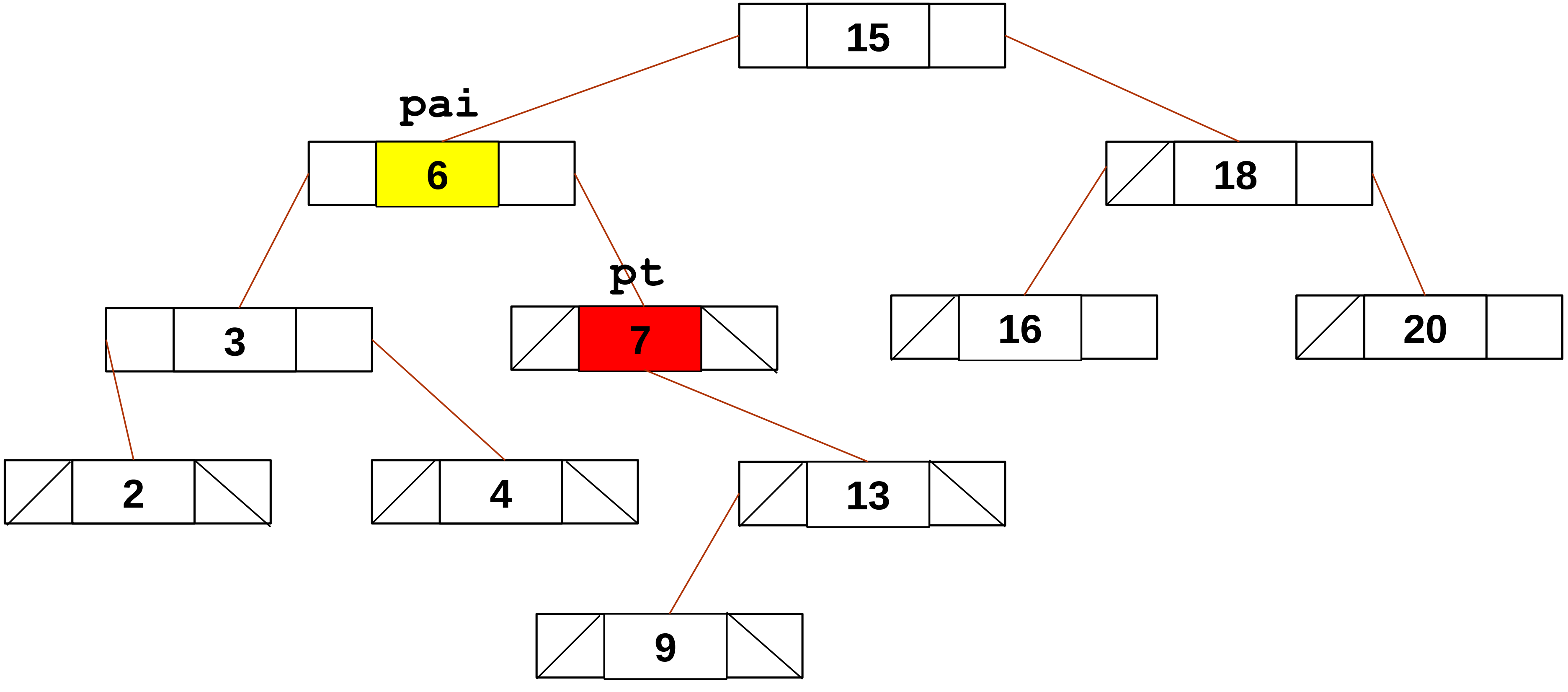
Exemplo

- Sucessor do nó 13:



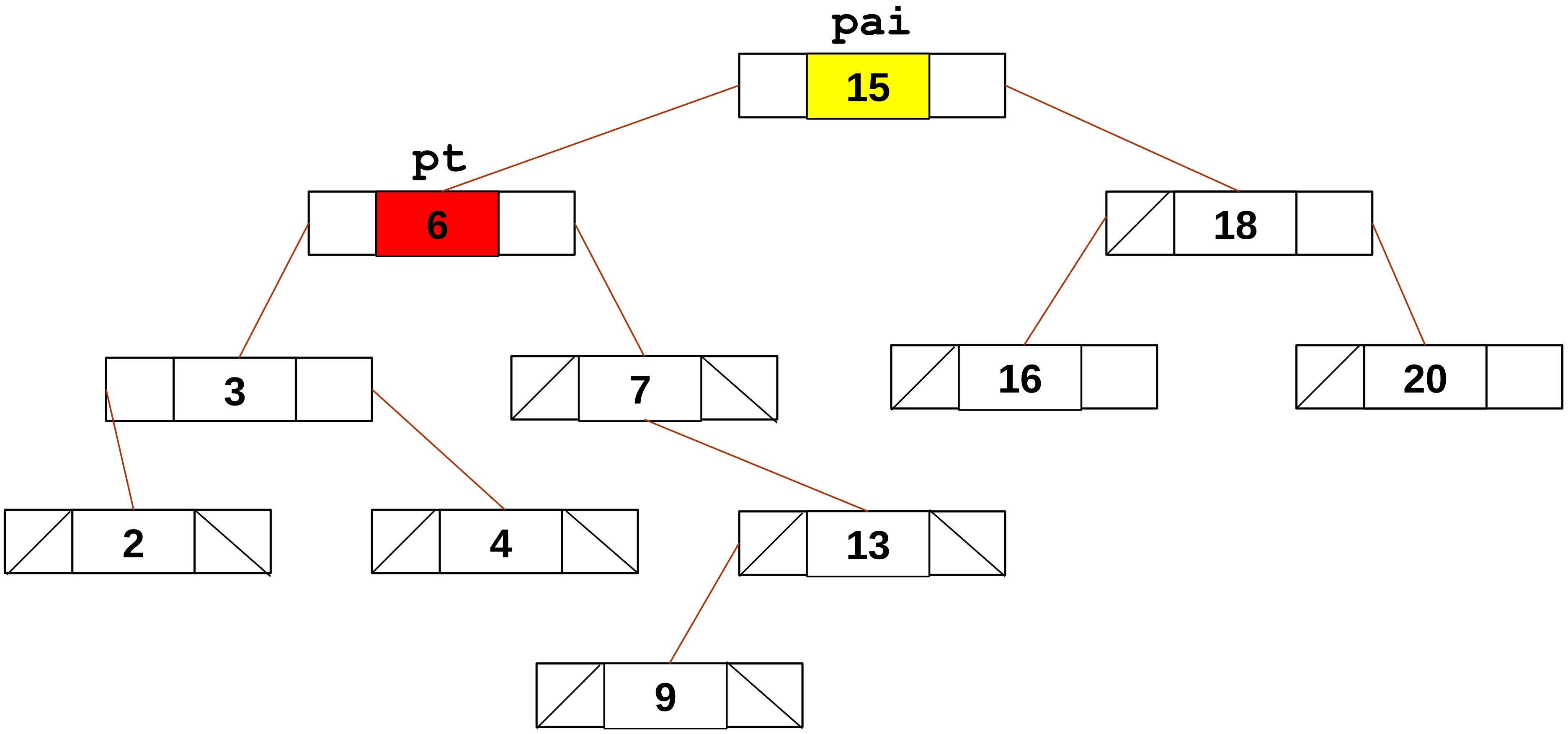
Exemplo

- Sucessor do nó 13:



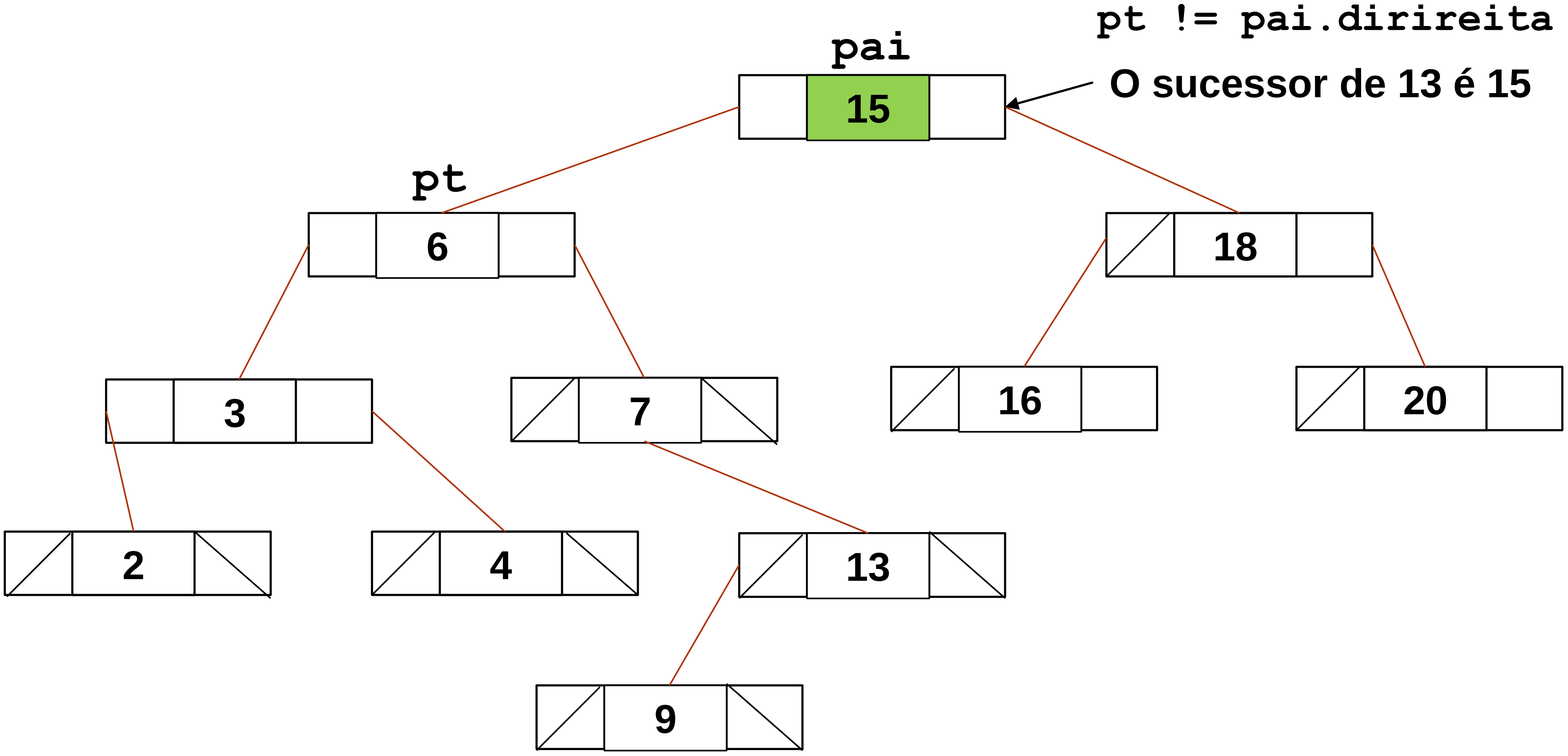
Exemplo

- Sucessor do nó 13:



Exemplo

- Sucessor do nó 13:



Remoção

```
//x -> chave do nó a ser removido
//T -> ArvoreBin
procedimento removerArvoreBin(x, T)
    pt = buscarArvoreBinIter(x, T.raiz) //poderia ser buscarArvoreBinRec
    se pt == NIL
        imprimir(“Nó ” + x + “ inexistente!”)
    senão
        se (pt.esquerda == NIL) e (pt.direita == NIL) //pt não tem filhos
            removerSemFilhos(pt, T)
        senão se (pt.esquerda != NIL) e (pt.direita != NIL) //pt tem dois filhos
            removerDoisFilhos(pt, T)
        senão //pt tem um dos filhos apenas
            removerUnicoFilho(pt, T)
    retorne pt
```

Remoção

```
//pt -> NoArvoreBin a ser removido
//T -> ArvoreBin
procedimento removerSemFilhos(pt, T)
    pai = pt.pai
    se pai != NIL
        se pai.chave >= pt.chave
            pai.esquerda = NIL
        senão
            pai.direita = NIL
    senão //o único nó sem pai em uma árvore é a raiz
        T.raiz = NIL
    pt.pai = NIL
```

Remoção

```
//pt -> NoArvoreBin a ser removido, que tem um único filho
//T -> ArvoreBin
procedimento removerUnicoFilho(pt, T)
    pai = pt.pai
    filho = NIL
    se pt.esquerda != NIL
        filho = pt.esquerda
        pt.esquerda = NIL
    senão
        filho = pt.direita
        pt.direita = NIL
    se pai != NIL
        filho.pai = pai
        se filho.chave <= pai.chave
            pai.esquerda = filho
        senão
            pai.direita = filho
    senão //o único nó sem pai em uma árvore é a raiz
        T.raiz = filho
    pt.pai = NIL
```

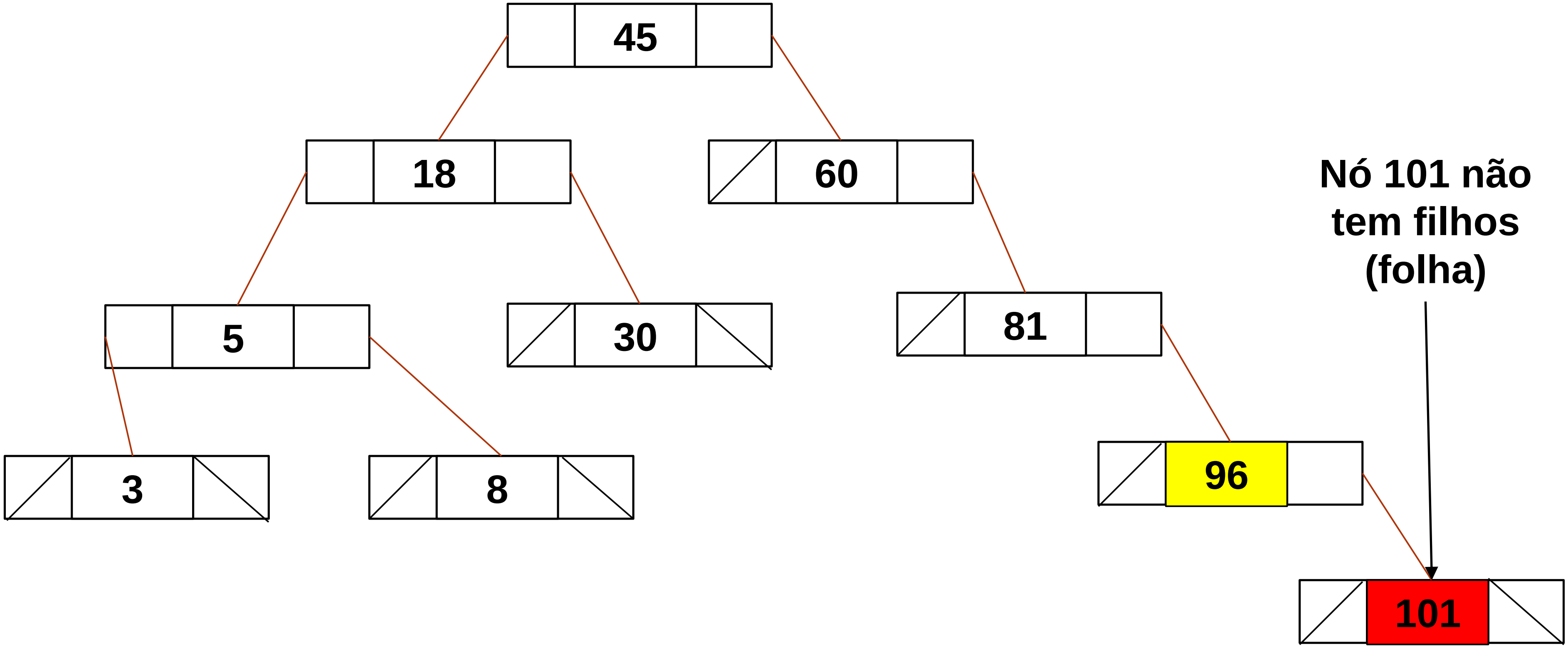
Remoção

- Obrigatoriamente o sucessor terá no máximo um filho quando chamarmos o procedimento abaixo na remoção.

```
//pt -> NoArvoreBin a ser removido, que tem dois filhos
//T -> ArvoreBin
procedimento removerDoisFilhos(pt, T)
    sucessor = encontrarSucessor(pt)
    sucessor = removerArvoreBin(sucessor.chave, T)
    //substituição de pt pelo seu sucessor na árvore
    sucessor.pai = pt.pai
    pt.pai = NIL
    sucessor.esquerda = pt.esquerda
    pt.esquerda = NIL
    sucessor.direita = pt.direita
    pt.direita = NIL
    pai = sucessor.pai
    se pai != NIL
        se pai.chave >= sucessor.chave
            pai.esquerda = sucessor
        senão
            pai.direita = sucessor
    senão
        T.raiz = sucessor
    sucessor.esquerda.pai = sucessor
    se sucessor.direita != NIL //pode ter ficado NIL, se antes pt.direita == sucessor
        sucessor.direita.pai = sucessor
```

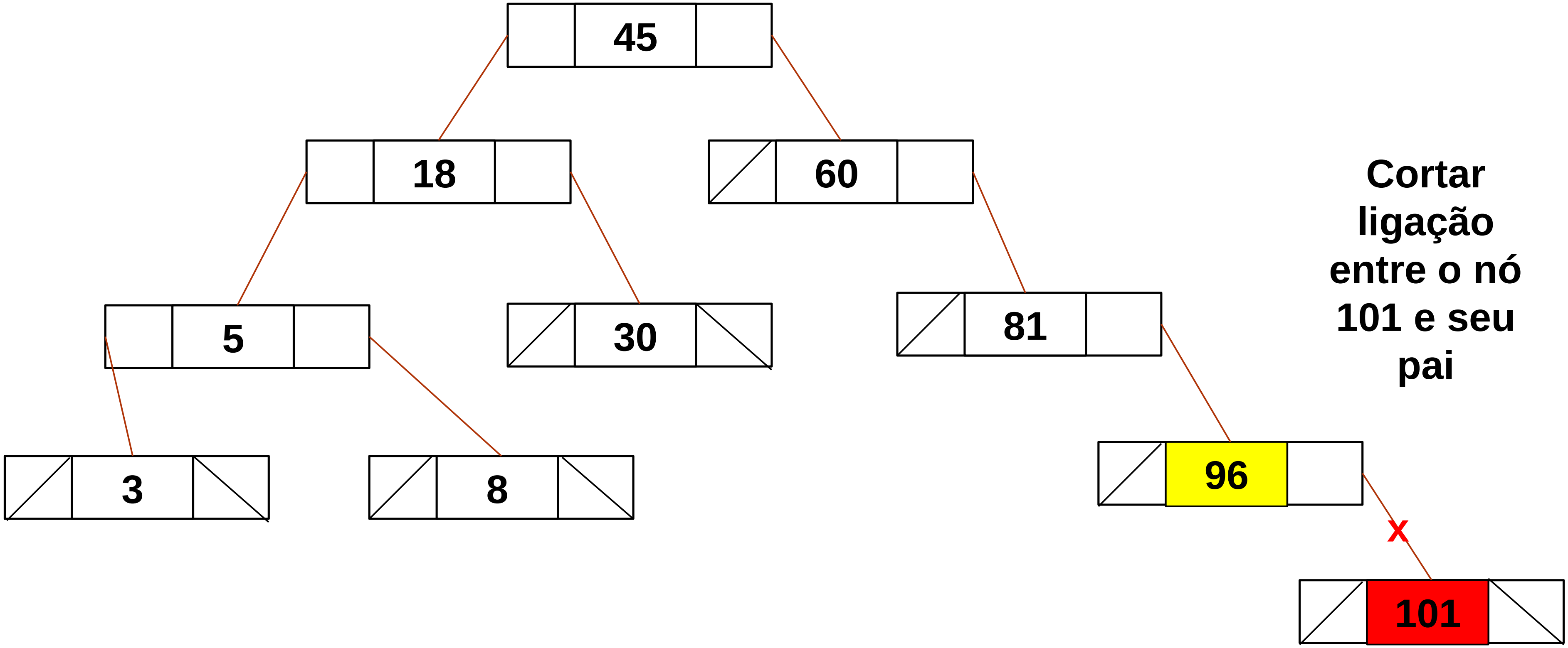
Exemplo 1

- Remover: 101



Exemplo 1

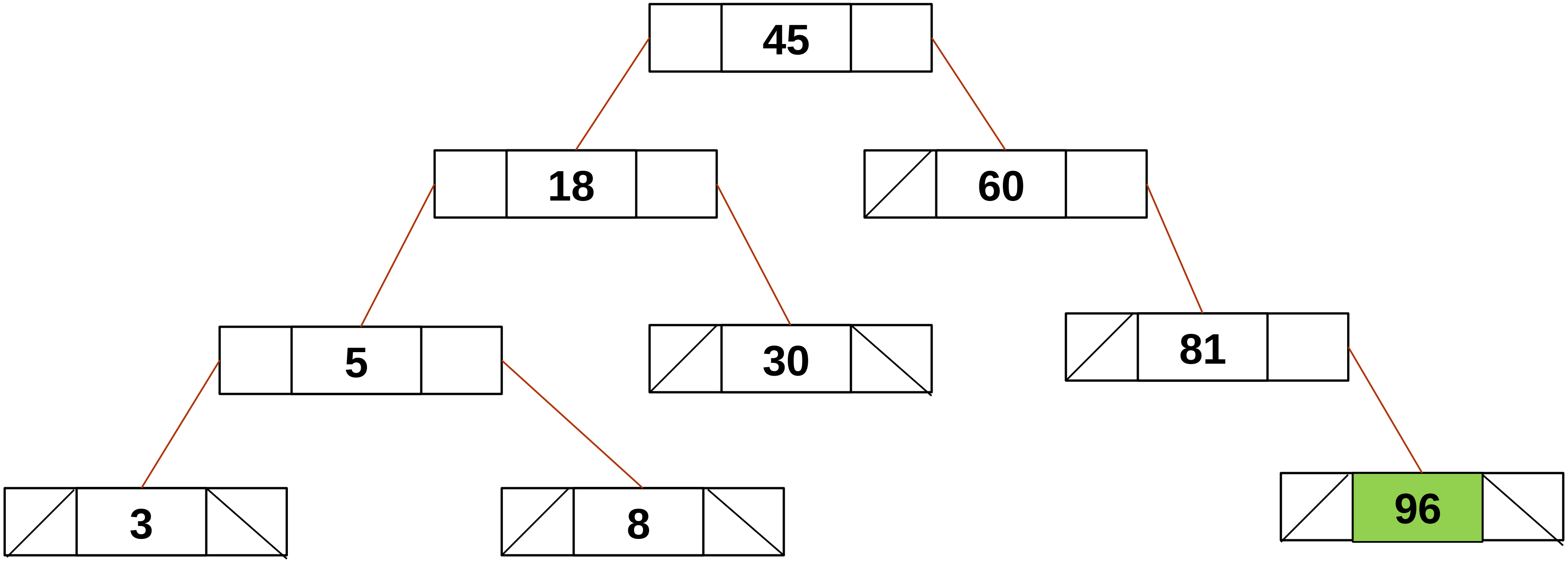
- Remover: 101



Cortar ligação entre o nó 101 e seu pai

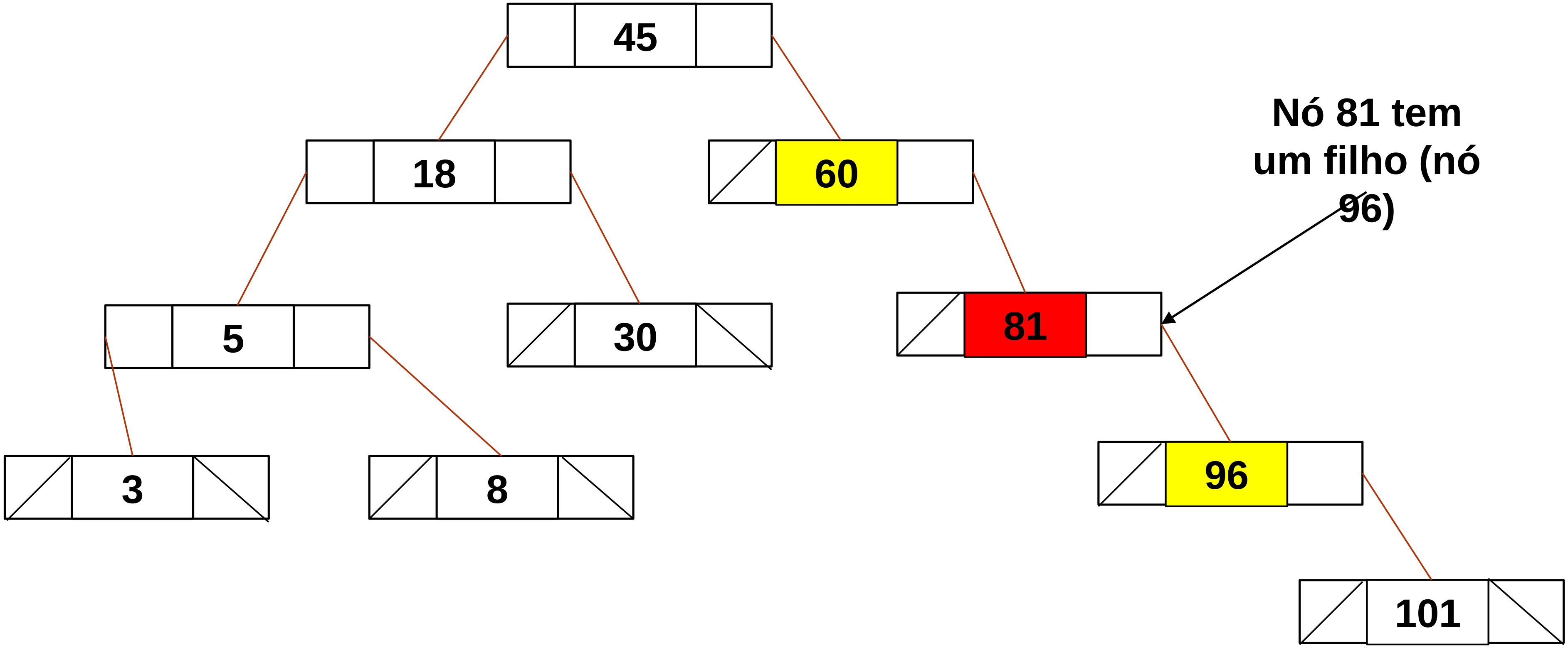
Exemplo 1

- Remover: 101



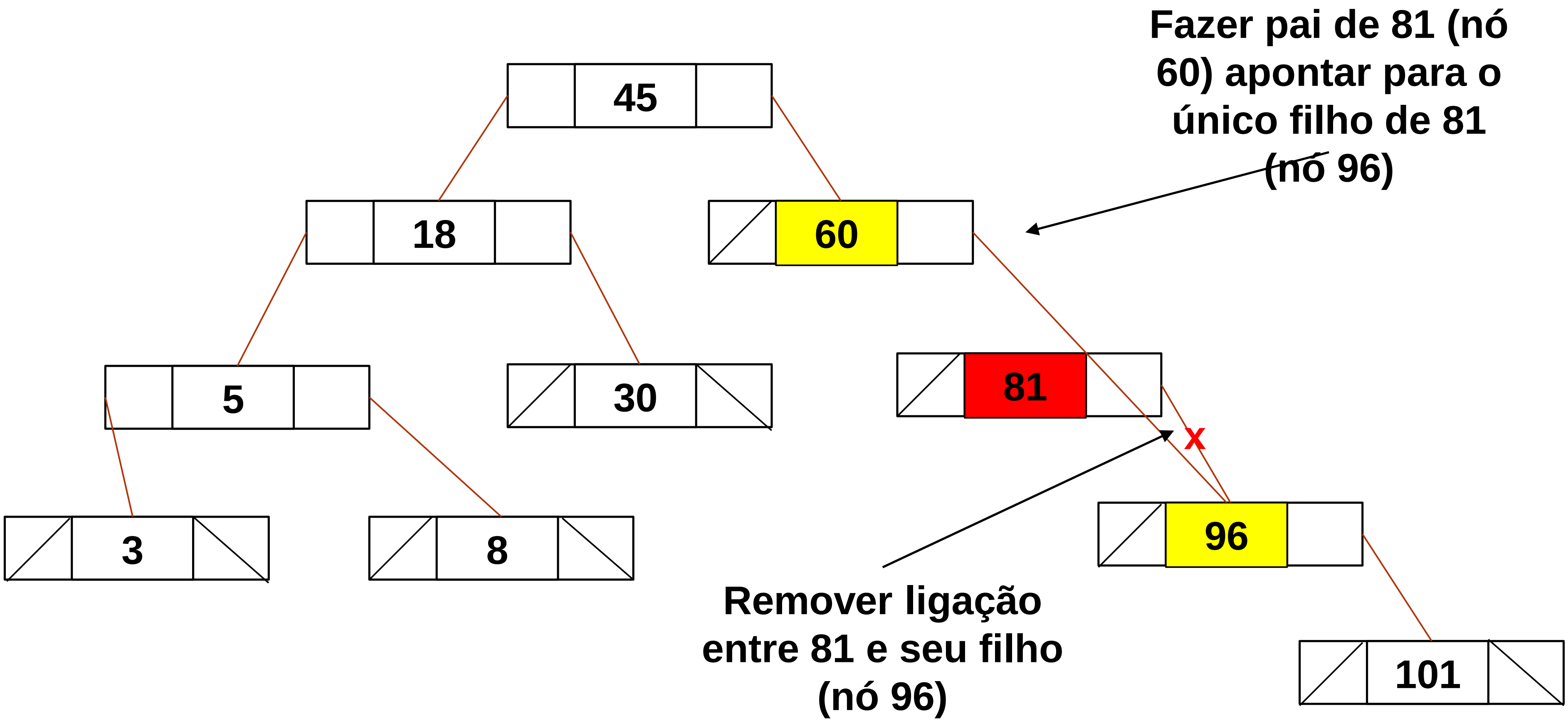
Exemplo 2

- Remover: 81



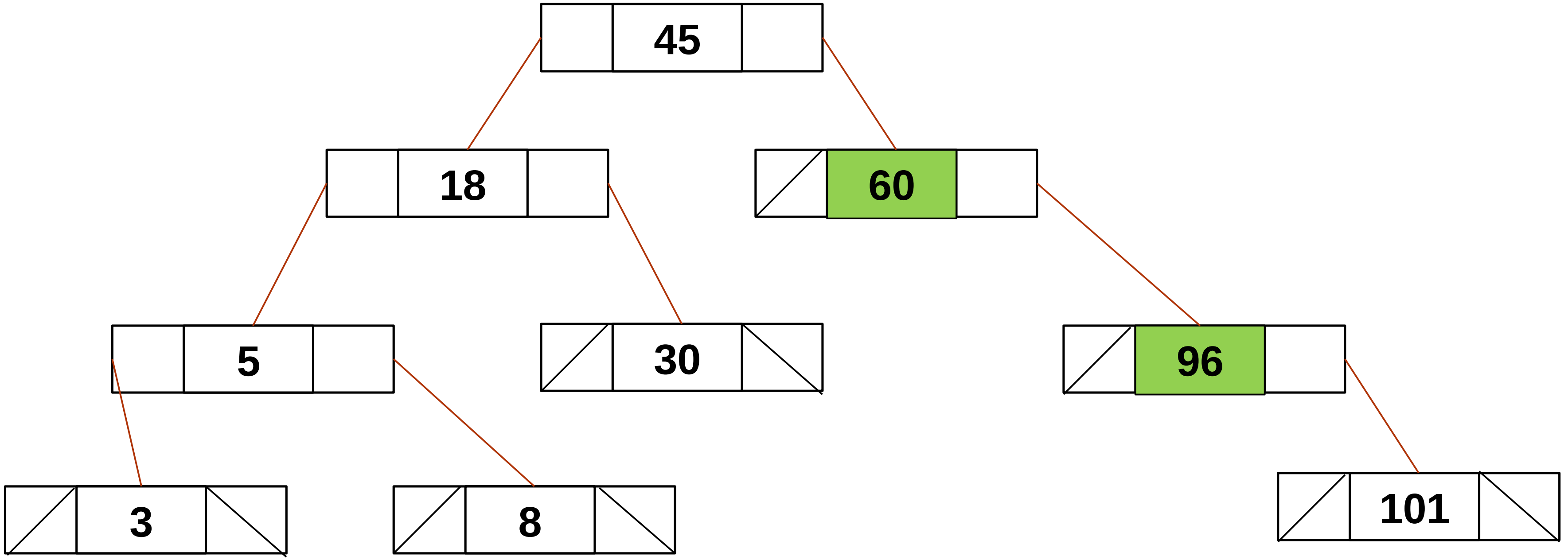
Exemplo 2

- Remover: 81



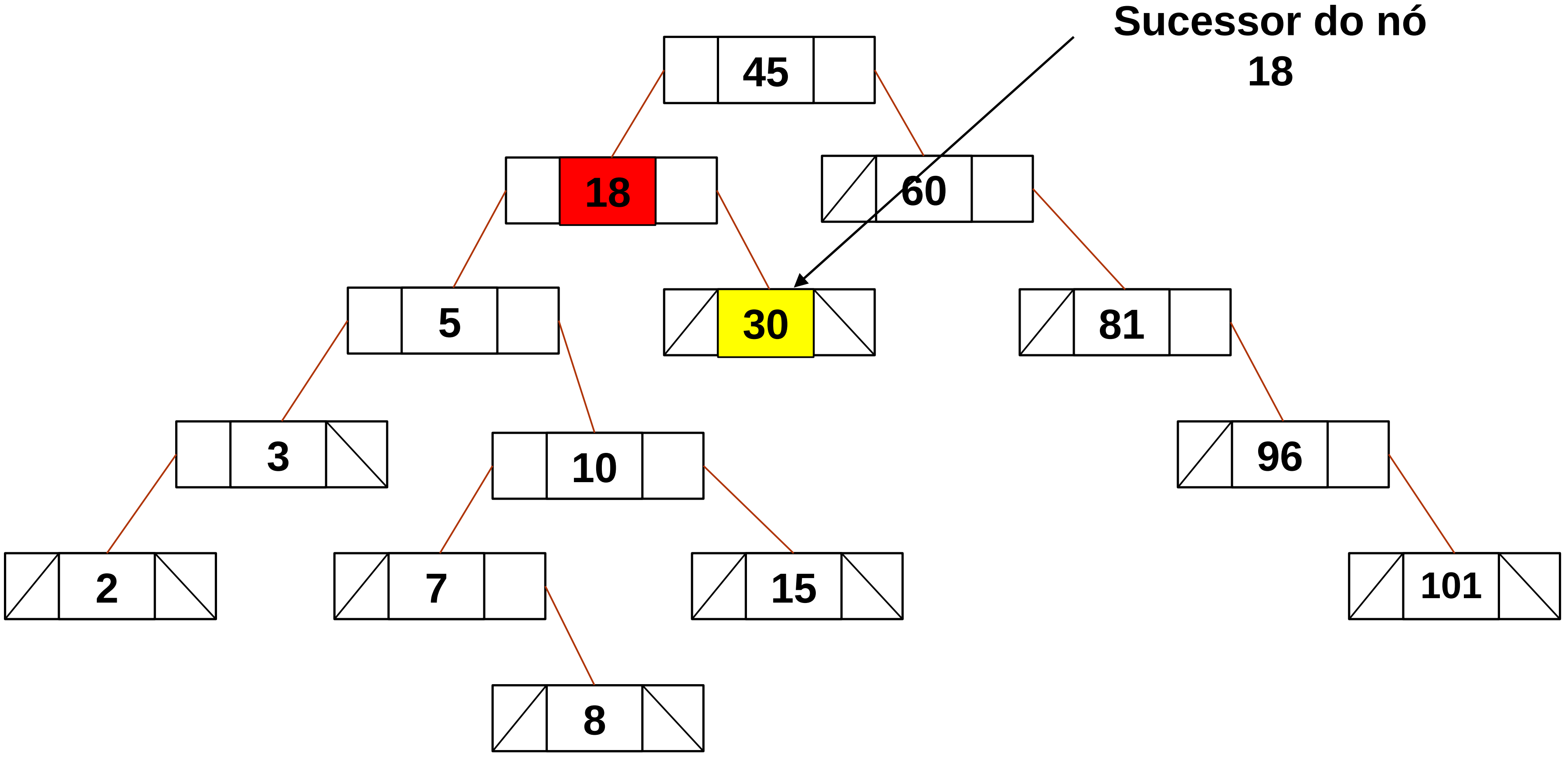
Exemplo 2

- Remover: 81



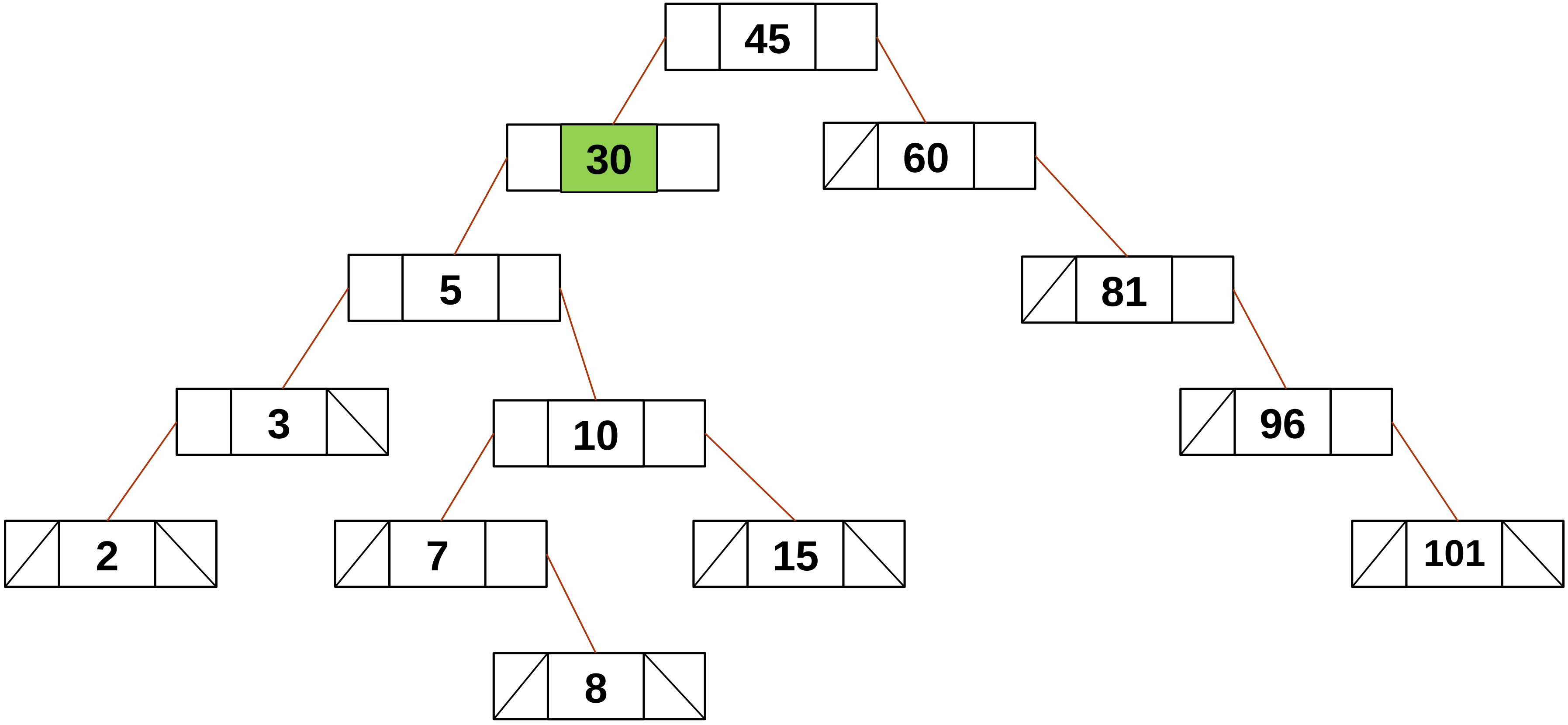
Exemplo 3

- Remover: 18



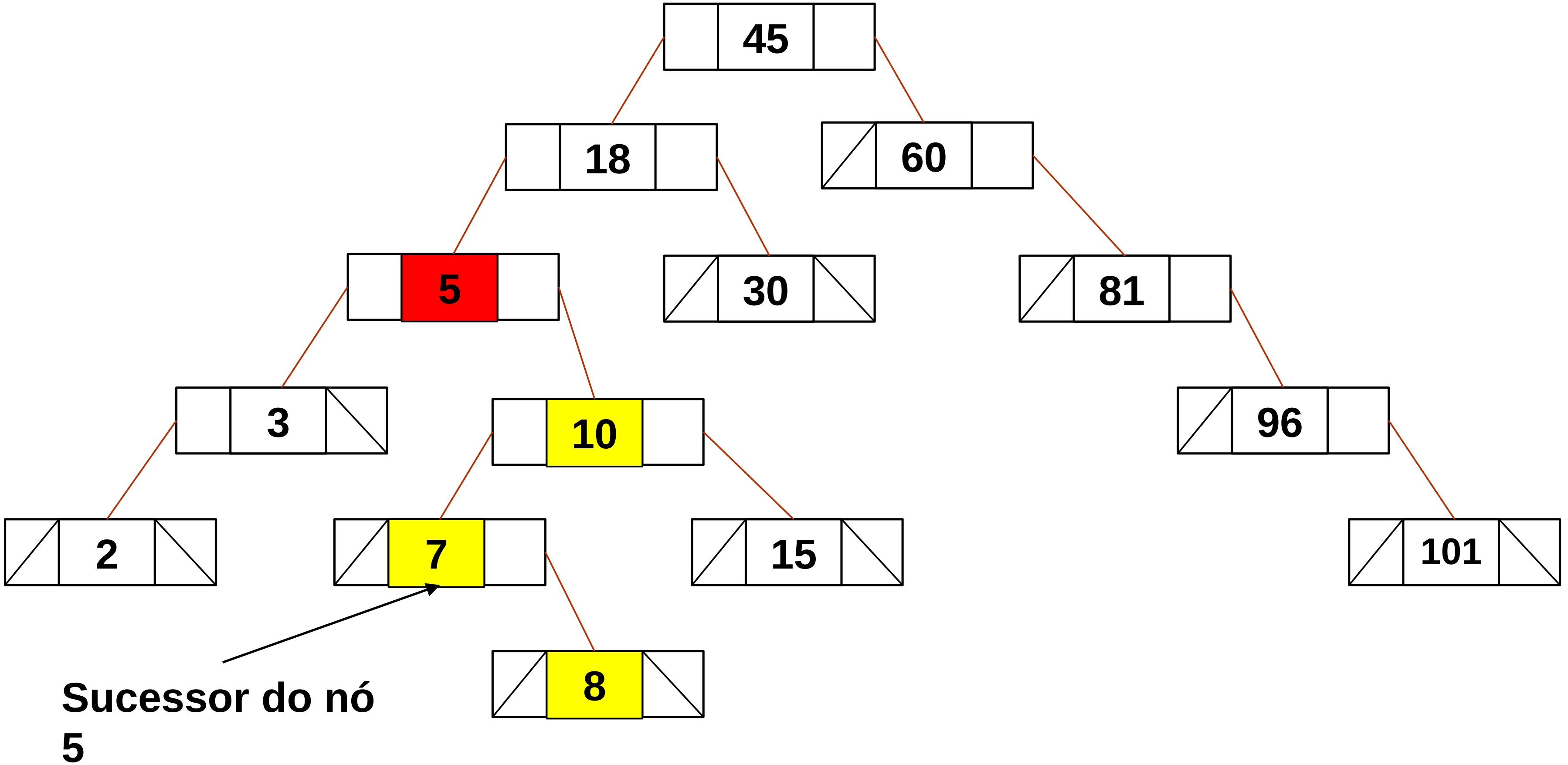
Exemplo 3

- Remover: 18



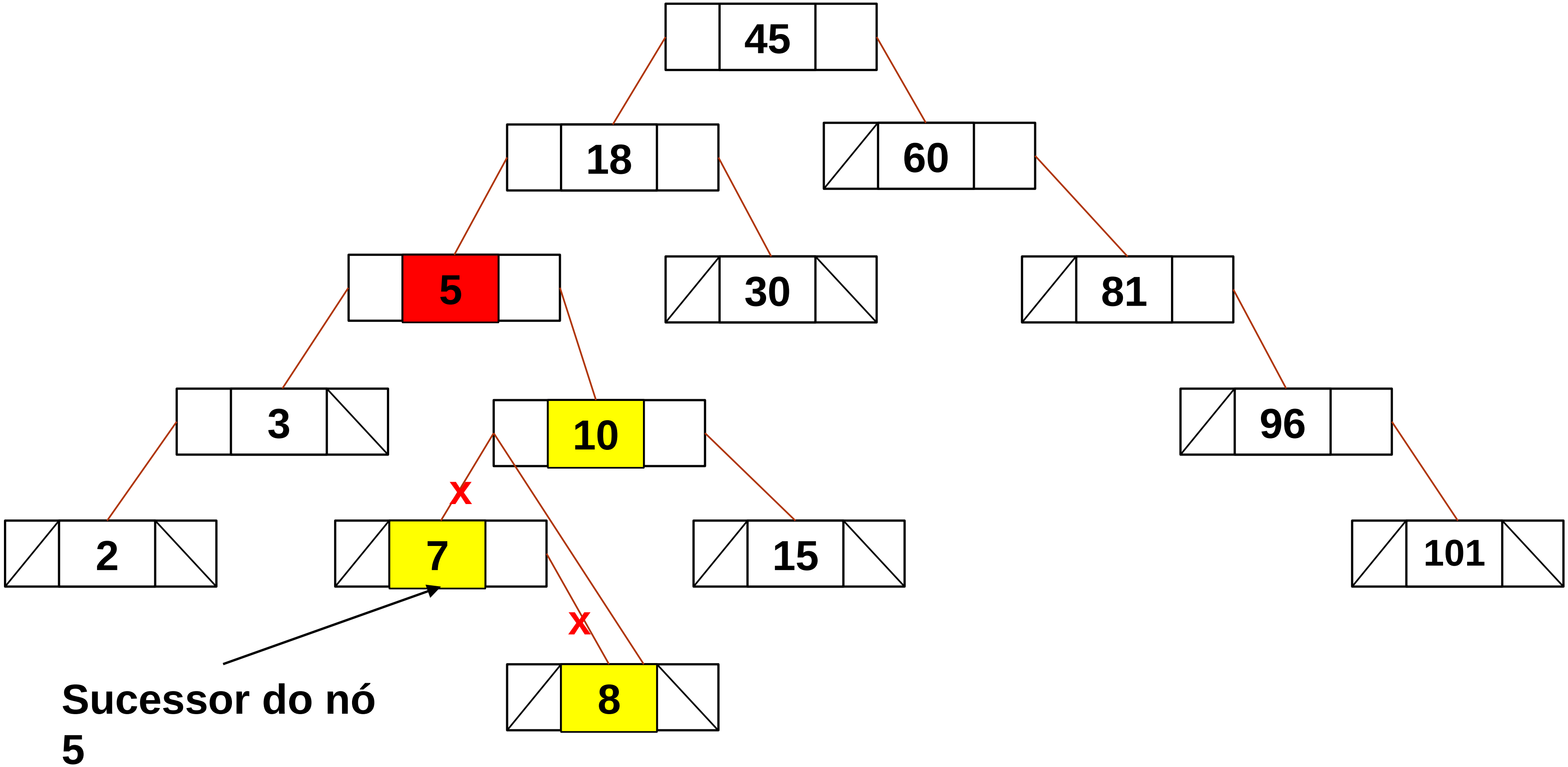
Exemplo 4

- Remover: 5



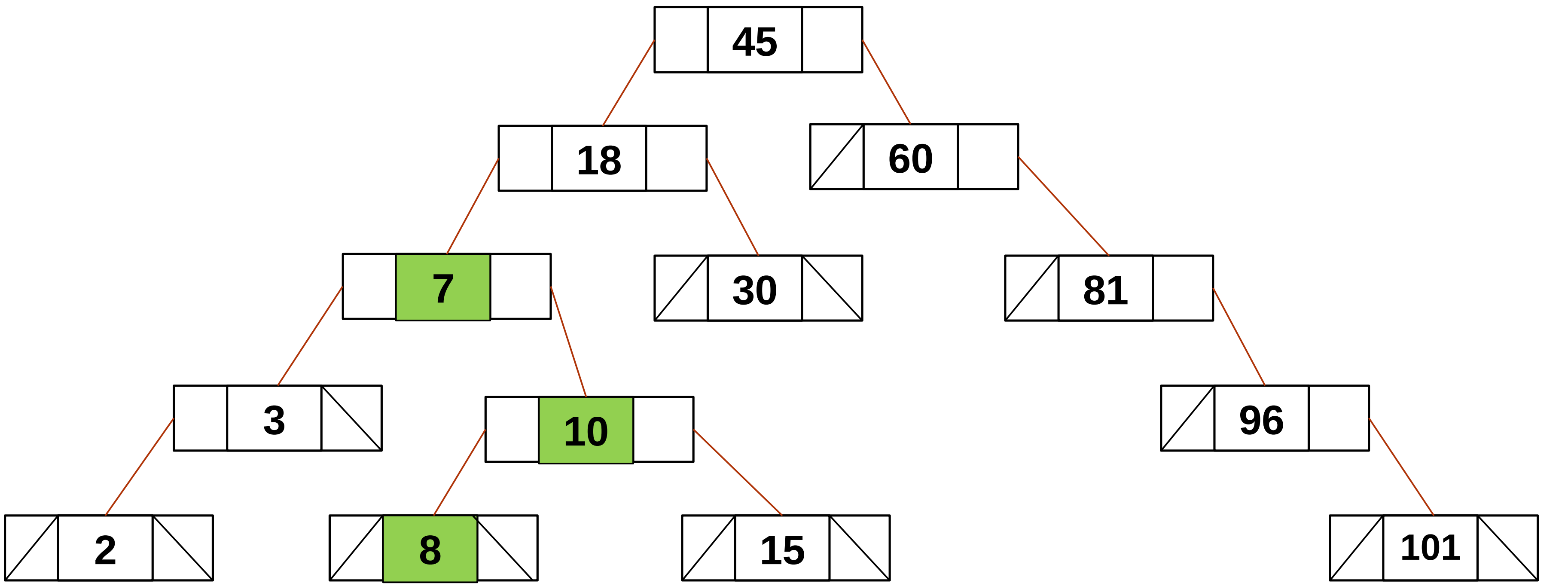
Exemplo 4

- Remover: 5



Exemplo 4

- Remover: 5



Referências

- CORMEN, H. T.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. Introduction to Algorithms, 3rd ed., *Boston: MIT Press*, 2009.

- SZWARCFITER, J.; MARKENZON, L. Estruturas de Dados e seus Algoritmos, 3ª ed. Rio de Janeiro: LTC, 2010.

- FEOFILOFF, Paulo. Algoritmos em Linguagem C. Editora Campus/Elsevier, 2009.

Árvores de Busca

Algoritmos e Estruturas de Dados
Prof. Dr. Luciano Demétrio Santos Pacífico
{luciano.pacifico@ufrpe.br}

UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO