Universidade Federal do Pampa
Campus-Bagé
Engenharia de Computação

# Laboratório de Programação I

## Aula 11 – Árvores de Pesquisa Binária

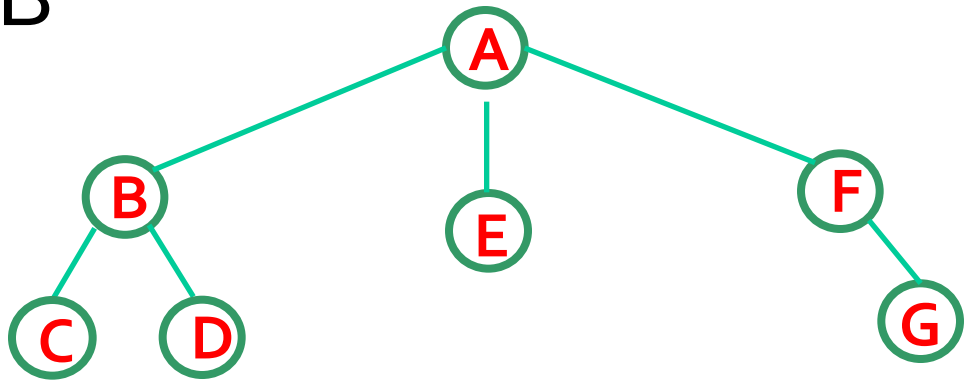
Prof. Julio Saraçol
juliodomingues@unipampa.edu.br

# Árvores

- Apresentam uma solução eficiente para:
  - inserção, remoção e busca;
- A forma mais comum de representar graficamente  é através de sua representação hierárquica;
- É composta por um conjunto de nós
- Existe um nó denominado raiz, que contém zero ou mais subárvores, cujas raízes são ligadas diretamente a ele

# Árvores

- B, E e F são **filhos** de A
- C e D são **filhos** de B
- G é **filho** de F
- B, E e F são **irmãos**
- E é **tio** de C, D e G
- A é **avô** de C, D e G



# Árvores

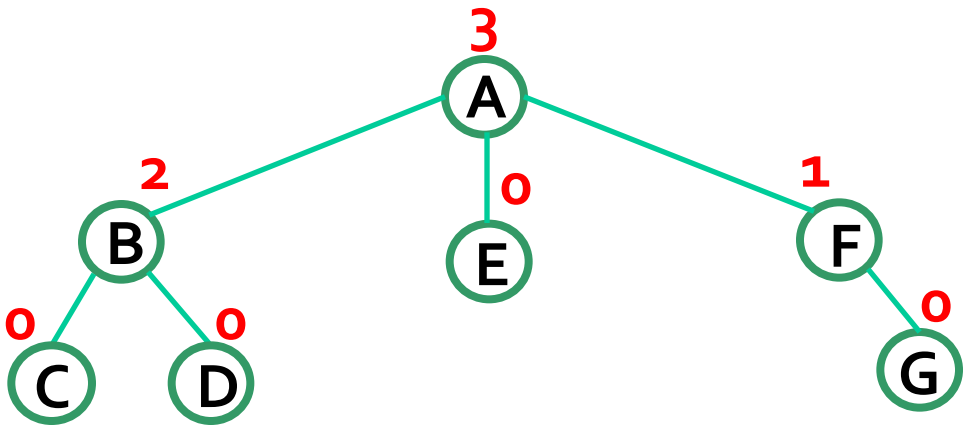
- Nós com **filhos** são comumente chamados de **nós internos** e nós que não têm filhos são chamados de **folhas**, ou **nós externos**

- O **número de filhos permitido** por nó e as informações armazenadas em cada nó diferenciam os diversos **tipos de árvores** existentes

# Árvores

- **GRAU**: o grau de um nó T de uma árvore é igual ao número de filhos do nó T;
- **GRAU DA ÁRVORE**: o grau de uma árvore T é o grau máximo entre os graus de todos os seus nós;
- **ALTURA**: a altura de um nó T é o número de passos do mais longo caminho que leva de T até uma folha. Os nós folha sempre têm altura igual a 0;
- **ALTURA ou PROFUNDIDADE DE UMA ÁRVORE**: A altura de uma árvore T é dada pela altura da raiz da árvore.

# Árvores



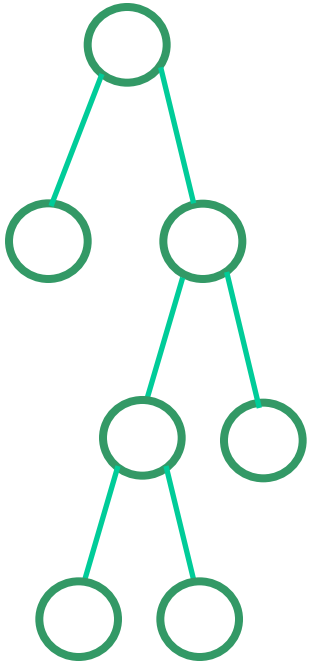

- Grau da árvore: 3
- h(C), h(D), h(E) e h(G): 0
- h(B): 1
- h(A): 2
- h(F): 1
- Altura da árvore: 2

# Árvores Binárias

- Uma árvore binária T é um conjunto finito de elementos denominados nós ou vértices, tal que:
  - T = ø e a árvore é dita vazia, ou
  - Existe um nó especial **r**, chamado de raiz de **T**, e os restantes podem ser divididos em dois subconjuntos disjuntos, $T_r^E$ e $T_r^D$ , a **subárvore esquerda** e a **direita de r**, respectivamente, as quais são também árvores binárias.

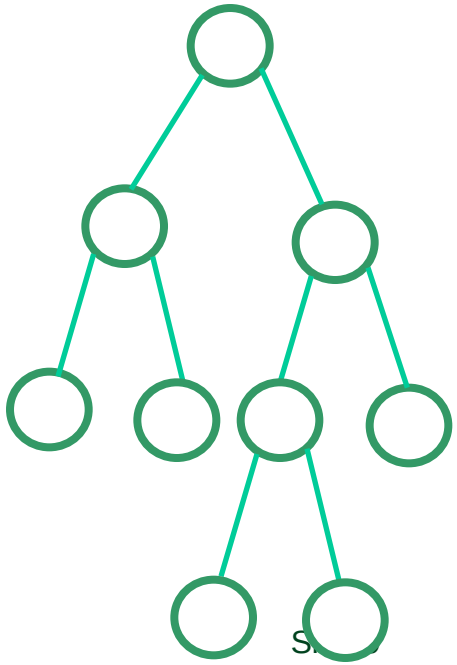
# Árvores Binárias

- Árvore estritamente binária: cada nó possui 0 ou 2 filhos



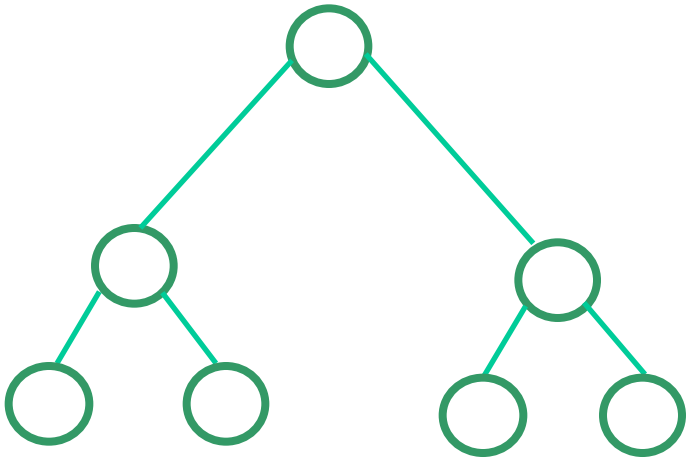
# Árvores Binárias

- **Árvore binária completa**: se v é um nó tal que alguma subárvore de v é **vazia**, então v se **localiza ou no último (maior) ou no penúltimo nível da árvore**.



# Árvores Binárias

- **Árvore binária cheia**: se v é um nó com alguma de suas subárvores **vazias**, então v se **localiza no último nível**

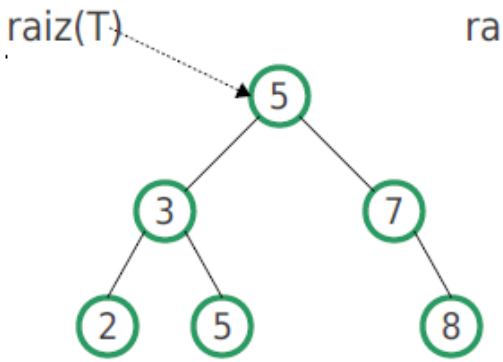


# Árvores de Pesquisa

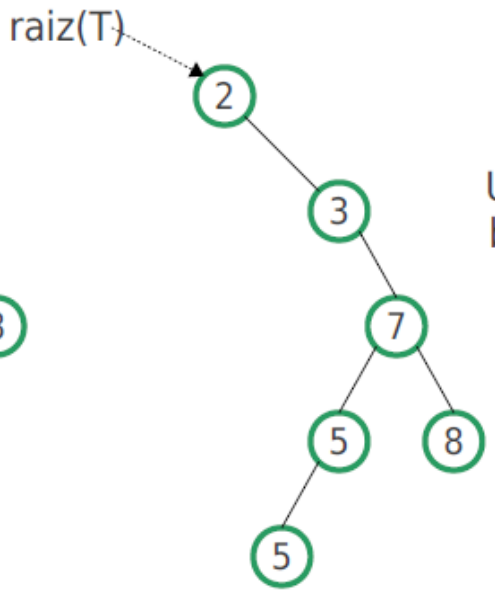
- As árvores de pesquisa são estruturas de dados que admitem muitas operações
  - SEARCH,
  - MINIMUM,
  - MAXIMUM,
  - PREDECESSOR,
  - SUCESSOR,
  - INSERT
  - DELETE
- Podem ser usadas como dicionário e fila de prioridades

# Árvore de Pesquisa Binária

- Uma árvore de pesquisa binária é organizada, conforme o nome sugere, em uma árvore binária



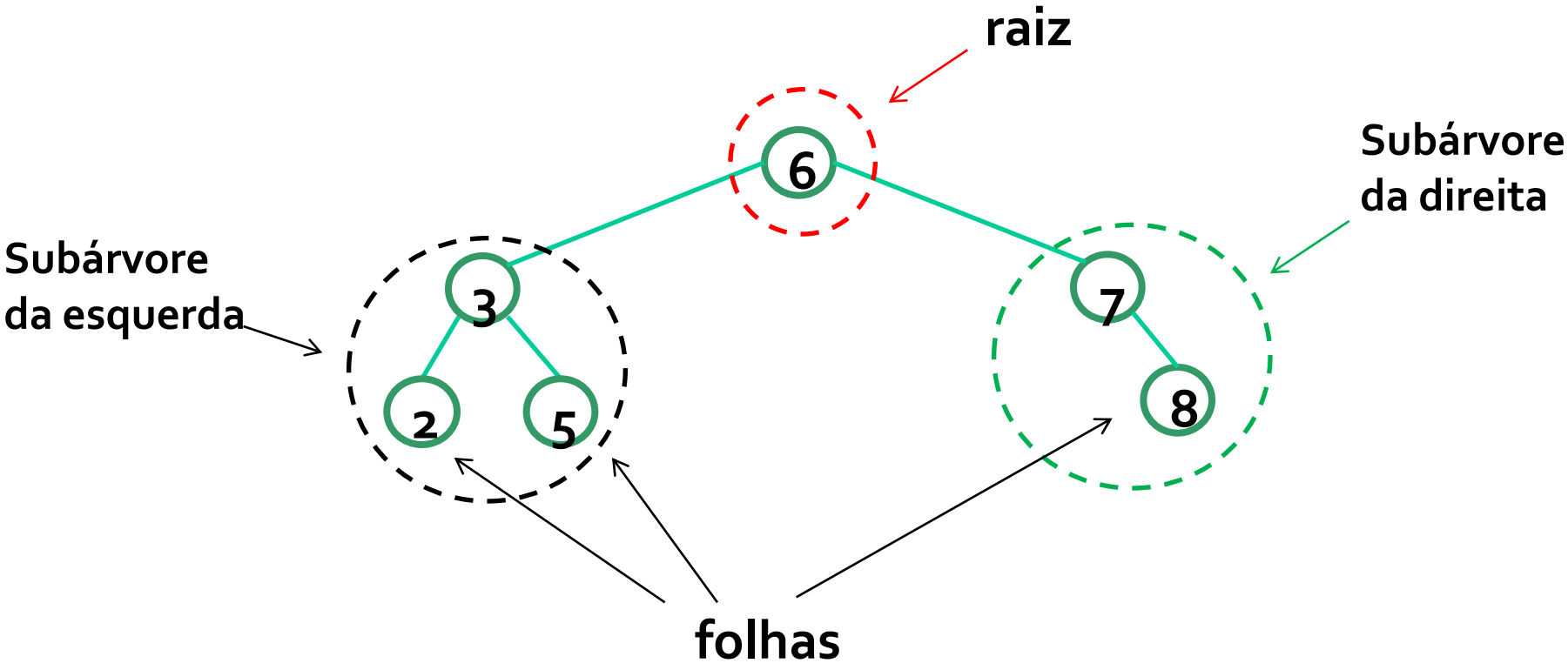

Uma árvore de pesquisa binária em 6 nós com altura 2.




Uma árvore de pesquisa binária menos eficiente, com altura 4 e com as mesmas chaves.

# Árvore de Pesquisa Binária

- Elementos



# Árvore de Pesquisa Binária
# Definições

- Uma árvore deste tipo pode ser representada por uma **estrutura de dados encadeada** em que cada nó é um objeto
- Nó contém:
  - campo *chave*,
  - campos

    *esquerdo, direito* e *pai*.
- Se um filho ou o pai estiver ausente, o campo apropriado conterá NIL/NULL.
- O nó raiz é o único nó da árvore cujo campo *pai é* NIL.

# Árvore de Pesquisa Binária
# Definições

- *Propriedade de árvore de pesquisa binária:*

Seja ***x*** um nó em uma árvore de pesquisa binária.

Se ***y*** é um nó na subárvore **esquerda** de ***x***, então ***chave[y] ≤ chave[x]***.

Se ***y*** é um nó na subárvore **direita** de ***x***, então ***chave[x] ≤ chave[y]***.

# Árvore de Pesquisa Binária
# Percurso

- A propriedade das APB permite **imprimir todas as chaves** em uma APB **em sequência ordenada**, por meio de um algoritmo recursivo bastante simples, chamado de ***percurso de árvore em ordem (INORDER).***

- Este nome deriva do fato de que a chave raiz de uma subárvore é impressa entre os valores de sua subárvore esquerda e aqueles de sua subárvore direita.

# Árvore de Pesquisa Binária
# Percurso

- Para realizar o percurso e imprimir todos os elementos a partir da raiz da árvore binária *T, chama-se INORDER-TREE-WALK(raiz[T]).*

INORDER-TREE-WALK(*x*)
**if** x ≠ NIL **then**
  INORDER-TREE-WALK(*esquerda[x]*)
  print *chave[x]*
  INORDER-TREE-WALK(*direita[x]*)

# Árvore de Pesquisa Binária
# Percurso

- Um ***percurso de árvore em pré-ordem*** imprime a raiz antes dos valores em uma ou outra subárvore

PRE-ORDER-TREE-WALK(*x*)
**if** x ≠ NIL **then**
  print *chave[x]*
  PRE-ORDER-TREE-WALK(*esquerda[x]*)
  PRE-ORDER-TREE-WALK(*direita[x]*)

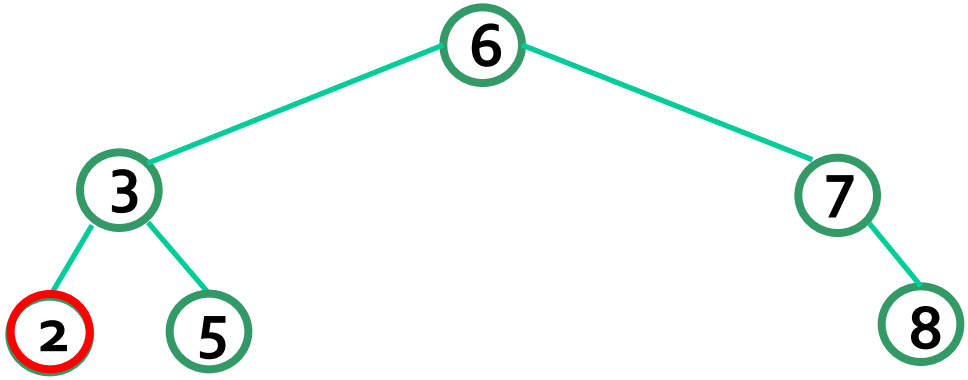
# Árvore de Pesquisa Binária
# Percurso

- Um ***percurso de árvore em pós-ordem imprime a raiz depois*** dos valores contidos em suas subárvores.

POST-ORDER-TREE-WALK(*x*)
**if** x ≠ NIL **then**
  POST-ORDER-TREE-WALK(*esquerda[x]*)
  POST-ORDER-TREE-WALK(*direita[x]*)
  print *chave[x]*

# Árvore de Pesquisa Binária
# Percurso

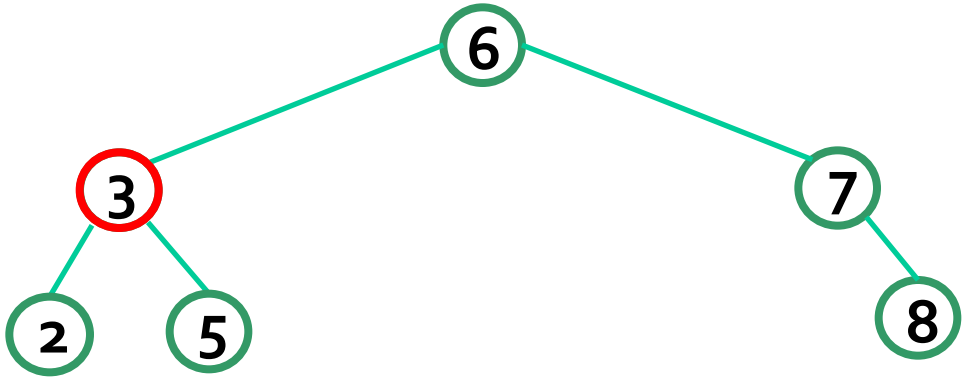
- Exemplo INORDER




2

# Árvore de Pesquisa Binária
# Percurso

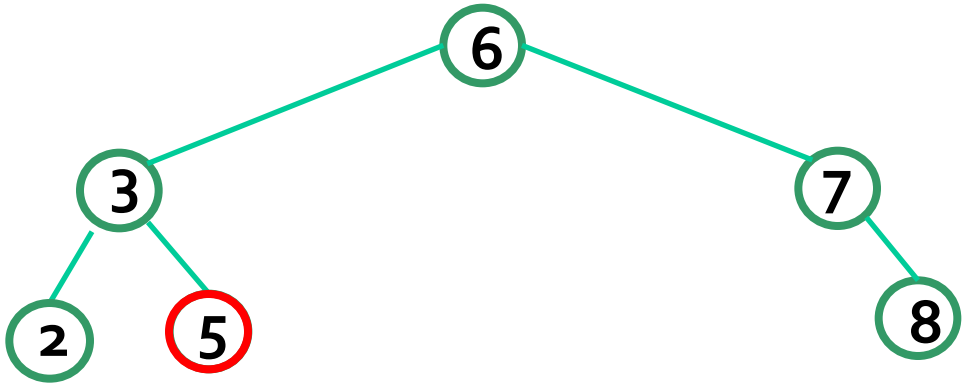
- Exemplo INORDER




2 3

# Árvore de Pesquisa Binária
# Percurso

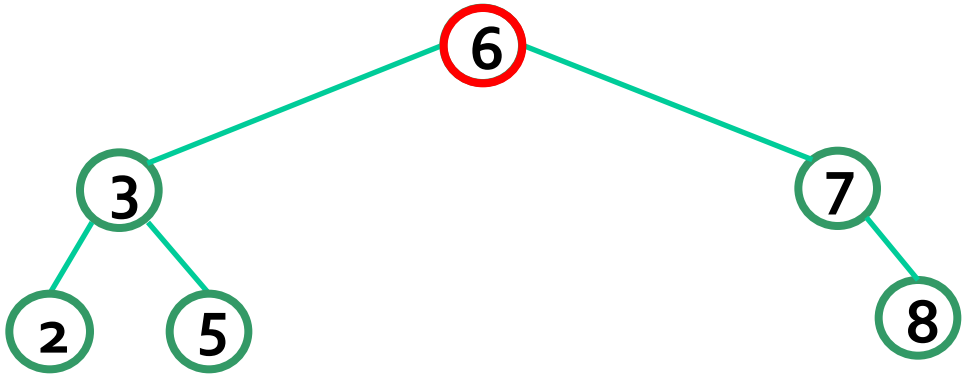
- Exemplo INORDER




2 3 5

# Árvore de Pesquisa Binária
# Percurso

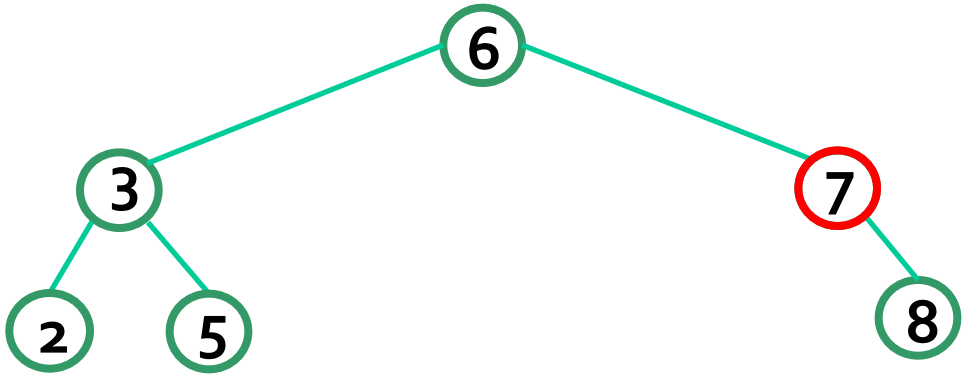
- Exemplo INORDER




2 3 5 6

# Árvore de Pesquisa Binária
# Percurso

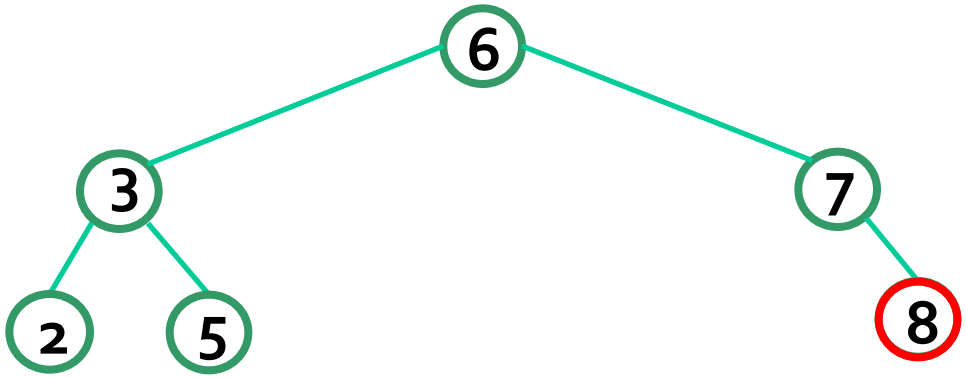
- Exemplo INORDER




2 3 5 6 7

# Árvore de Pesquisa Binária Percurso

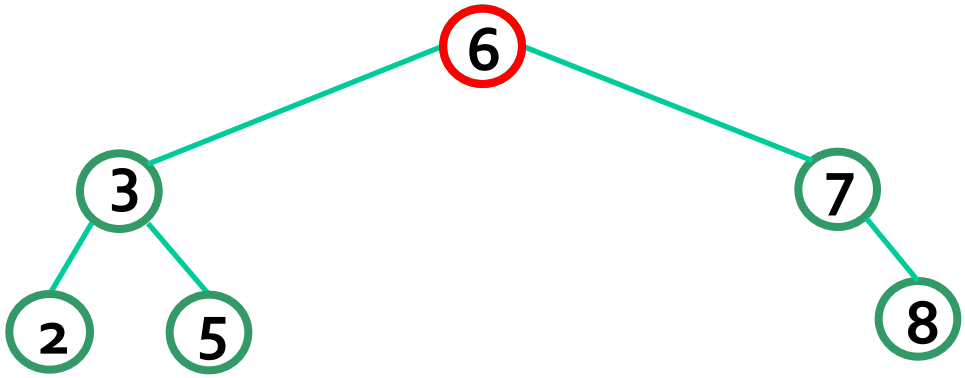
- Exemplo INORDER




2 3 5 6 7 8

# Árvore de Pesquisa Binária
# Percurso

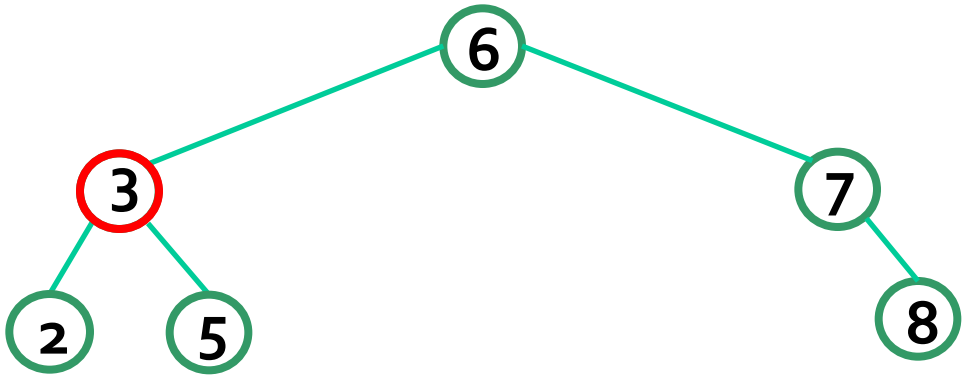
- Exemplo PRE-ORDER




6

# Árvore de Pesquisa Binária
# Percurso

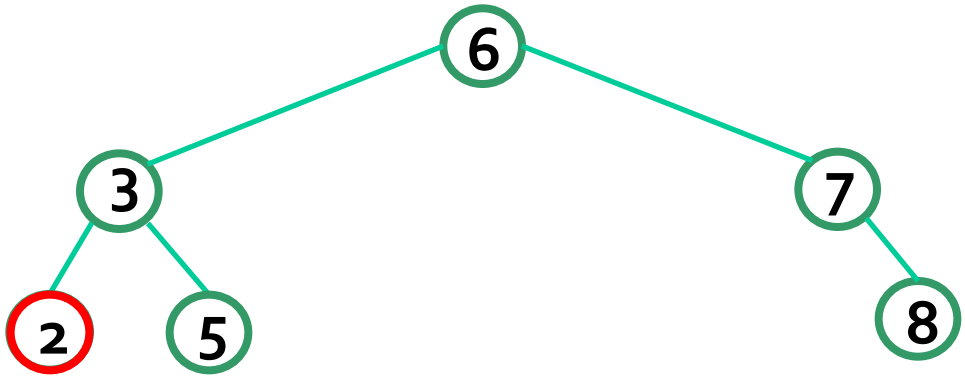
- Exemplo PRE-ORDER




6 3

# Árvore de Pesquisa Binária
# Percurso

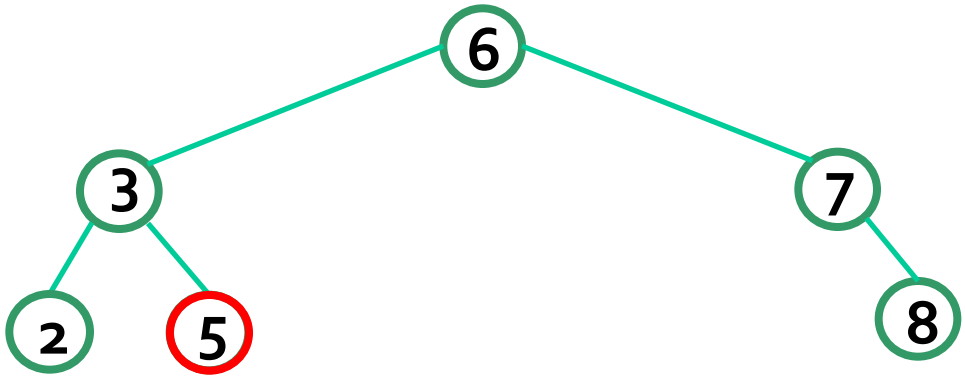
- Exemplo PRE-ORDER




6 3 2

# Árvore de Pesquisa Binária
# Percurso

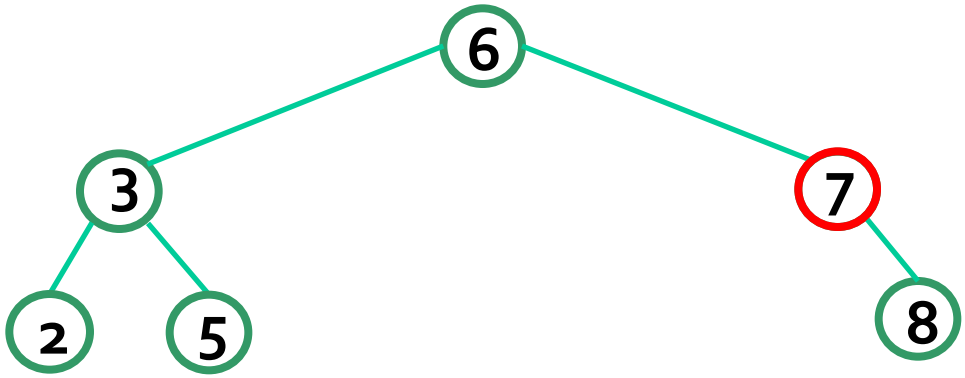
- Exemplo PRE-ORDER




6 3 2 5

# Árvore de Pesquisa Binária
# Percurso

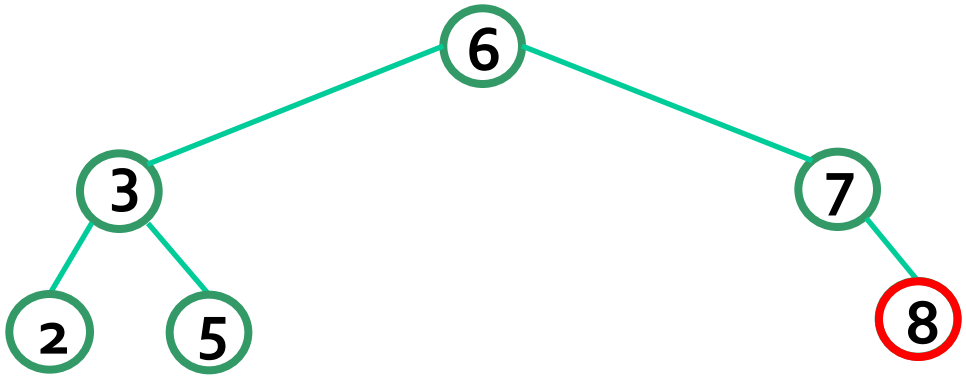
- Exemplo PRE-ORDER




6 3 2 5 7

# Árvore de Pesquisa Binária
# Percurso

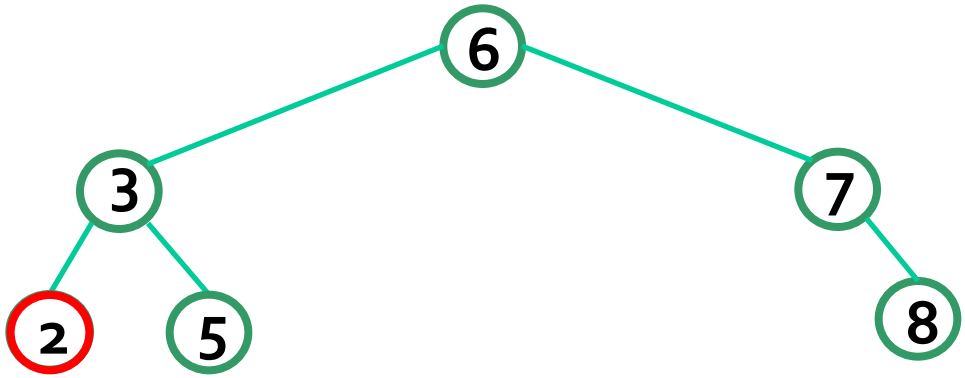
- Exemplo PRE-ORDER




6 3 2 5 7 8

# Árvore de Pesquisa Binária
# Percurso

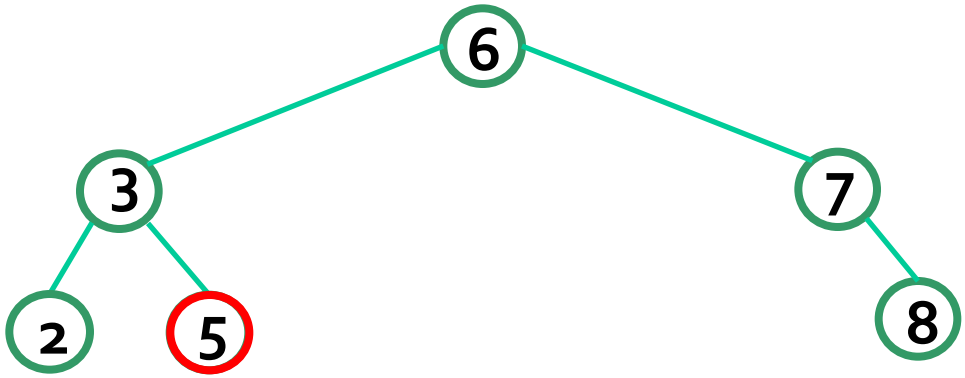
- Exemplo POST-ORDER




2

# Árvore de Pesquisa Binária
# Percurso

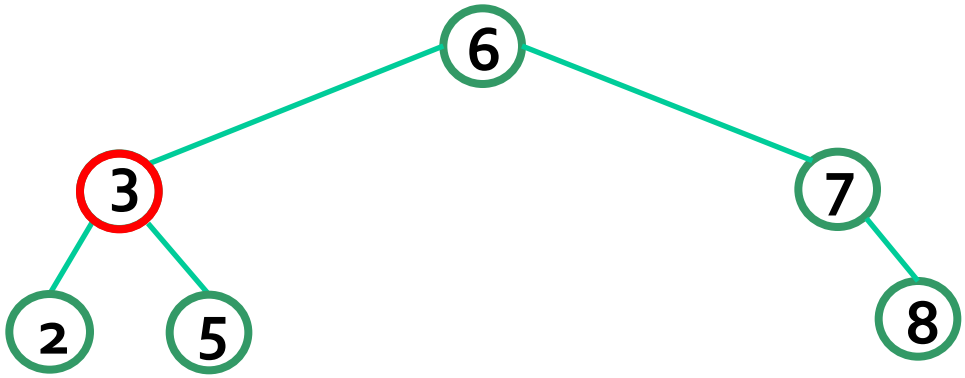
- Exemplo POST-ORDER




2 5

# Árvore de Pesquisa Binária
# Percurso

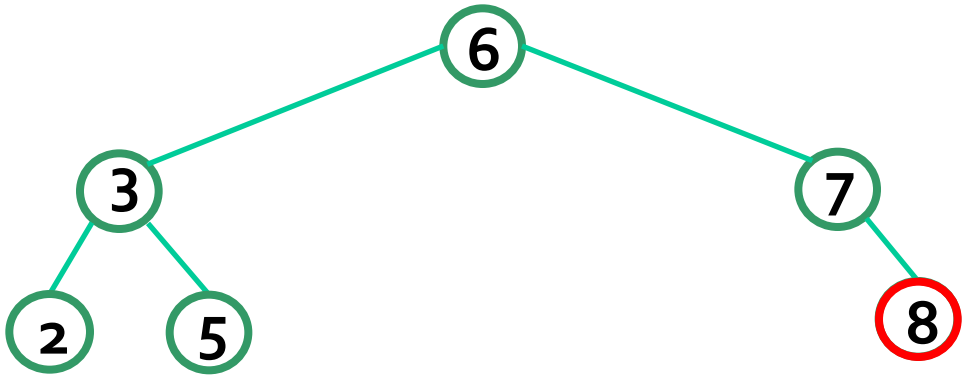
- Exemplo POST-ORDER




2 5 3

# Árvore de Pesquisa Binária
# Percurso

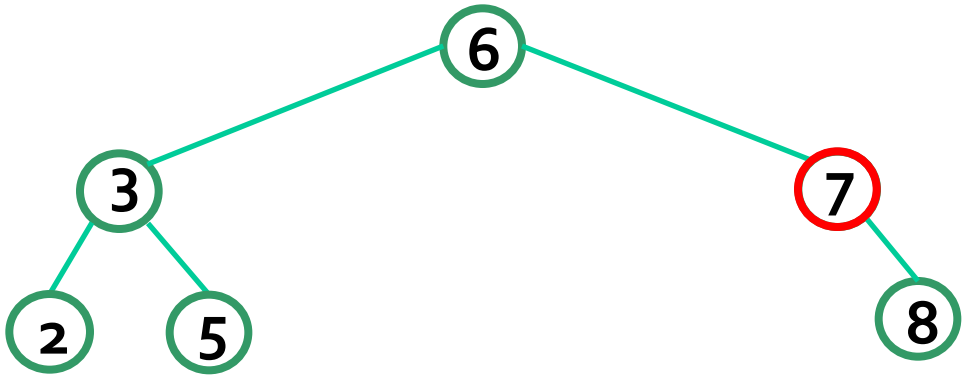
- Exemplo POST-ORDER




2 5 3 8

# Árvore de Pesquisa Binária
# Percurso

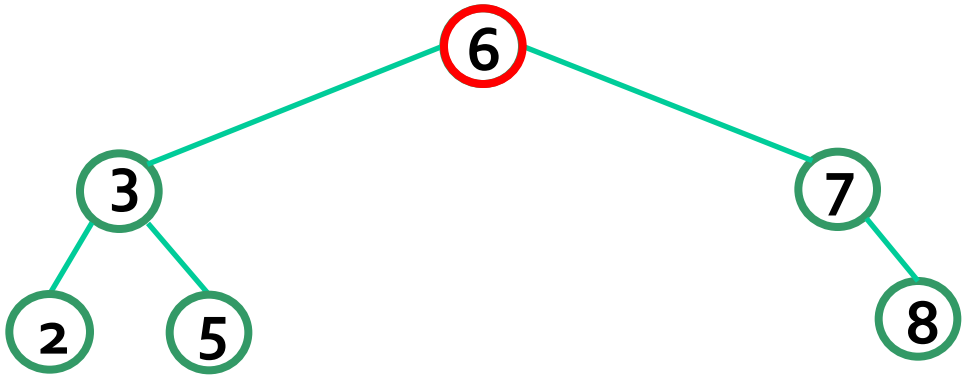
- Exemplo POST-ORDER




2 5 3 8 7

# Árvore de Pesquisa Binária
# Percurso

- Exemplo POST-ORDER



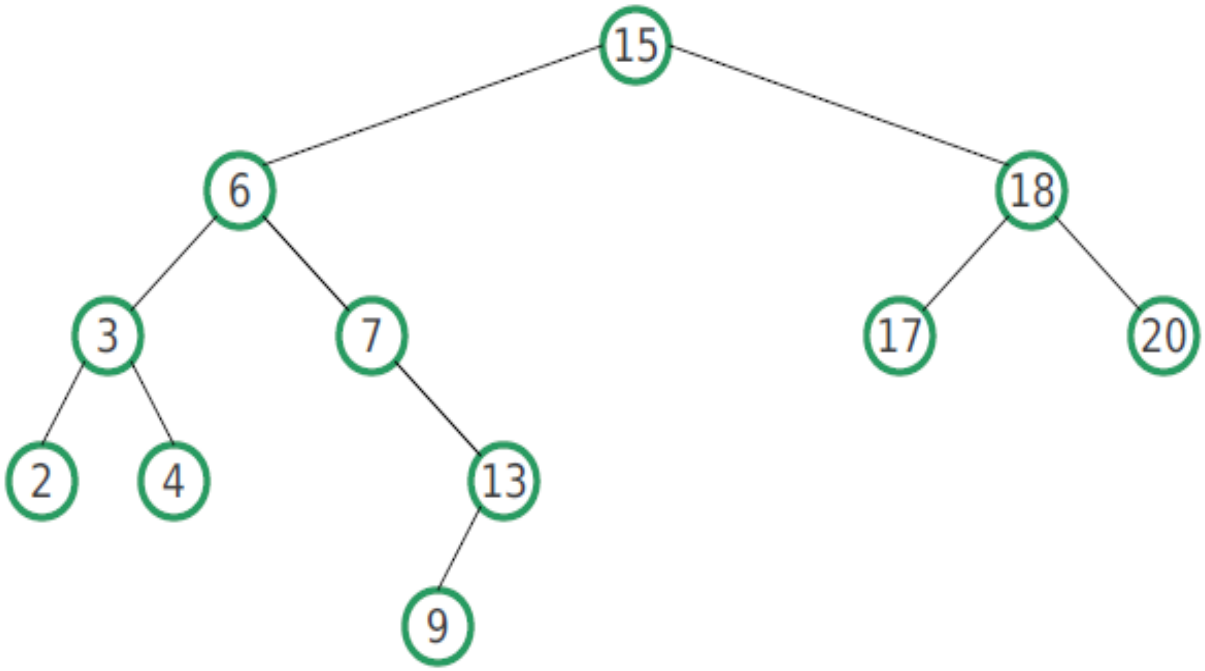

2 5 3 8 7 6

# Árvore de Pesquisa Binária
# Percurso

- Operação mais comum executada sobre uma APB é **procurar por uma chave** armazenada na árvore.

- Também serão apresentadas as operações de
  - MINIMUM,
  - MAXIMUM,
  - SUCESSOR,
  - PREDECESSOR.

# Árvore de Pesquisa Binária
# Percurso



# Árvore de Pesquisa Binária
# Consulta

## Como pesquisar

- Dado um ponteiro para a raiz da árvore e uma chave *k*, *BUSCA retorna um ponteiro* para um nó com chave *k*, *se existir; caso contrário, ele retorna NIL.*

TREE-SEARCH*(x, k)*
**if** *x* = NIL or *k* = *chave[x]* ***t*hen**
**return** *x*
**if** *k* < *chave[x]* **then**
return TREE-SEARCH*(esquerda[x], k)*
**else**
return TREE-SEARCH(direita[x]*, k)*
Tempo de execução *O(h).*

# Como pesquisar

- O mesmo procedimento pode ser escrito de forma iterativa

ITERATIVE-TREE-SEARCH*(x, k)*
**while** *x* ≠ NIL e *k* ≠ *chave[x]*
**if** *k* < *chave[x]* **then**
*x* <- *esquerda[x]*
**else**
*x* <- direita[x]
**return** *x*

# Mínimo e Máximo

- Um elemento em uma APB cuja chave é um mínimo sempre pode ser encontrado seguindo-se ponteiros filhos da ***esquerda*** desde a raiz até ser encontrado um valor **NIL.**

TREE-MINIMUM*(x)*
  **while** *esquerda[x]* ≠ NIL
    *x* <- *esquerda[x]*
  **return** *x*

Tempo de execução *O(h)*

# Mínimo e Máximo

- O pseudocódigo para encontrar o elemento máximo é simétrico

TREE-MAXIMUM*(x)*
  **while** *direita[x]* ≠ NIL
  *x* <- *direitaa[x]*
  **return** *x*

Tempo de execução *O(h)*

# Sucessor e Predecessor

- Se todas as chaves são distintas, o sucessor de um nó *x* é o nó com a menor chave maior que *chave[x].*

TREE-SUCESSOR*(x)*
**if** *direita[x]* ≠ NIL
**then return** TREE-MINIMUM*(direita[x])*
*y <- pai[x]*
**while** *y* ≠ NIL e *x = direita[y]* *//caso seja necessário subir*
*x <- y* *//para encontrar o sucessor*
*y <- pai[x]*
**return** *y*

Tempo de execução *O(h)*

# Árvore de Pesquisa Binária
# Inserção

- As operações de inserção e eliminação provocam mudanças no conjunto dinâmico representado por uma APB.

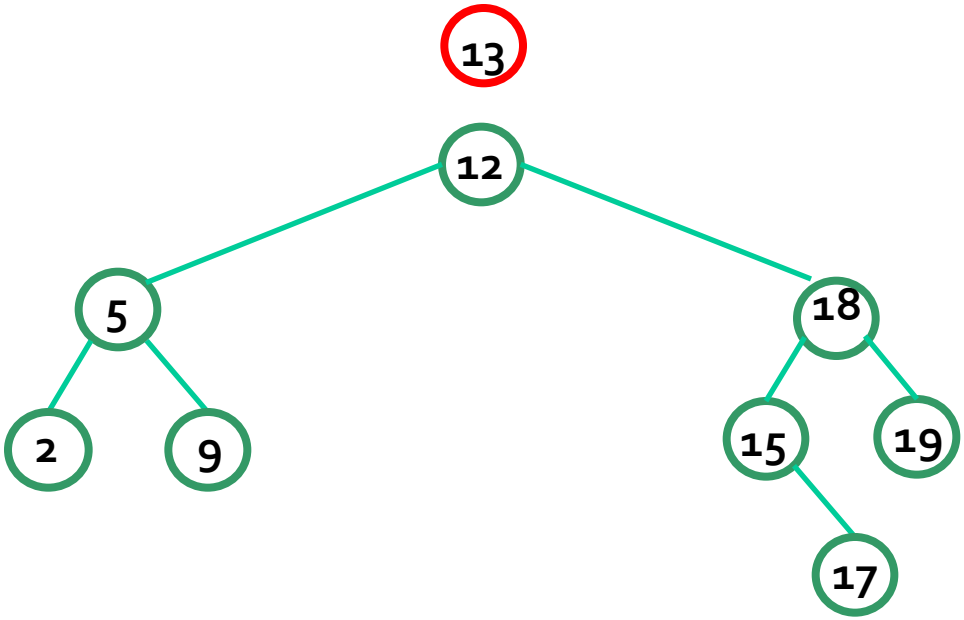
- A estrutura de dados deve ser modificada para refletir essa mudança, mas de tal modo que a propriedade de APB continue válida.

# Árvore de Pesquisa Binária
# Inserção

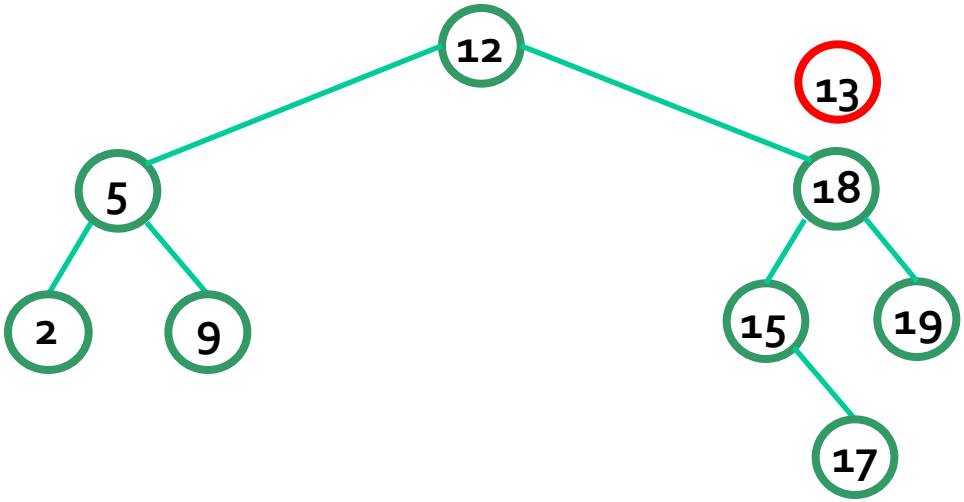
- Para inserir um novo valor *v* em uma APB *T*, o procedimento recebe a passagem de um nó *z* para o qual *chave[z]* = *v*, *esquerda[z]* = NIL e *direita[z]* = NIL.

- Ele modifica *T e alguns dos* campos de *z* de tal modo que *z* é inserido em uma posição apropriada na árvore.

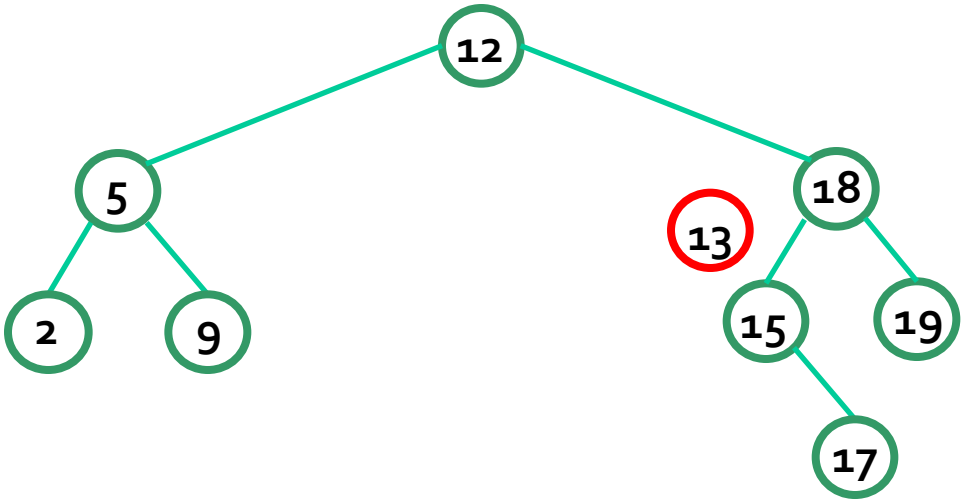
# Inserção



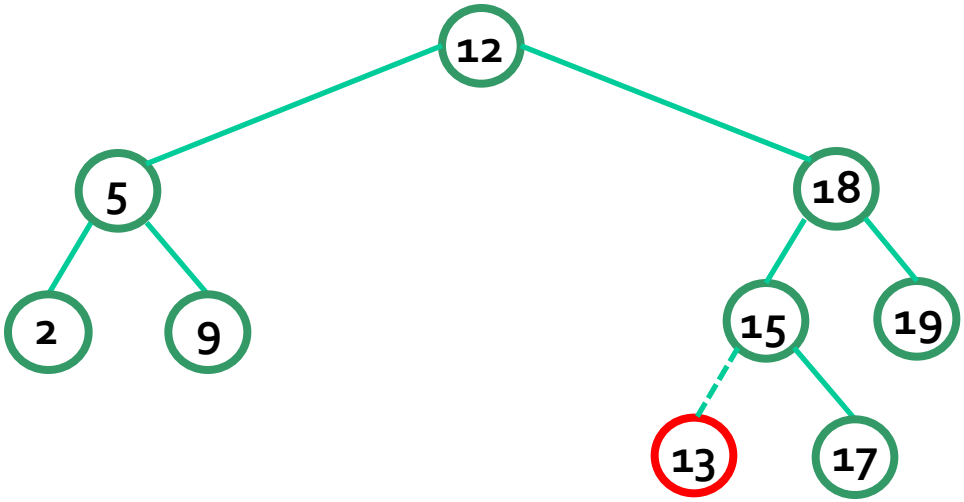
# Inserção



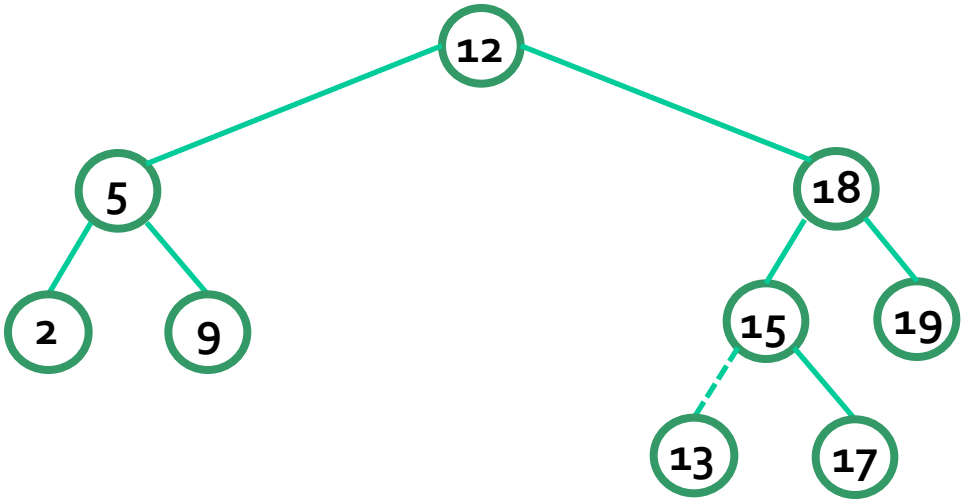
# Inserção



# Inserção



# Inserção



# Inserção

```
struct nodo * insert( element_type x, struct nodo * T ){
   if( T == NULL ){  /* Árvore era vazia */
       T = (struct nodo *) malloc ( sizeof (struct nodo) );
       if( T == NULL )
          error("Problema na alocação!");
       else{
          T->element = x;
          T->left = T->right = NULL;
        }
   }
   else{
       if( x < T->element )
          T->left = insert( x, T->left );
       else
          if( x > T->element )
              T->right = insert( x, T->right );
 /* se x já na árvore, faz nada */

return T;

}
```

# Árvore de Pesquisa Binária
# Eliminação

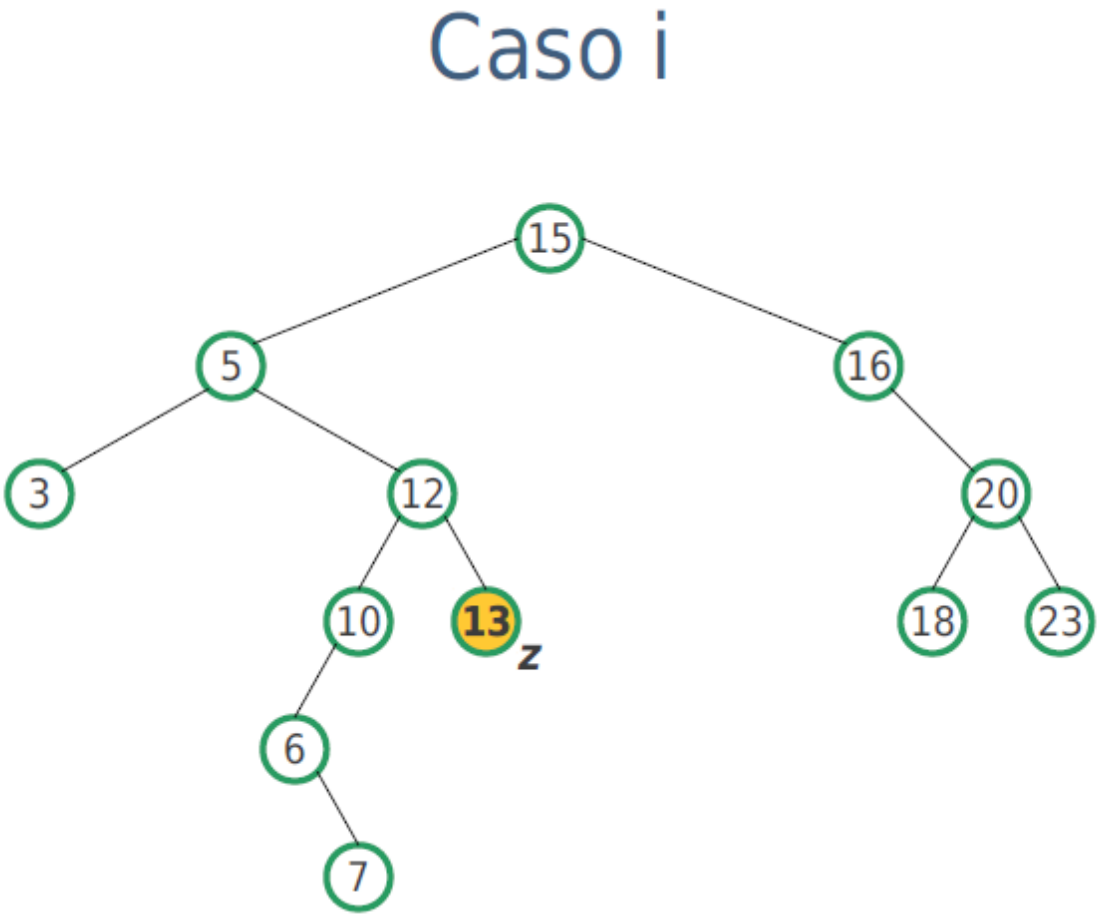
- O procedimento para **remover** um dado nodo *z de uma APB* toma como argumento um ponteiro para *z. O procedimento* considera os três casos a seguir:

- **i.** Se z não tem filhos, modifica-se pai(z) para substituir *z* com NIL como seu filho

- **ii.** Se o nó tem somente um filho, remove-se *z* criando um novo link entre seu no filho e seu pai.

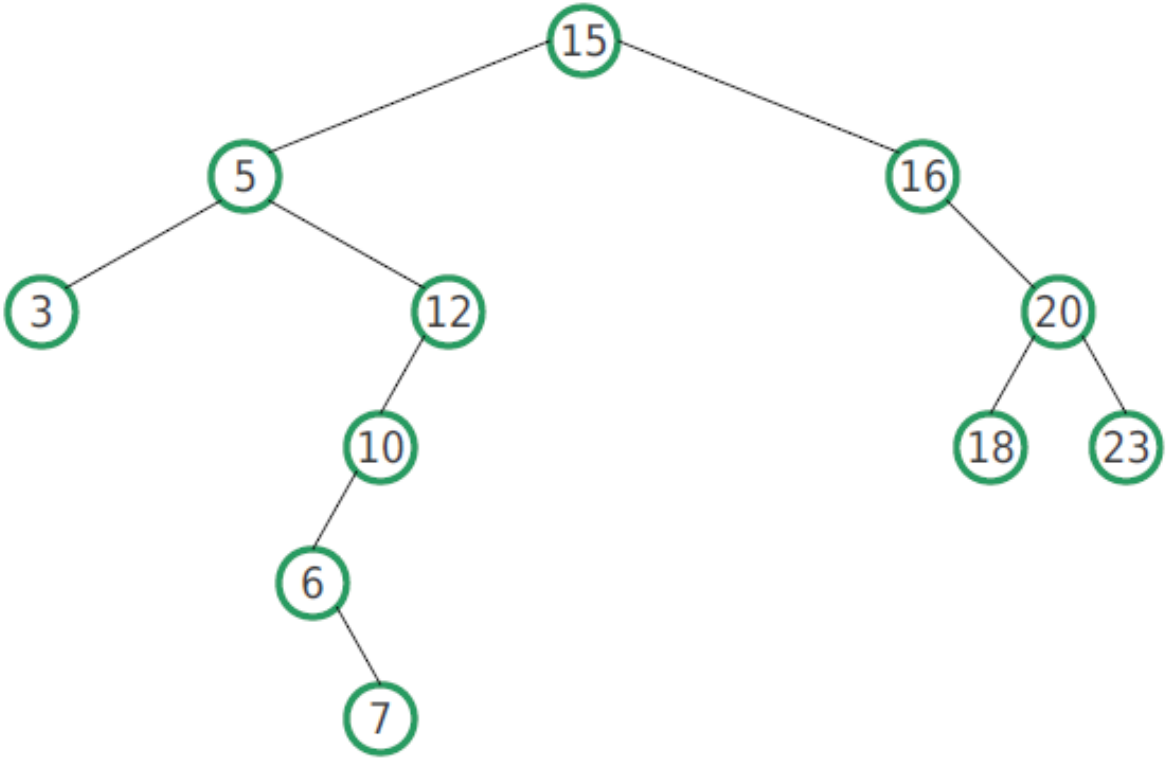
- **iii.** Se o nó tem dois nós filhos, move a chave e os dados satélite do sucessor *y* de *z*, que não tem nenhum filho à esquerda.

# Árvore de Pesquisa Binária
# Eliminação



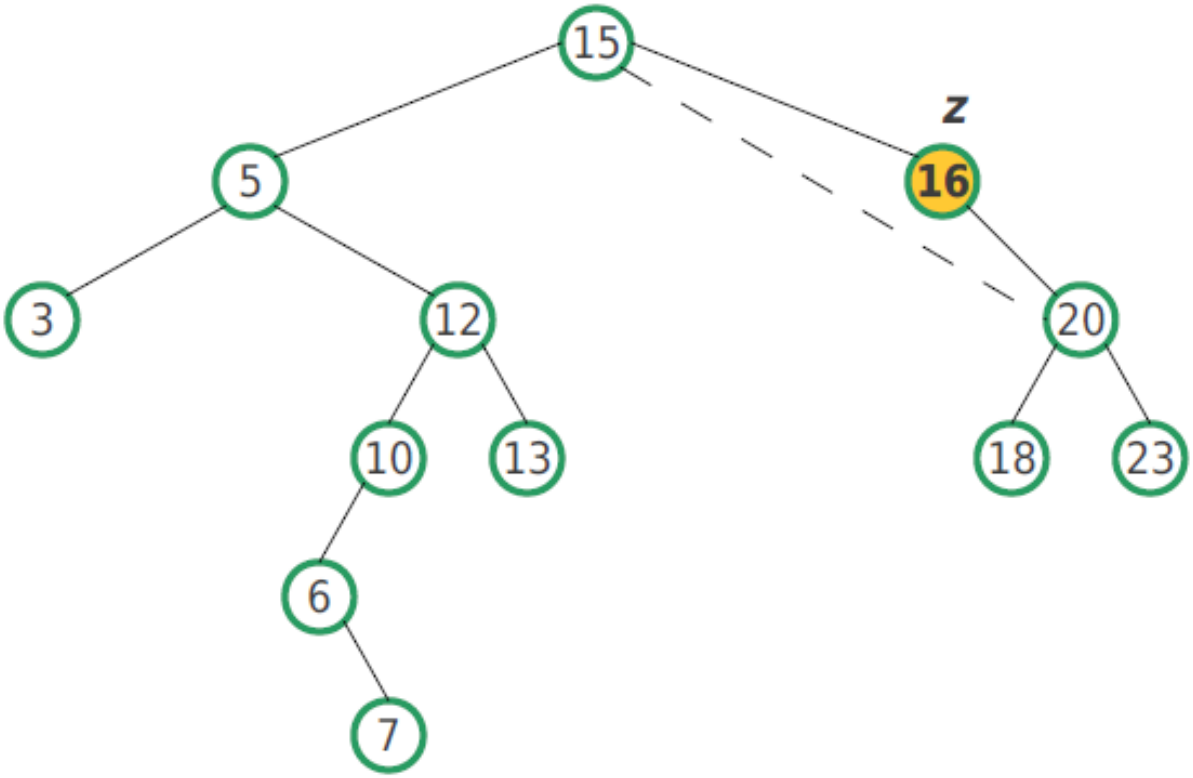
# Árvore de Pesquisa Binária
# Eliminação

## Caso i



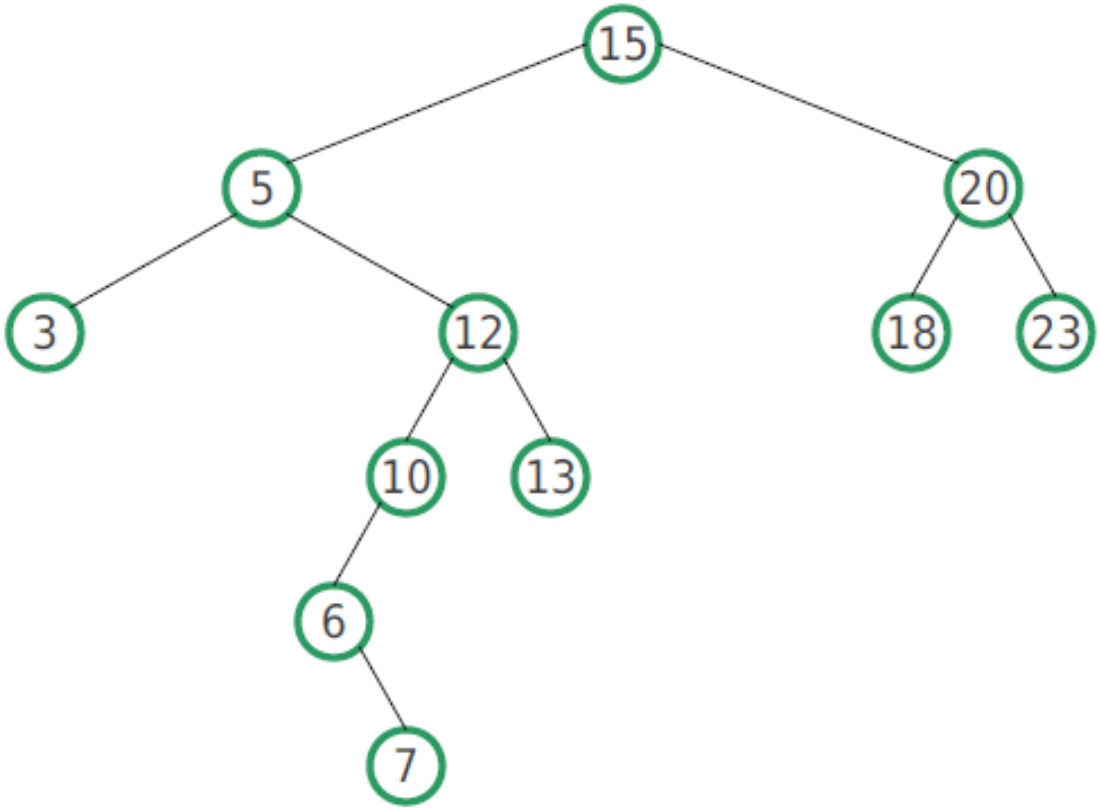
# Árvore de Pesquisa Binária
# Eliminação

## Caso ii



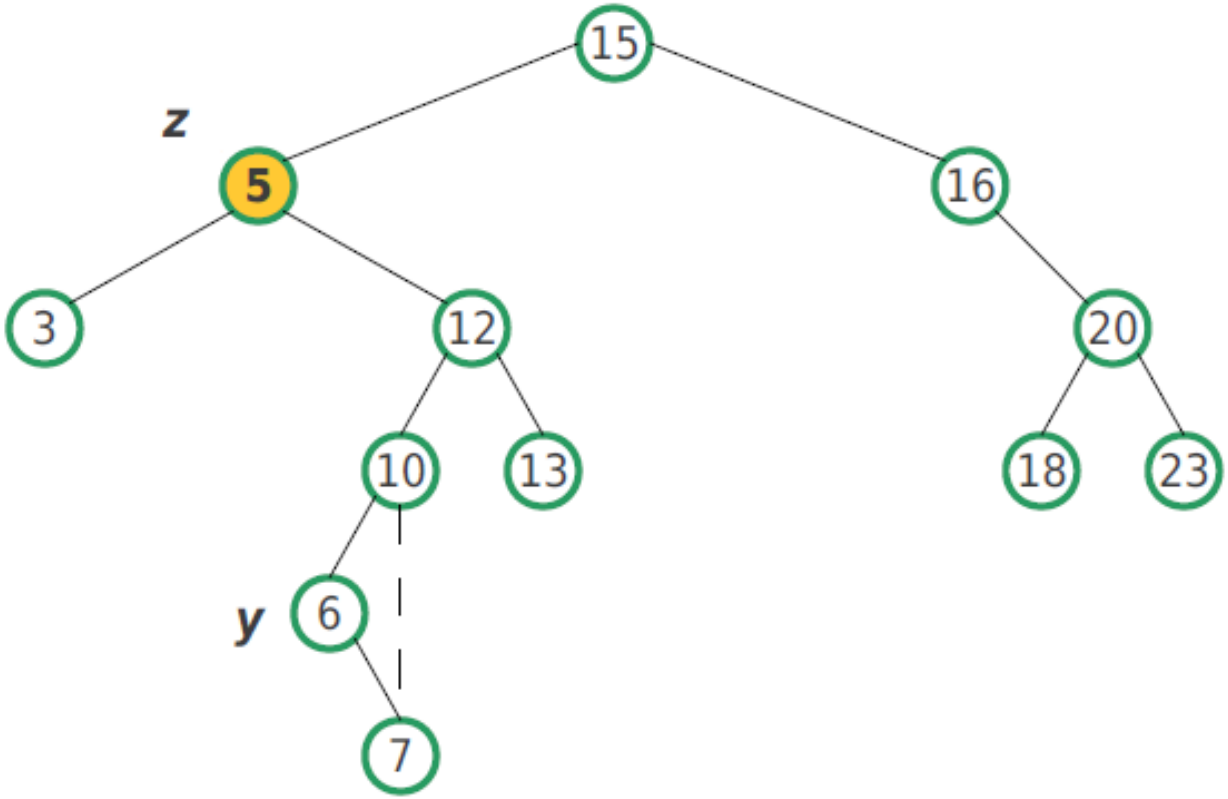
# Árvore de Pesquisa Binária
# Eliminação

## Caso ii



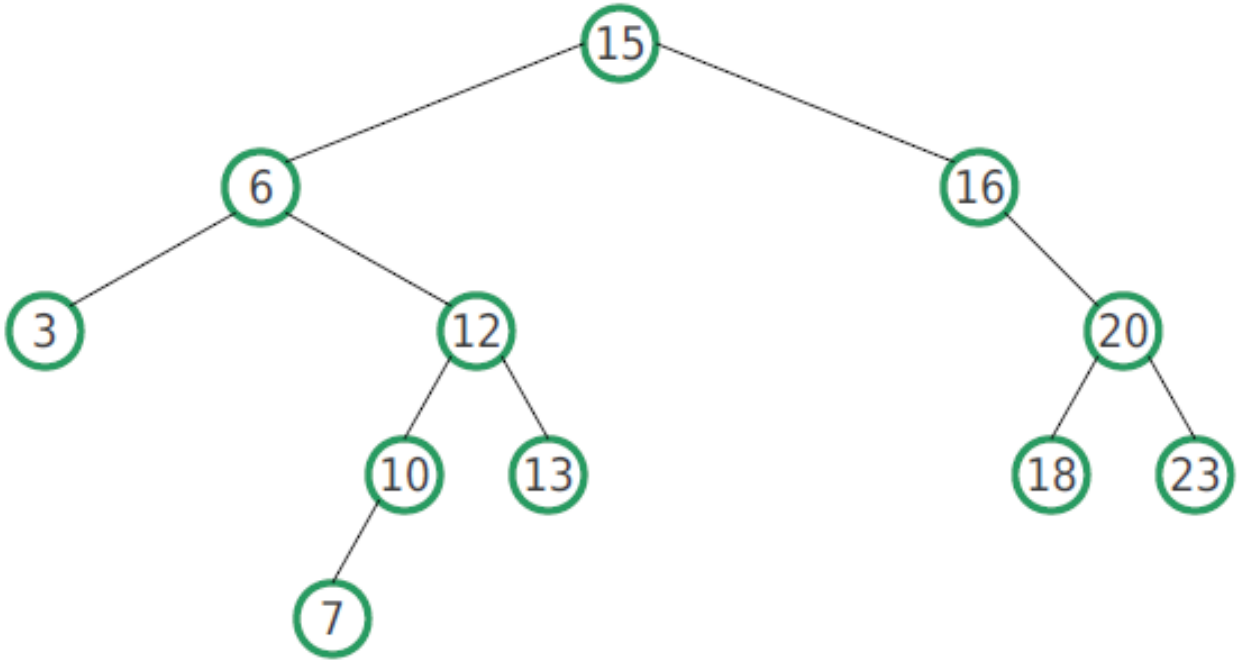
# Árvore de Pesquisa Binária
# Eliminação

## Caso iii



# Árvore de Pesquisa Binária
# Eliminação

## Caso iii



# Deleção

```
struct nodo * delete( element_type x, struct nodo * T ){
 struct nodo * tmp_cell, * child;

if( T == NULL )
   error("Elemento nao encontrado");
else
  if( x < T->element )  /* Esquerda */
     T->left = delete( x, T->left );
  else
     if( x > T->element )  /* Direita */
        T->right = delete( x, T->right );
     else /* Encontrou elemento */
       if( T->left && T->right ){ /* caso dois filhos */
        /* escolhe menor da subarvore direita */
        tmp_cell = find_min( T->right );
        T->element = tmp_cell->element;
        T->right = delete( T->element, T->right );
       }
     else{ /* filho único */
        tmp_cell = T;
        if( T->left == NULL ) /* Só filho à direita */
           child = T->right;
        if( T->right == NULL )      /* Só à direita */
           child = T->left;
        free( tmp_cell );

        return child;
     }
  return T;
}
```