Algoritmos e Estrutura de Dados II

Prof. Fellipe Guilherme Rey de Souza

Aula 09 – Operações em Árvore Binária

Agenda

- Inserção
- Remoção
- Busca
- Percurso
- Tamanho
- Altura

Operações em Árvore

- O TAD Árvore possui algumas operações básicas, que são:
  - **Inserir (*insert*)**: Adicionar um novo elemento na árvore.
  - **Remover (*remove*)**: Remover um elemento da árvore.
  - **Busca (*search*)**: Verificar (sem remover) se existe o item na árvore.
  - **Percurso (tree traversal)**: Percorrer a árvore
  - **Tamanho (*size*)**: Verifica o tamanho (quantidade de nós) da árvore.
  - **Altura (height)**: Verifica a altura da árvore

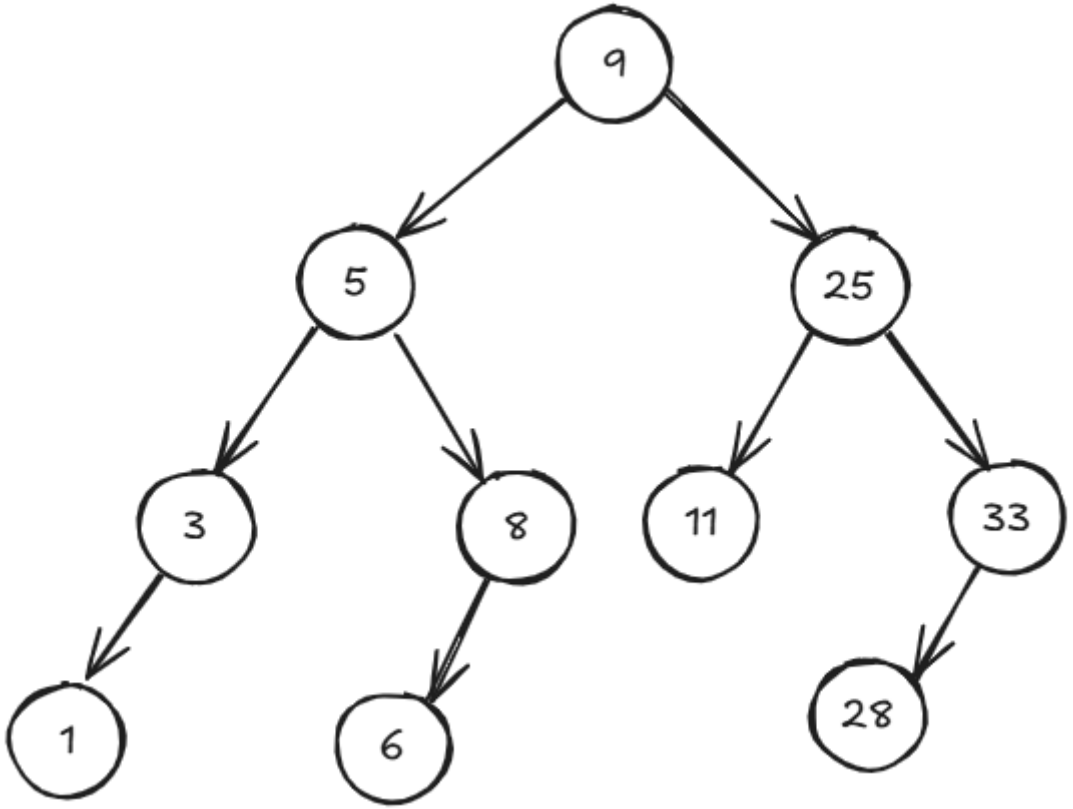
Operações em Árvore – Inserção

- Antes de discurtimos a inserção, é importante destacarmos que existem dois tipos de árvores binárias:
  i. Árvores binárias de busca e;
  ii. Árvores binárias genéricas.

Operações em Árvore – Inserção

- Em uma árvore binária de busca (ou árvore de pesquisa binária), a inserção segue uma regra específica. Para cada nó:
  - **Subárvore esquerda**: Os valores dos nós são menores ou iguais ao valor do nó pai.
  - **Subárvore direita**: Os valores dos nós são maiores ou iguais ao valor do nó pai.

Operações em Árvore – Inserção



Exemplo de uma árvore binária de busca

Operações em Árvore – Inserção

- Em uma árvore binária genérica (não necessariamente de busca), a inserção pode ser feita de várias maneiras.

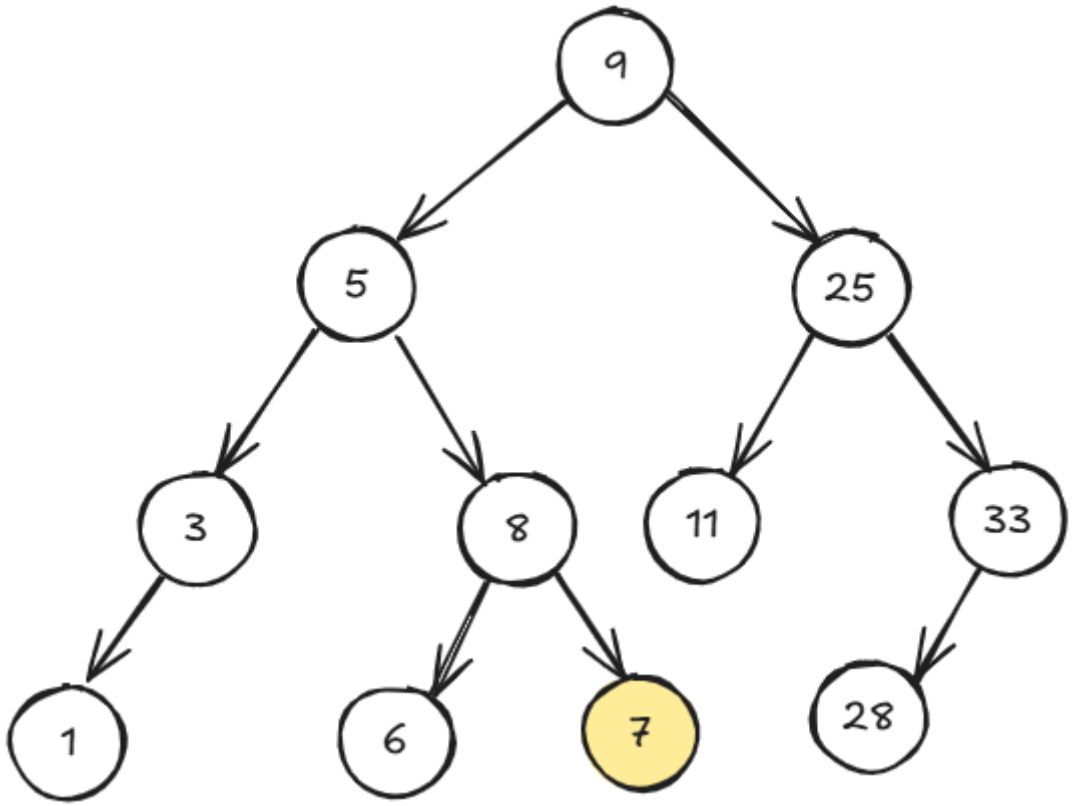
- Uma abordagem comum é usar a inserção por nível (ou seja, em nível de uma árvore binária completa).

Operações em Árvore – Inserção

- Passo-a-passo:
  i. Comece na raiz.
  ii. Vá para o próximo nível da árvore, sempre inserindo no primeiro local vazio disponível (da esquerda para a direita).

- Essa abordagem garante que a árvore será "completa", ou seja, todos os níveis estarão preenchidos antes de começar um novo nível.

Operações em Árvore – Inserção



Exemplo de uma árvore binária genérica

Operações em Árvore – Inserção

- Durante esta aula, iremos utilizar a árvore binária de busca.

- No exemplo a seguir, vamos inserir na nossa árvore binária de busca, os seguintes elementos (em ordem): 4, 6, 7, 2, 3, 1 e 5.

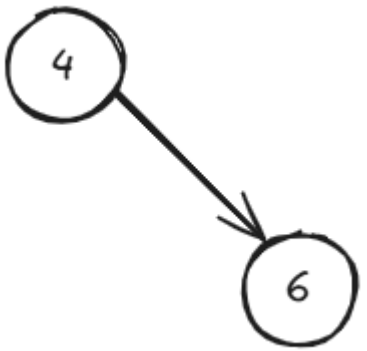
Operações em Árvore – Inserção

- Inserir (4, 6, 7, 2, 3, 1 e 5):
  - Ação: Inserir o número 4

Subárvore esquerda: Valores menores que o pai
Subárvore direita: Valores maiores que o pai

Operações em Árvore – Inserção

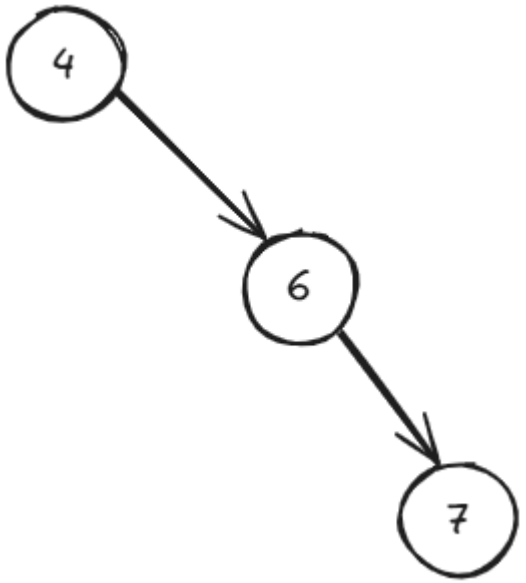
- Inserir (4, 6, 7, 2, 3, 1 e 5):
  - ~~Ação: Inserir o número 4~~
  - Ação: Inserir o número 6



Subárvore esquerda: Valores menores que o pai
Subárvore direita: Valores maiores que o pai

Operações em Árvore – Inserção

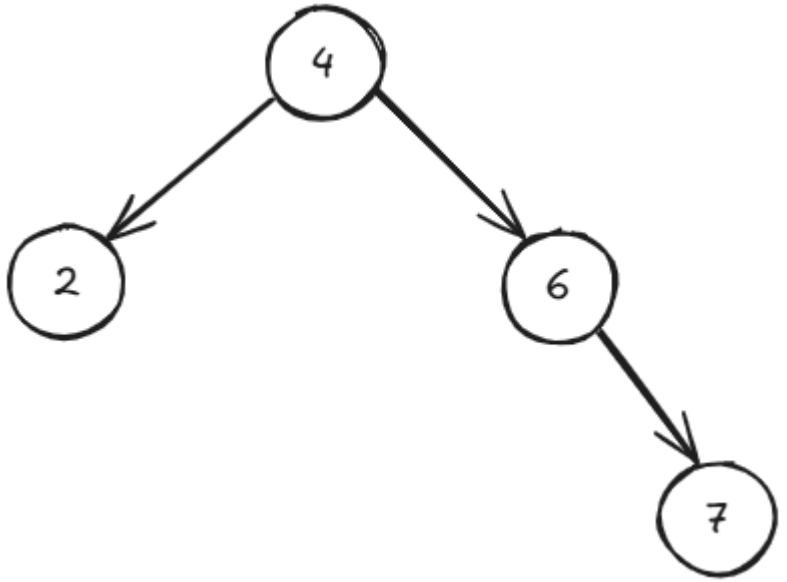
- Inserir (4, 6, 7, 2, 3, 1 e 5):
  - ~~Ação: Inserir o número 4~~
  - ~~Ação: Inserir o número 6~~
  - Ação: Inserir o número 7



Subárvore esquerda: Valores menores que o pai
Subárvore direita: Valores maiores que o pai

Operações em Árvore – Inserção

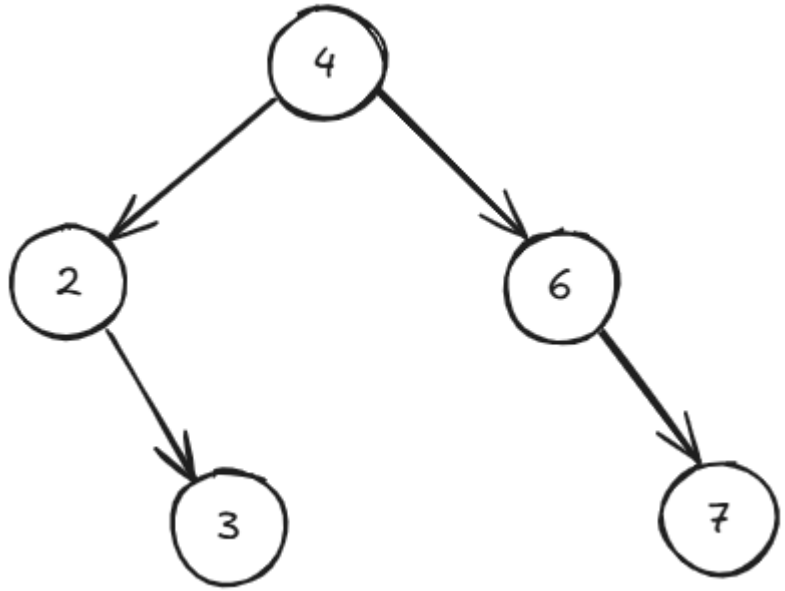
- Inserir (4, 6, 7, 2, 3, 1 e 5):
  - ~~Ação: Inserir o número 4~~
  - ~~Ação: Inserir o número 6~~
  - ~~Ação: Inserir o número 7~~
  - Ação: Inserir o número 2



Subárvore esquerda: Valores menores que o pai
Subárvore direita: Valores maiores que o pai

Operações em Árvore – Inserção

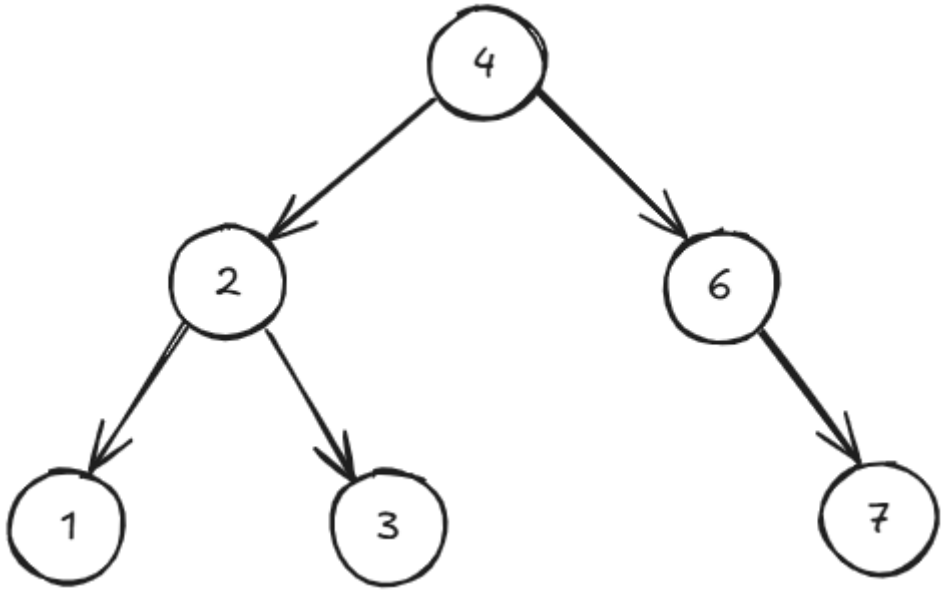
- Inserir (4, 6, 7, 2, 3, 1 e 5):
  - ~~Ação: Inserir o número 4~~
  - ~~Ação: Inserir o número 6~~
  - ~~Ação: Inserir o número 7~~
  - ~~Ação: Inserir o número 2~~
  - Ação: Inserir o número 3



Subárvore esquerda: Valores menores que o pai
Subárvore direita: Valores maiores que o pai

Operações em Árvore – Inserção

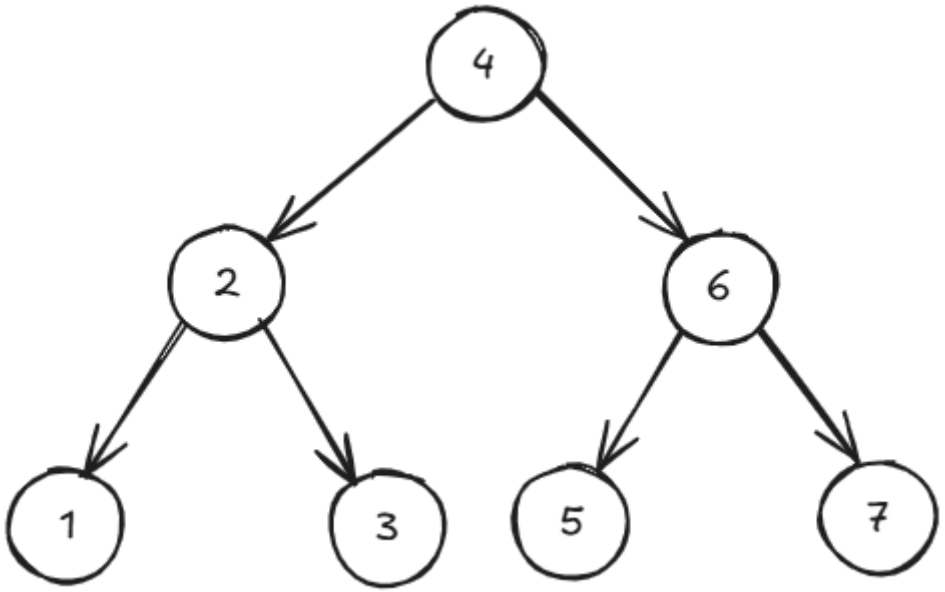
- Inserir (4, 6, 7, 2, 3, 1 e 5):
  - ~~Ação: Inserir o número 4~~
  - ~~Ação: Inserir o número 6~~
  - ~~Ação: Inserir o número 7~~
  - ~~Ação: Inserir o número 2~~
  - ~~Ação: Inserir o número 3~~
  - Ação: Inserir o número 1



Subárvore esquerda: Valores menores que o pai
Subárvore direita: Valores maiores que o pai

Operações em Árvore – Inserção

- Inserir (4, 6, 7, 2, 3, 1 e 5):
  - ~~Ação: Inserir o número 4~~
  - ~~Ação: Inserir o número 6~~
  - ~~Ação: Inserir o número 7~~
  - ~~Ação: Inserir o número 2~~
  - ~~Ação: Inserir o número 3~~
  - ~~Ação: Inserir o número 1~~
  - Ação: Inserir o número 5



Subárvore esquerda: Valores menores que o pai
Subárvore direita: Valores maiores que o pai

Operações em Árvore – Inserção

```
1  FUNCTION insert(raiz, valor)
2      Cria noh
3      SE raiz eh nula
4          ENTAO RETORNE noh
5      FIM-SE
6      SE valor < raiz->info
7          ENTAO raiz->esquerda ← insert(raiz->esquerda, valor)
8      FIM-SE
9      SE valor > raiz->info
10         ENTAO raiz->direita ← insert(raiz->direita, valor)
11     FIM-SE
12     RETORNE raiz
13 END-FUNCTION
```

Operações em Árvore – Remoção

- Sabemos que pela definição de árvore binária de busca, todos os elementos à esquerda do nó são necessariamente menores do que ele.

- Analogamente, todos os elementos à direita do nó são necessariamente maiores do que ele.

Operações em Árvore – Remoção

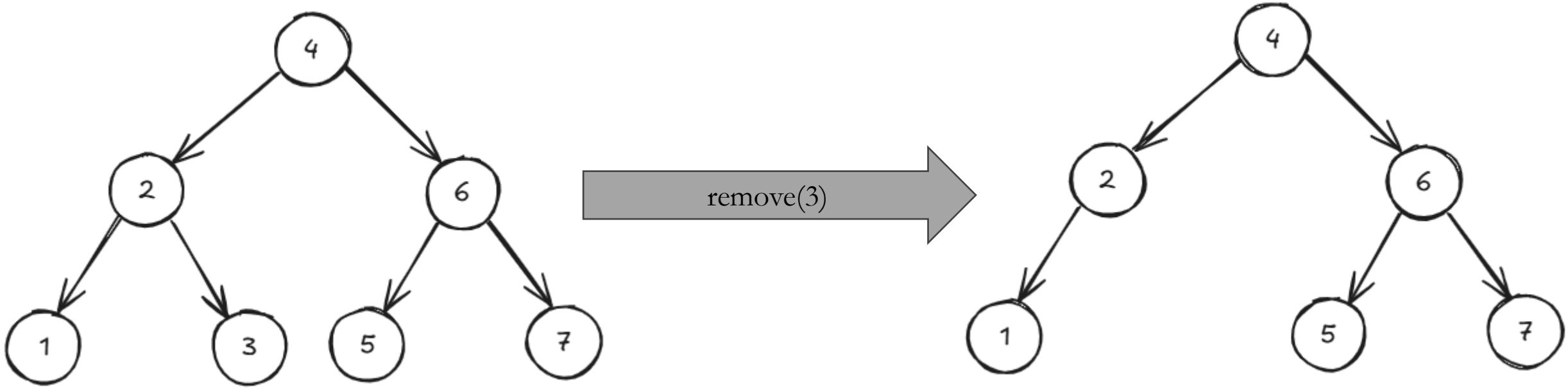
- Ao remover um elemento, nós precisamos que essa propriedade (menores valores à esquerda e maiores valores à direita) seja garantida.

- Assim, nós temos três casos de remoção em uma árvore binária de busca:
  i. Remoção de um nó sem filhos
  ii. Remoção de um nó com um filho
  iii. Remoção de um nó com dois filhos

Operações em Árvore – Remoção

i. Remoção de um nó sem filhos

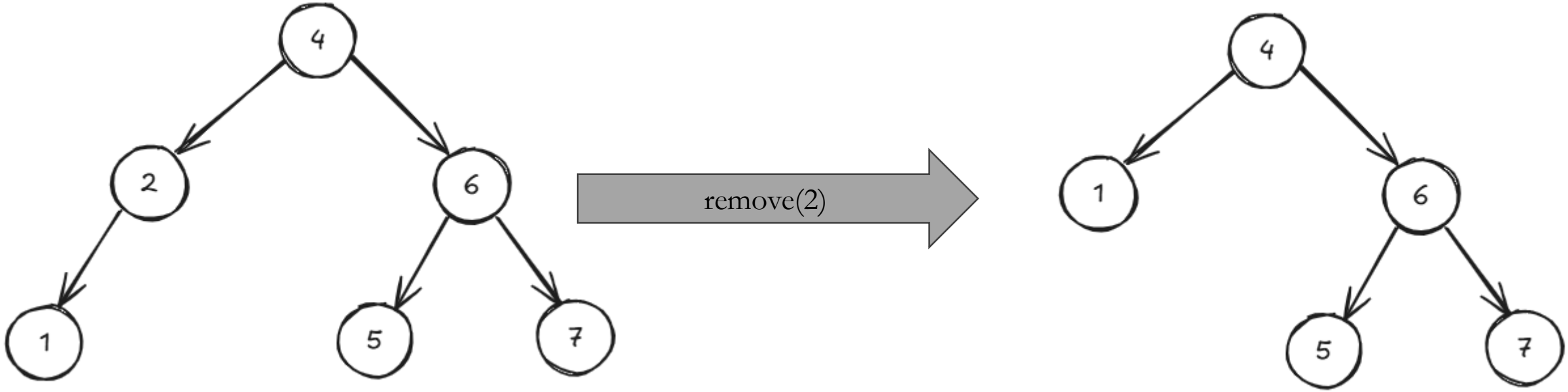
- Se o nó que você quer remover não tem filhos (é uma folha), você simplesmente pode remover o nó sem mais alterações, pois ele não afeta a estrutura da árvore.



Operações em Árvore – Remoção

ii. Remoção de um nó com um filho

- Se o nó a ser removido tem apenas um filho, basta substituir o nó pelo seu único filho. Qualquer referência ao nó removido é redirecionada para o filho.



Operações em Árvore – Remoção

iii. Remoção de um nó com dois filhos

- Este é o caso mais complexo, e a remoção exige uma estratégia especial para garantir que a árvore de busca binária continue válida.

- Quando um nó tem dois filhos, você precisa encontrar um valor que possa substituir o nó removido sem violar as propriedades da árvore de busca binária.

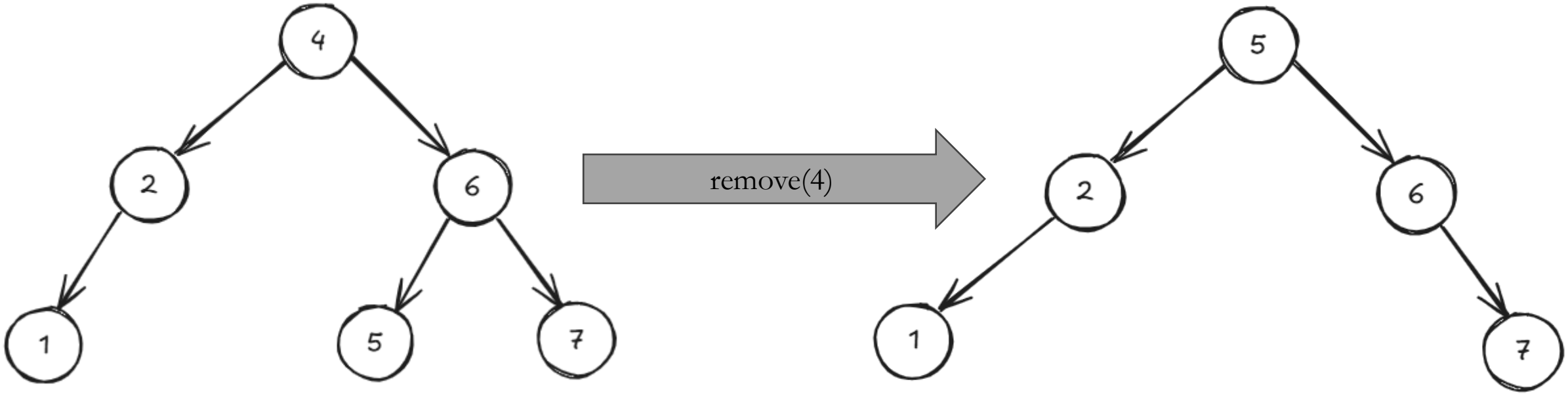
Operações em Árvore – Remoção

iii. Remoção de um nó com dois filhos (*cont.*)

- O valor substituto pode ser:
  - O maior valor da subárvore esquerda (o nó mais à direita da subárvore esquerda), ou;
  - O menor valor da subárvore direita (o nó mais à esquerda da subárvore direita).

- A escolha mais comum é o sucessor **em-ordem** (o menor valor da subárvore direita).

Operações em Árvore – Remoção



Em-ordem: 1, 2, 4, 5, 6, 7

Operações em Árvore – Remoção

```
FUNCTION remove(raiz, valor)
    SE raiz eh nula
        ENTAO RETORNE NULL
    FIM-SE
    SE valor < raiz->valor
        ENTAO raiz->esquerda ← remove(raiz->esquerda, valor)
        SENAO SE valor > raiz->valor
                ENTÃO raiz->direita ← remove(raiz->direita, valor)
                SENÃO SE raiz->esquerda = NULL E raiz->direita = NULL
                        ENTÃO RETORNE NULL
                      FIM-SE

                      SE raiz->esquerda = NULL
                        ENTÃO RETORNE raiz->direita
                      FIM-SE
                      SE raiz->direita = NULL
                        ENTÃO RETORNE raiz->esquerda
                      FIM-SE

                      SE raiz->esquerda != NULL E raiz->direita != NULL
                        ENTÃO sucessor ← encontraSucessorEmOrdem(raiz->direita)
                              raiz->valor = sucessor->valor
                              raiz->direita = remover(raiz->direita, valor)
                      FIM-SE
              FIM-SE
    FIM-SE
END-FUNCTION
```

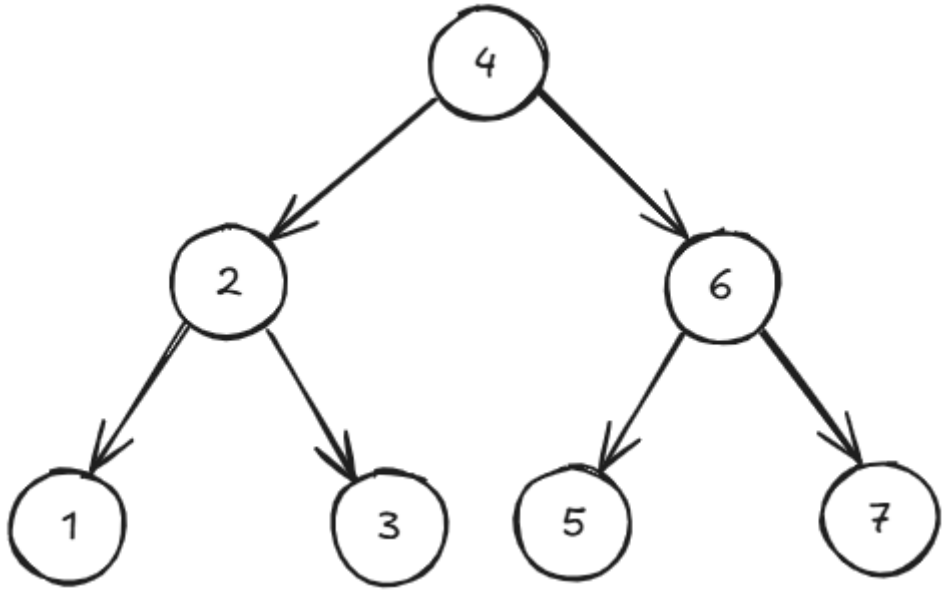
Operações em Árvore – Busca

- A busca em uma árvore binária de busca é eficiente devido à maneira como os elementos são organizados.

- Em uma árvore binária de busca, para cada nó:
  - Todos os valores na subárvore esquerda são menores que o valor do nó.
  - Todos os valores na subárvore direita são maiores que o valor do nó.

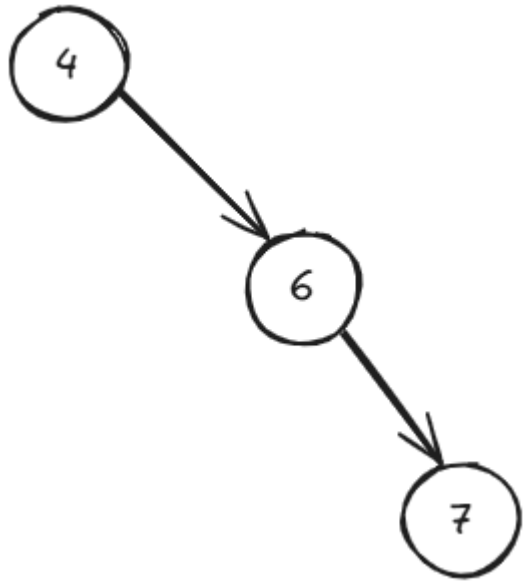
Operações em Árvore – Busca

- Esse comportamento permite que a busca seja feita de maneira eficiente, em tempo **O(log n)** no melhor e no pior caso para árvores balanceadas.



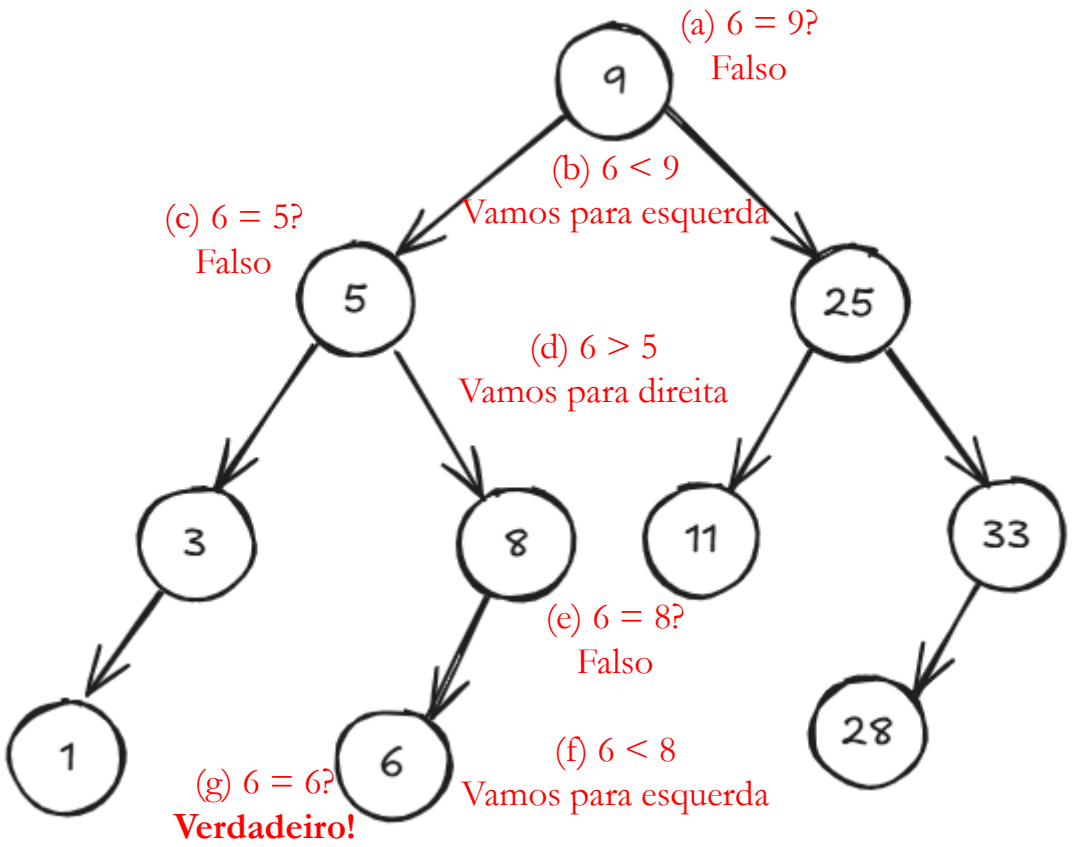
Operações em Árvore – Busca

- Caso a árvore não seja balanceada, a busca pode se comportar como uma lista encadeada, ou seja, **O(n)**.



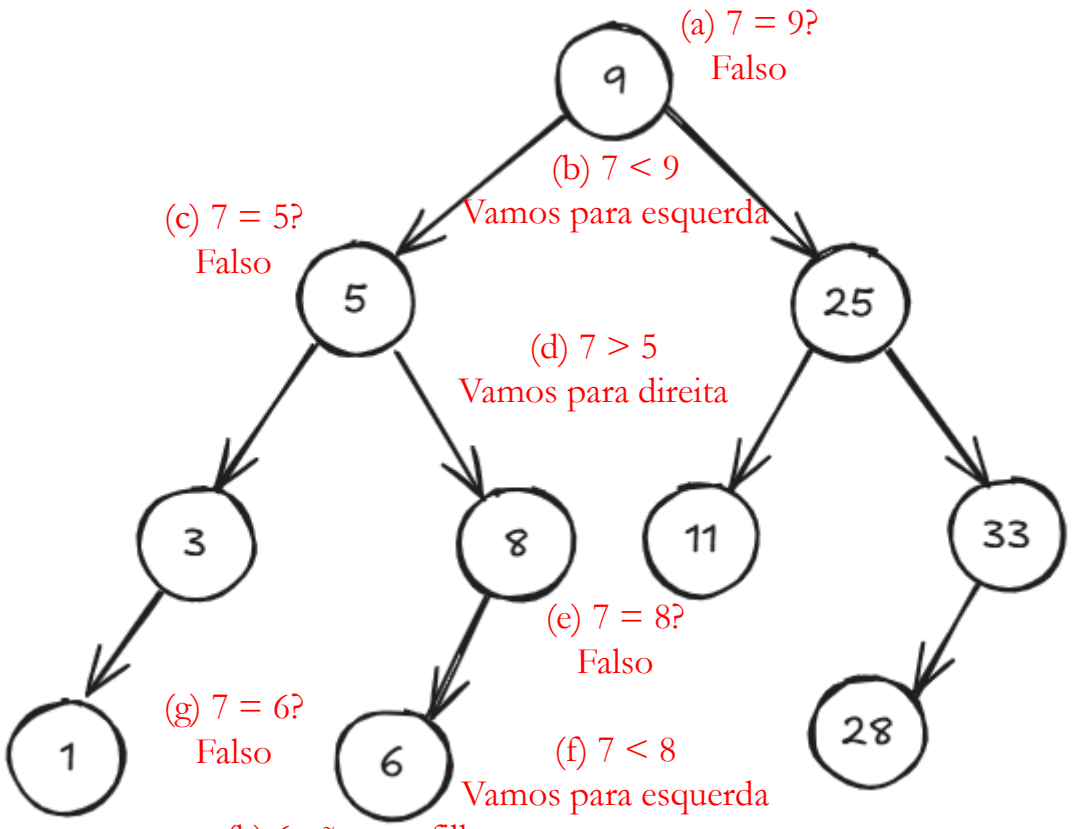
Operações em Árvore – Busca

- Exemplo 01: Buscar o elemento 6 na árvore



Operações em Árvore – Busca

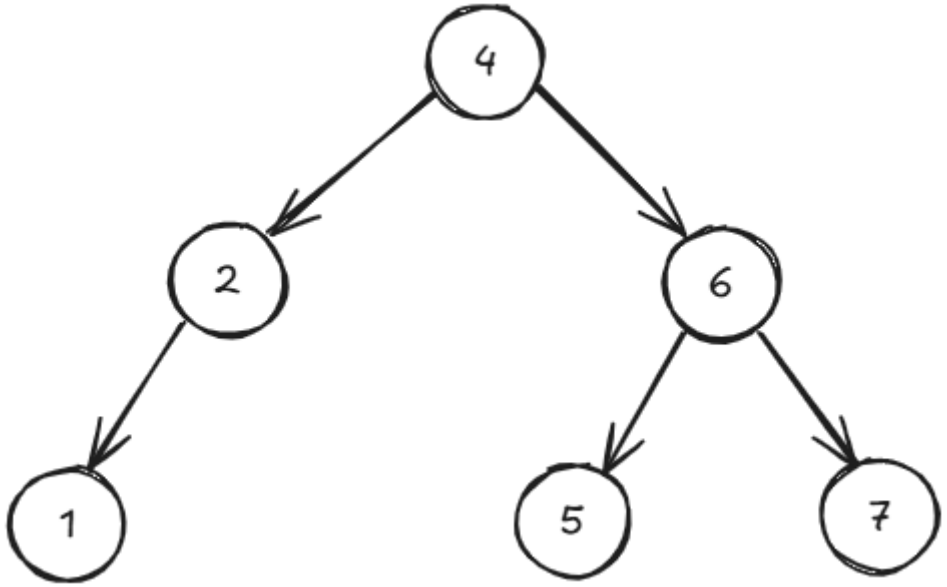
- Exemplo 02: Buscar o elemento 7 na árvore



Operações em Árvore – Busca

```
1   FUNCTION search(raiz, valor)
2       SE raiz = NULL
3           ENTAO RETORNE FALSE
4       FIM-SE
5
6       SE raiz->valor = valor
7           ENTAO RETORNE VERDADEIRO
8       FIM-SE
9
10      SE raiz->valor > valor
11          ENTAO search(raiz->esquerda, valor)
12      FIM-SE
13
14      SE raiz->valor < valor
15          ENTAO search(raiz->direita, valor)
16      FIM-SE
17  END-FUNCTION
```

Operações em Árvore – Percurso



Pré-ordem: 4, 2, 1, 6, 5, 7

Em-ordem: 1, 2, 4, 5, 6, 7

Pós-ordem: 1, 2, 5, 7, 6, 4

Operações em Árvore – Percurso

```
pre-ordem(pt){
    visita(pt);
    se pt^.esq ≠ λ então pre-ordem(pt^.esq)
    se pt^.dir ≠ λ então pre-ordem(pt^.dir)
}
```

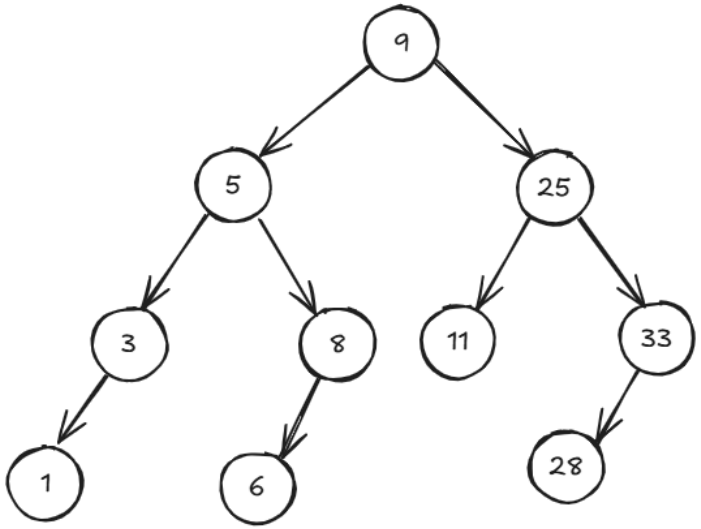
```
em-ordem(pt){
    se pt^.esq ≠ λ então em-ordem(pt^.esq)
    visita(pt);
    se pt^.dir ≠ λ então em-ordem(pt^.dir)
}
```

```
pos-ordem(pt){
    se pt^.esq ≠ λ então pos-ordem(pt^.esq)
    se pt^.dir ≠ λ então pos-ordem(pt^.dir)
    visita(pt);
}
```

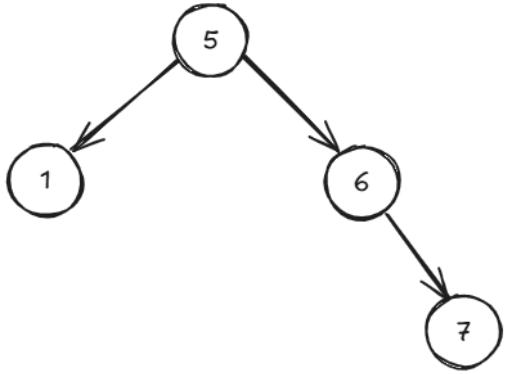
```
visita(pt){
    imprime(pt^.info);
}
```

Operações em Árvore – Tamanho

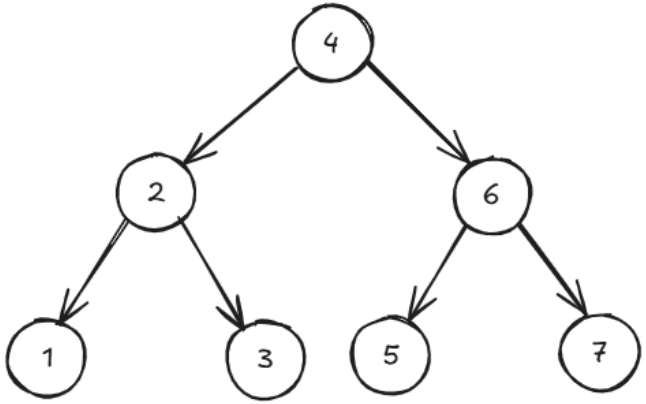
- O tamanho de uma árvore binária de busca é definido pelo número total de nós que uma árvore possui.



Tamanho: 10



Tamanho: 4



Tamanho: 7

Operações em Árvore – Tamanho

```
1   FUNCTION size(raiz)
2       SE raiz = NULL
3           ENTAO RETORNE 0
4       FIM-SE
5
6       tamanhoEsquerda = size(raiz->esquerda)
7       tamanhoDireita = size(raiz->direita)
8
9       RETORNE 1 + tamanhoEsquerda + tamanhoDireita
10  END-FUNCTION
```

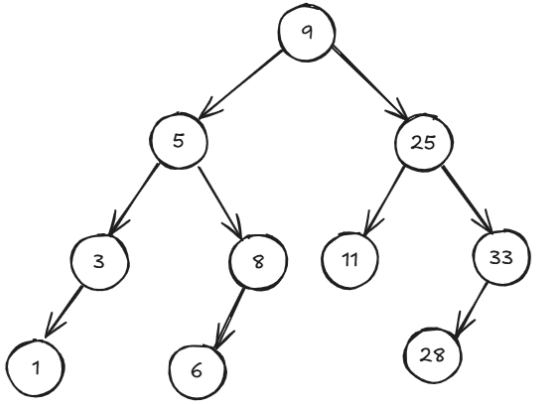
Operações em Árvore – Altura

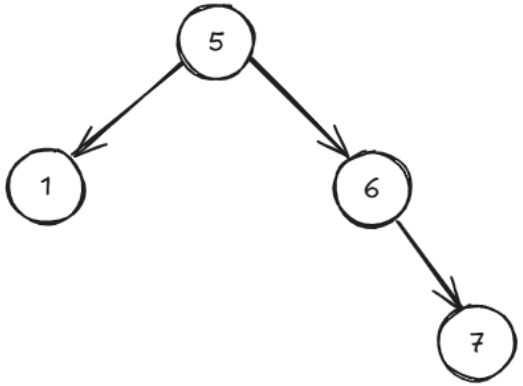
- A **altura** de uma árvore binária é o número máximo de arestas (ou níveis) que você precisa percorrer para ir da raiz até o nó mais distante (ou folha) da árvore.

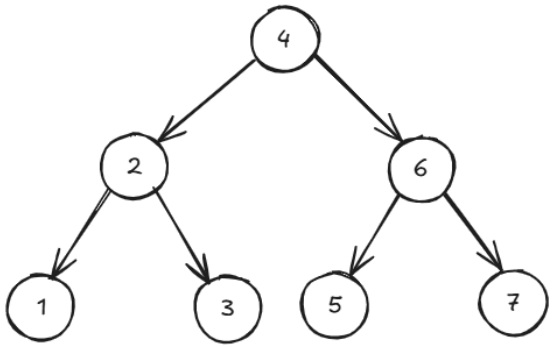
- A altura é uma medida importante para entender a profundidade da árvore e a eficiência de suas operações de busca, inserção e remoção.

Operações em Árvore – Altura

NULL







Altura: 0

Altura: 4

Altura: 3

Altura: 3

Operações em Árvore – Altura

```
1   FUNCTION height(raiz)
2       SE raiz = NULL
3           ENTAO RETORNE 0
4       FIM-SE
5
6       alturaEsquerda = height(raiz->esquerda)
7       alturaDireita = height(raiz->direita)
8
9       RETORNE 1 + max(alturaEsquerda, alturaDireita)
10  END-FUNCTION
```

Algoritmos e Estrutura de Dados II

Prof. Fellipe Guilherme Rey de Souza

Aula 09 – Operações em Árvore Binária