Curso Estruturas de Dados e Algoritmos Expert

Prof. Nelio Alves

Árvores

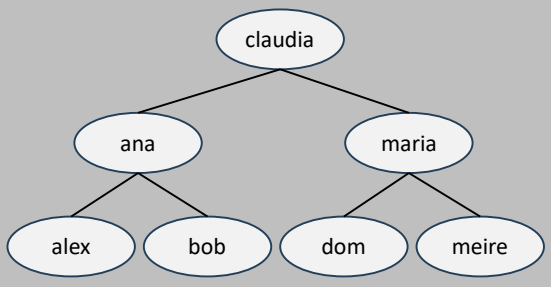
DEVSUPERIOR

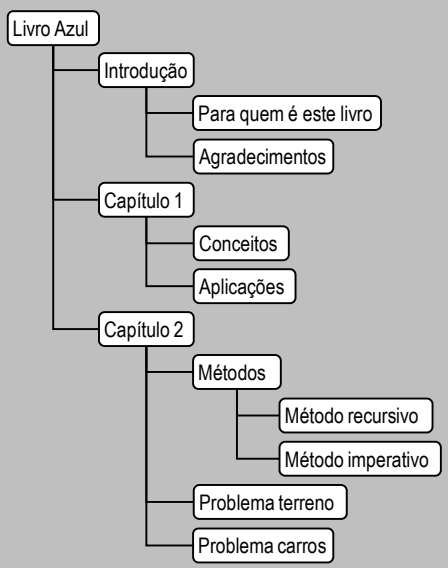
https://devsuperior.com.br

1

Visão geral

Neste módulo vamos apresentar a estrutura de dados árvore, os conceitos relacionados às árvores, e vamos explorar algumas implementações.



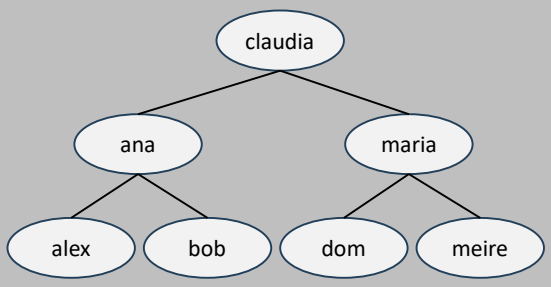


2

Árvore

Uma árvore é uma estrutura de dados não linear que organiza elementos de forma **hierárquica** em nós.

Algumas implementações de árvores permitem criar coleções ordenadas com operações eficientes de busca, inserção e remoção: O(log n).



3

Aplicações comuns de árvores

Sistemas de arquivos (diretórios (pastas) e arquivos) e menus.

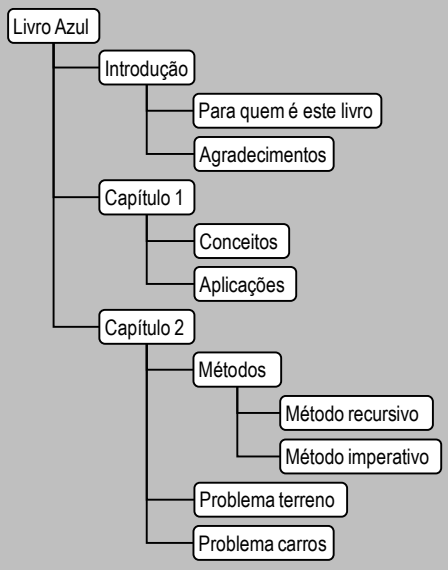
Interfaces gráficas, páginas web (HTML) e estrutura de documentos.

Compiladores: o processo de *parsing* organiza os símbolos de uma linguagem em uma árvore de derivação sintática.

Sistemas de banco de dados, sistemas com buscas e atualizações frequentes: árvores AVL, árvores rubro-negras, árvores B e B+.

Árvores de decisão: em sistemas de machine learning, análise de dados.

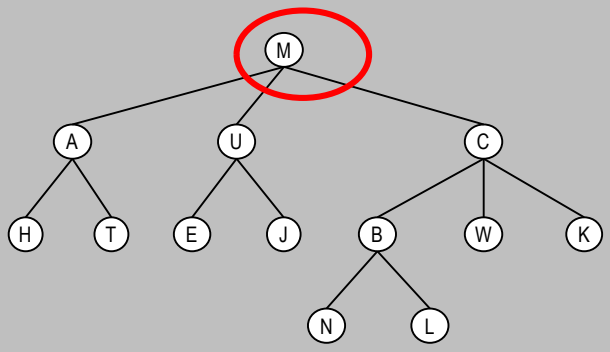
Árvores geradoras: em sistemas de otimização, logística, cobertura de caminhos.



4

Raiz

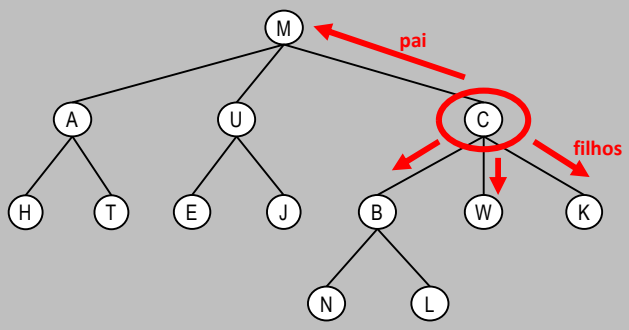
É o nó do topo da árvore.



5

Nós filhos, nó pai

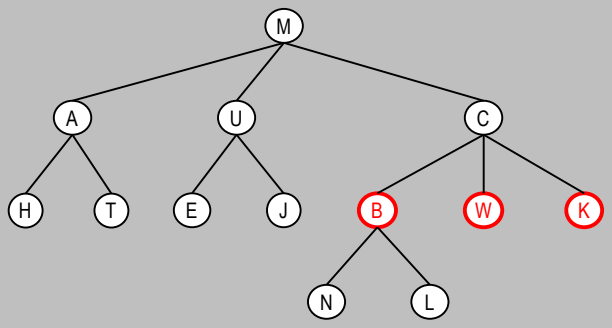
Cada nó pode possuir zero ou mais filhos.
Cada nó possui um pai, exceto a raiz, que não possui pai.



6

Nós irmãos

São nós filhos do mesmo pai.

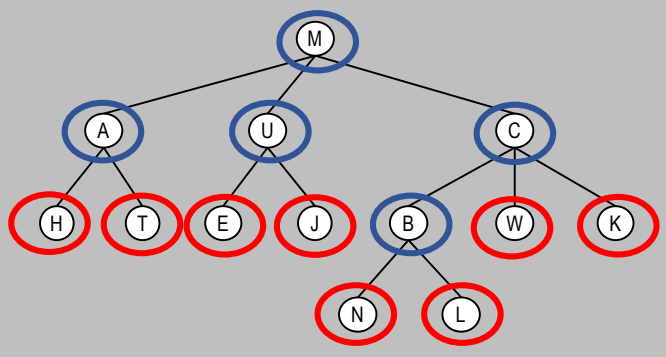


7

Nós externos (folhas) e internos

Nós **externos** são nós que não possuem filhos.
Também chamados de **folhas**.

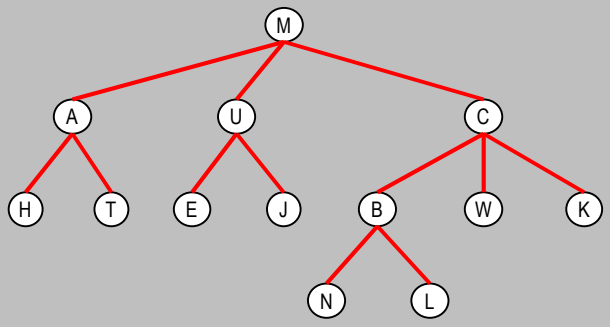
Nós **internos** são nós que possuem pelo menos um filho.



8

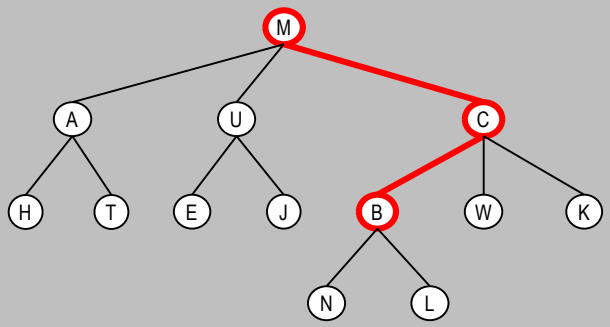
Arestas

São as conexões entre os nós.



Caminho

Uma sequência de **um** ou mais nós conectados por arestas.



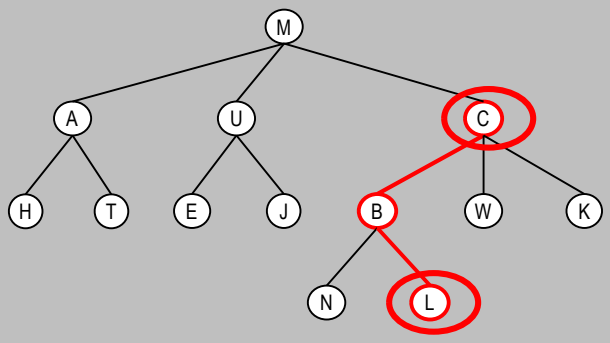
No exemplo: caminho M-C-B, ou B-C-M dependendo da origem adotada.

Ancestral, descendente

Um nó X é ancestral de um nó Y, se existe um caminho da raiz da árvore até Y que passe por X.

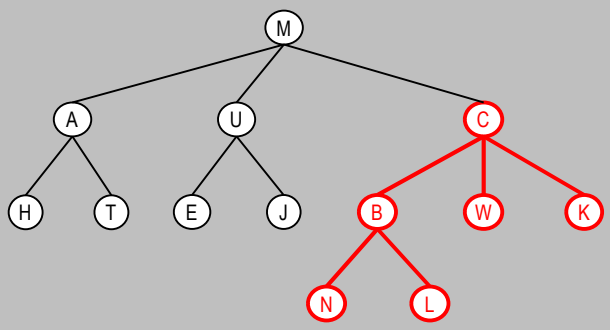
Um nó X é descendente de um nó Y, se e somente se Y for ancestral de X.

Todo nó é ancestral e descendente dele mesmo.



Subárvore enraizada em um nó

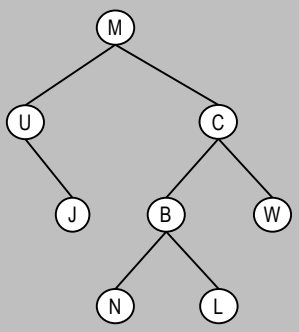
A subárvore enraizada em um nó X é a árvore que consiste em todos os descendentes de X, incluindo o próprio X.



Árvore binária

Uma árvore é binária se cada um de seus nós possui no máximo dois filhos.

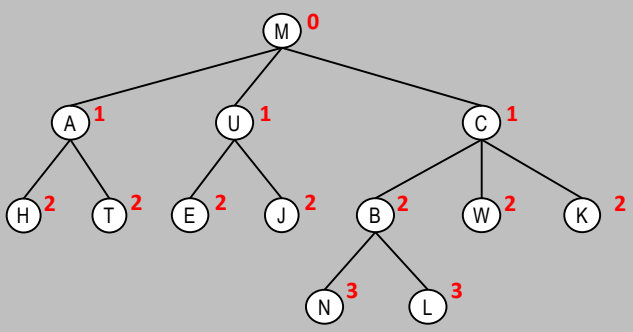
Tipicamente os filhos são definidos como filho à esquerda e filho à direita.



13

Profundidade de um nó

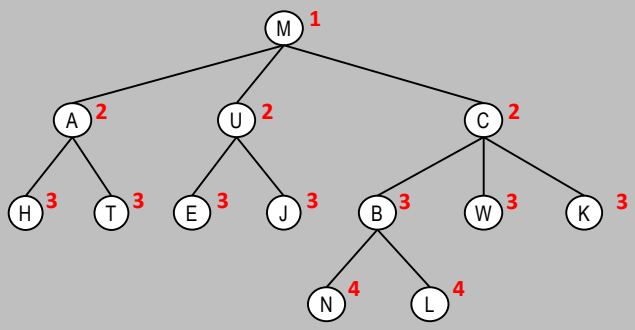
A profundidade de um nó é o número de ancestrais deste nó, excluindo o próprio nó.



14

Nível de um nó

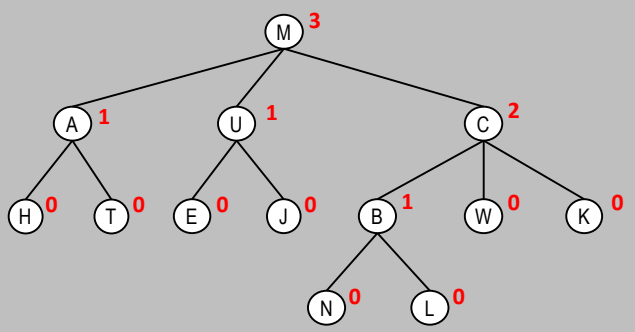
Em essência, é sinônimo de profundidade, porém alguns autores iniciam a contagem do nível em 1, pois deseja-se saber "quantos" níveis tem uma árvore.



15

Altura de um nó

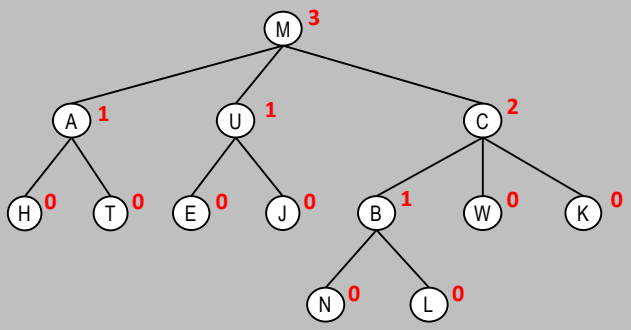
Se um nó é externo, então sua altura é 0.
Se um nó não é externo, então sua altura é um mais a altura máxima de seus filhos.



16

Altura de uma árvore não vazia

É a altura da raiz da árvore.




```
function height(tree)
    return heightRec(tree.root)


function heightRec(node)
    if external(node)
        return 0
    h = 0;
    for child in node.children
        h = max(h, heightRec(child))
    return 1 + h;
```

Altura desta árvore = 3

17

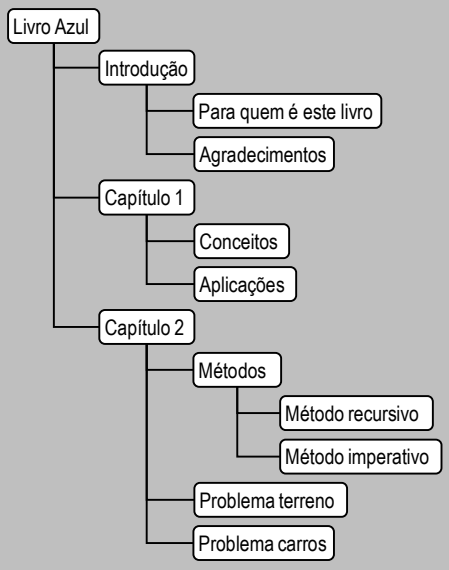
Árvores genéricas

18

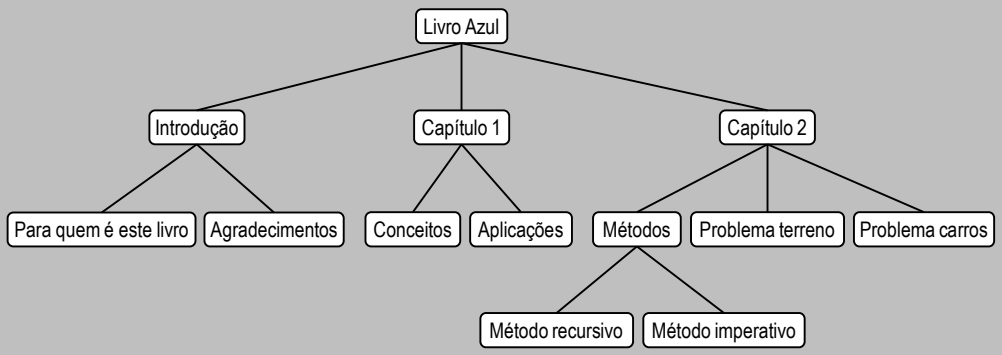
Árvores genéricas

São árvores de propósito geral para armazenar objetos de forma hierárquica.

Não possuem qualquer restrição específica de armazenamento dos objetos (ordenação, número de filhos, etc.).



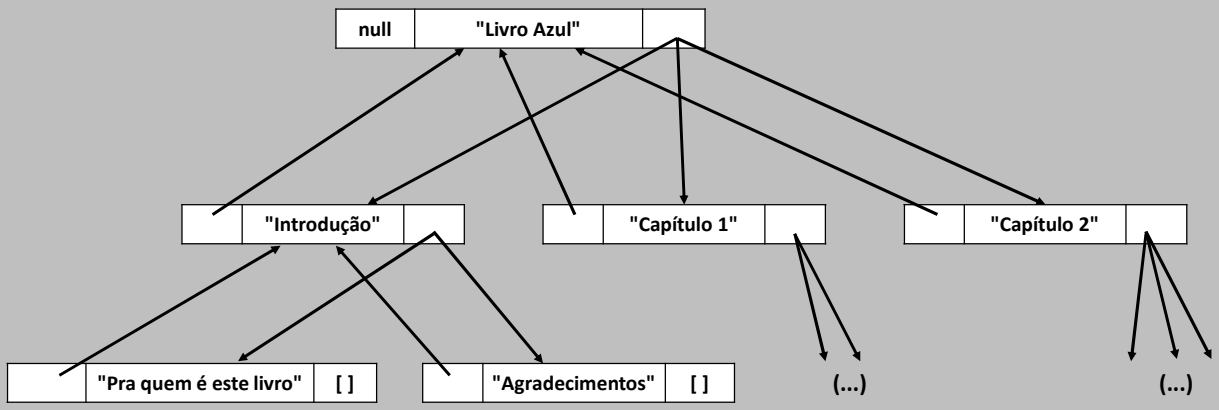
19



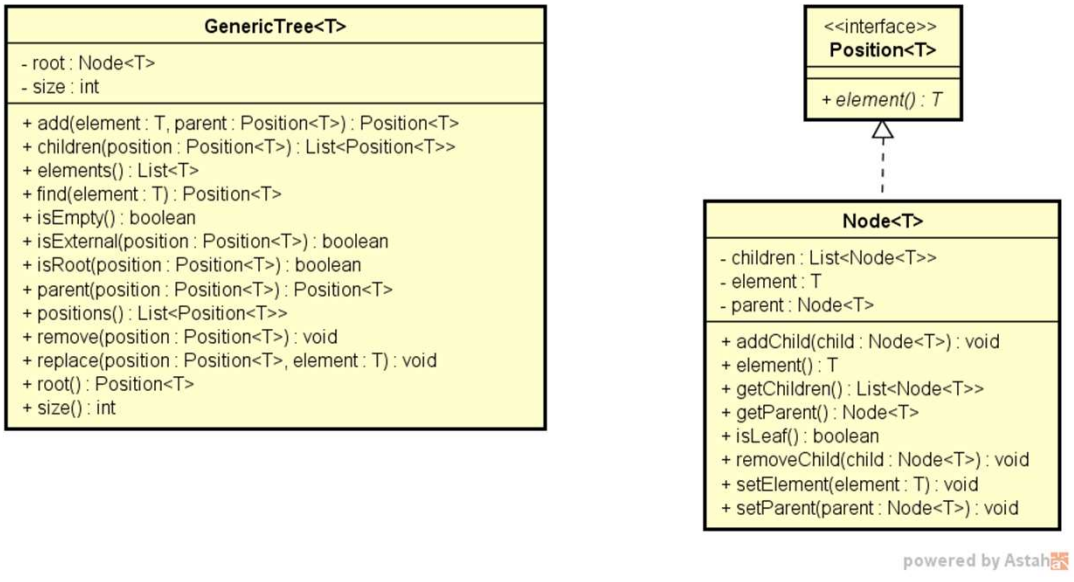
20

Projeto da árvore genérica

Cada nó terá: (1) o objeto armazenado, (2) a referência para o pai, e (3) a referência para os filhos.

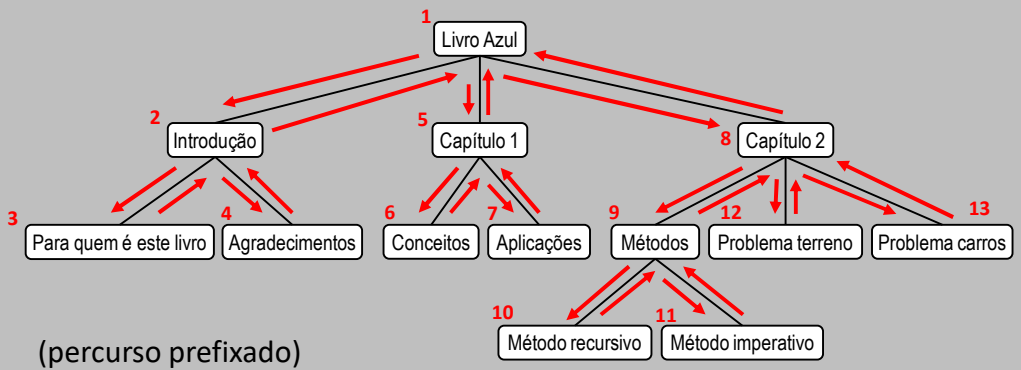


Projeto da árvore genérica



Percurso em profundidade
DFS - Depth-First Search

O DFS explora tão profundamente quanto possível ao longo de cada ramo antes de retroceder.



(percurso prefixado)

Algoritmo DFS (prefixado)

```
preOrder(tree)
   preOrderRecursive(tree.root)

preOrderRecursive(node)
   callAction(node)
   for (child in node.children)
      preOrderRecursive(child)
```

Percursos em árvores

- Percurso prefixado (pre order)
- Percurso pós-fixado (post order)
- Percurso interfixo (in order)
  (somente árvores binárias)

Algoritmo DFS (pós-fixado)

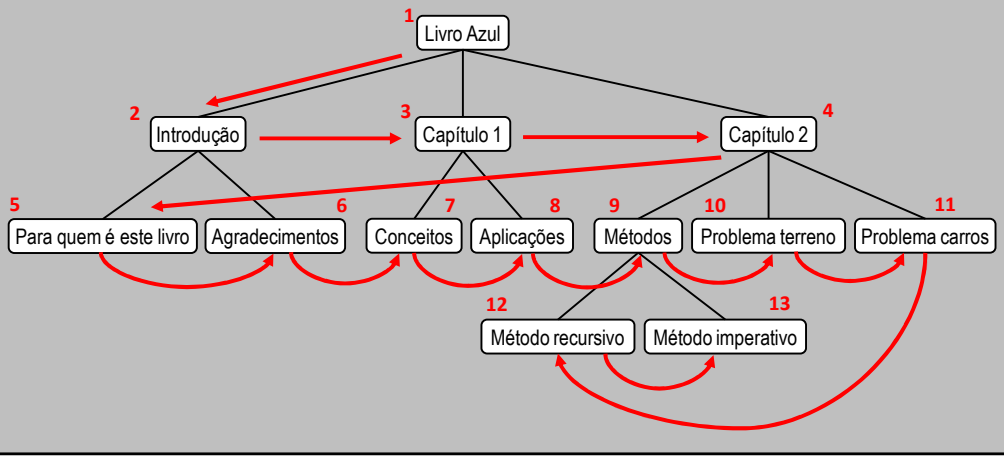
```
postOrder(tree)
   postOrderRecursive(tree.root)

postOrderRecursive(node)
   for (child in node.children)
      postOrderRecursive(child)
   callAction(node)
```

Algoritmo DFS (interfixo)

```
inOrder(tree)
   inOrderRecursive(tree.root)

inOrderRecursive(node)
   inOrderRecursive(node.left)
   callAction(node)
   inOrderRecursive(node.right)
```


Percurso por nível ou largura
BFS - Breadth-First Search

O BFS visita todos os nós em um nível antes de mover para o nível seguinte.



Algoritmo BFS

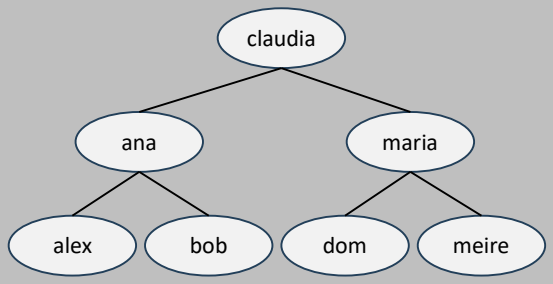
```
bfs(tree)
   if (tree.isEmpty())
      return

   queue = new Queue()
   queue.add(tree.root)
   while (!queue.isEmpty())
      node = queue.remove()
      callAction(node)
      queue.add(node.children)
```


Árvore binária de pesquisa - BST

Árvore binária de pesquisa - BST

Árvore binária de pesquisa (BST - binary search tree) é uma árvore binária onde:

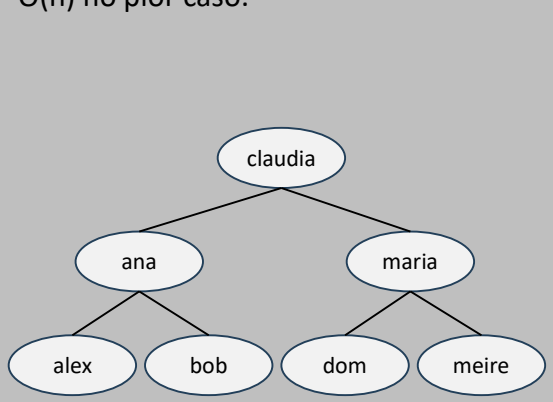
- Cada nó armazena um elemento identificado por uma chave.
- Cada chave é comparável, ou seja, pode ser igual, menor ou maior que outra chave.
- Para cada nó que possui uma chave K:
  - As chaves armazenadas nos nós da subárvore à esquerda são menores ou iguais a K.
  - As chaves armazenadas nos nós da subárvore à direita são maiores ou iguais a K.

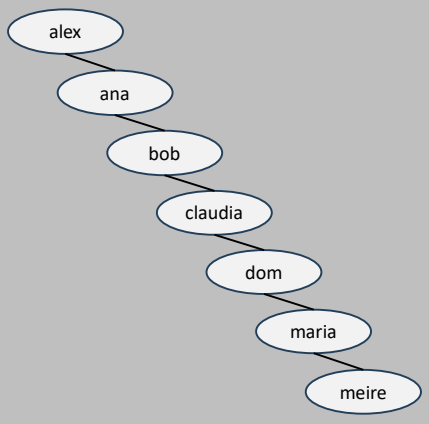


31

Complexidade da BST

As implementações simples da BST possuem complexidade de tempo para busca, inserção e remoção, de O(log n) no melhor caso, e O(n) no pior caso.





32

Aplicações de BST

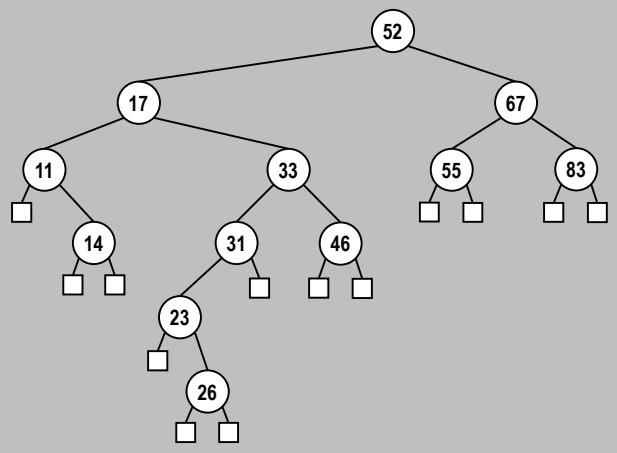
BSTs são indicadas para aplicações que fazem muitas consultas, inserções e remoções de objetos.

Particularmente, são indicadas para implementar **conjuntos** e **dicionários (mapas)**. Nestes casos, não pode haver repetição de chaves.

BinarySearchTreeSet<K>

BinarySearchTreeMap<K, V>

Implementação

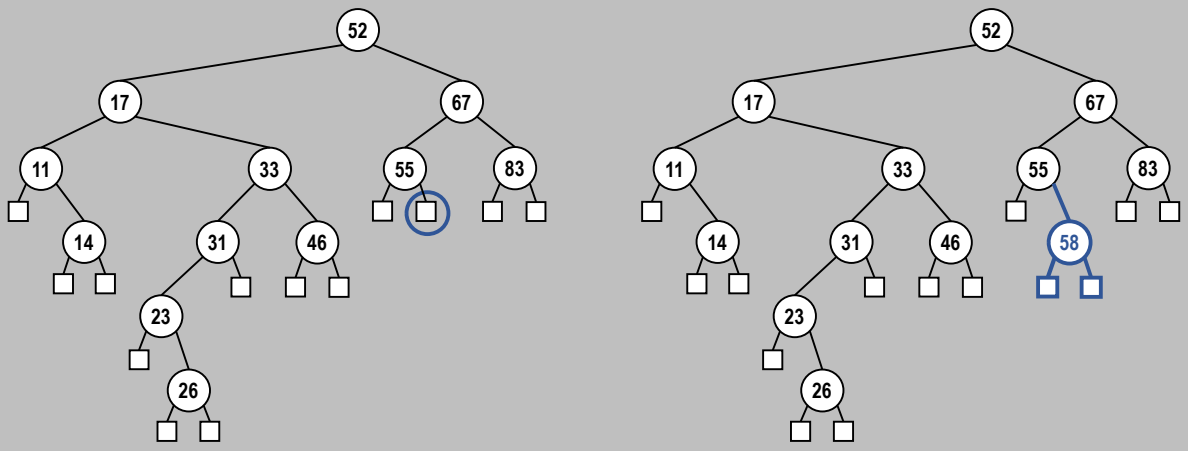


- Cada nó aponta para seu pai, esquerda e direita.
- As folhas serão nós adicionais chamados **sentinelas**, cuja chave, esquerda e direita, serão nulos.
- O uso de sentinelas vai ajudar muito nos algoritmos.

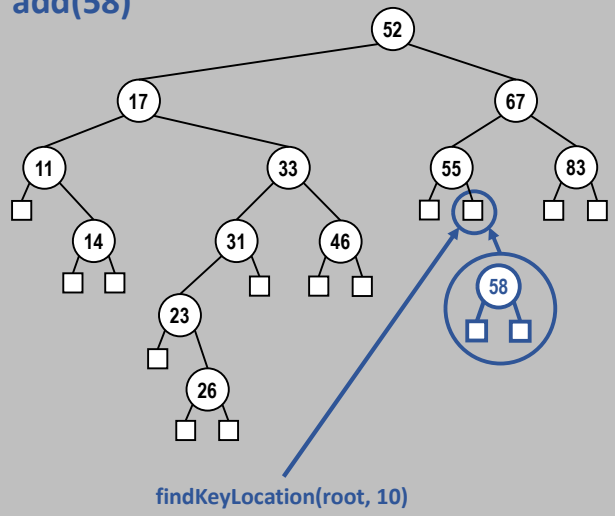
Inserção

Ideia geral: localiza a posição de inserção (sentinela) e troca esta sentinela pelo novo nó

Exemplo: add(58)



add(58)



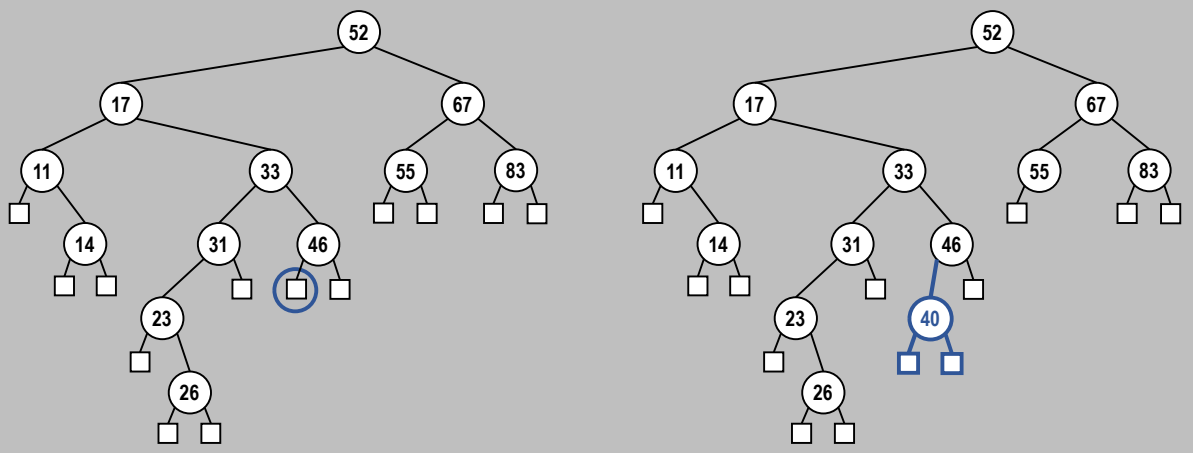
Algoritmo **add(key)**:

- se **key** é nula: lançar erro (**fim**)
- se a árvore é vazia: a raiz será um novo nó contendo a chave **key** (**fim**)
- **node** ← **findKeyLocation**(**root**, **key**)
- **se node** é sentinela (chave não encontrada):
  - Troca **node** por um novo nó contendo a chave.

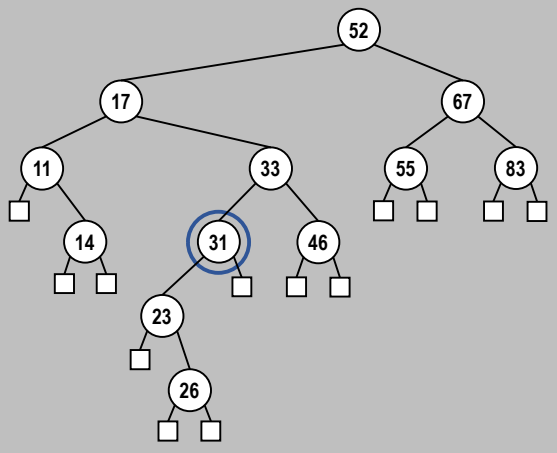
Algoritmo **findKeyLocation(node, key)**:

- **enquanto node** não é sentinela:
  - Se **key** = **node.key**: **retorne node**
  - Se **key** < **node.key**: **node** ← **node.left**
  - Se **key** > **node.key**: **node** ← **node.right**
- **retorne node**

add(40)

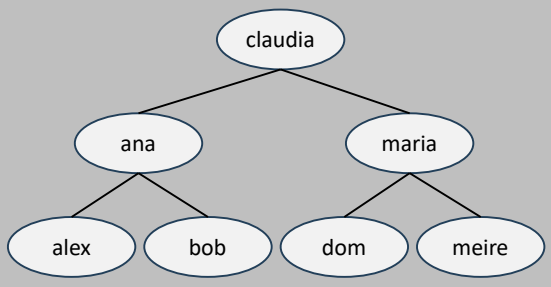


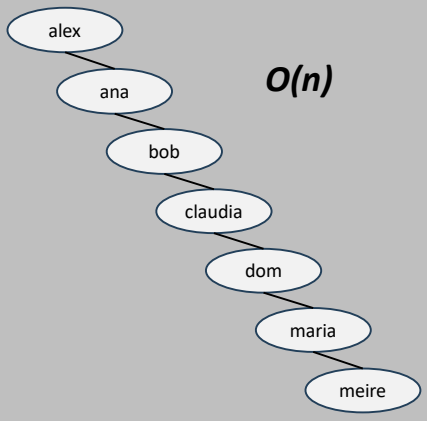
add(31)



** Como a chave já existia, nada acontece.*

O balanceamento da BST depende da ordem em que os elementos são inseridos





O(n)

O(log n)

$log_2 8 = 3$

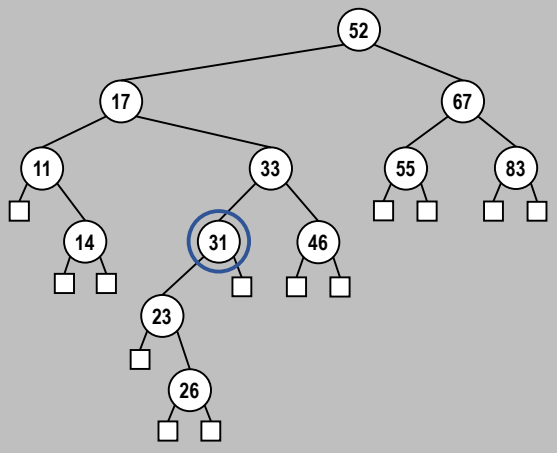
$log_2 1024 = 10$

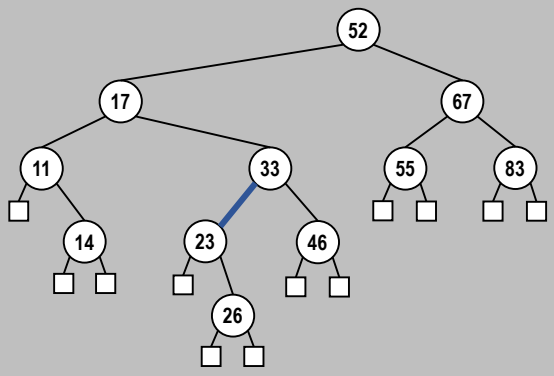
39

Remoção (nó com no máximo 1 filho)

Ideia geral válida para nó que possui **no máximo 1 filho**: trocar o nó pela sua subárvore

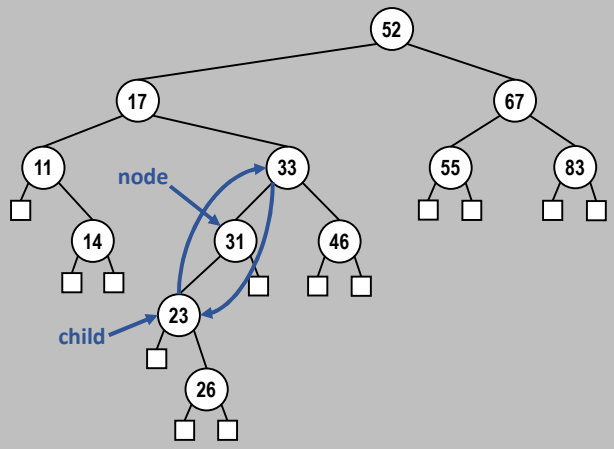
Exemplo: remove(31)





40

remove(31)



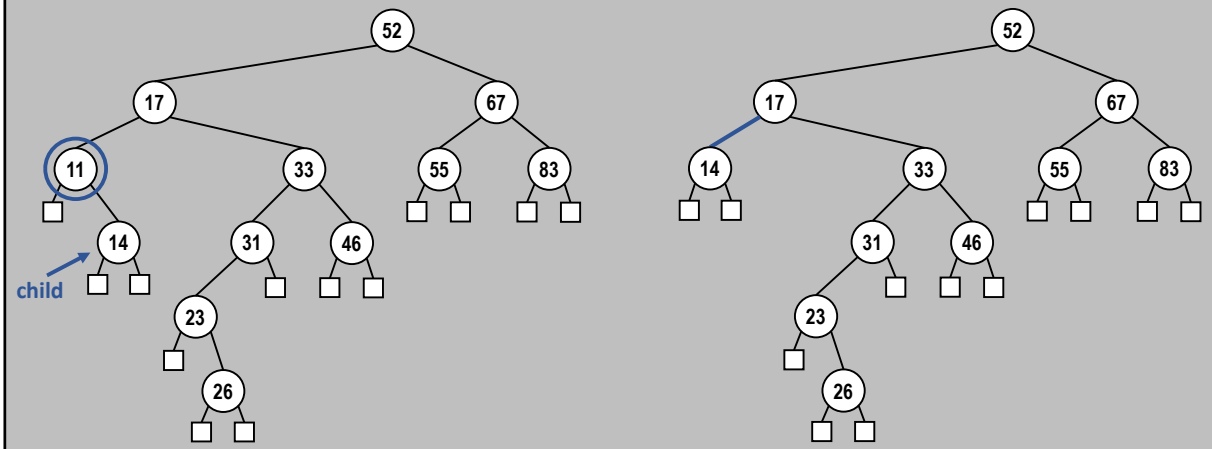
Algoritmo (remover nó que possui no máximo 1 filho):

remove(key):

- se **key** é nula: lançar erro (**fim**)
- **node** ← **findKeyLocation**(**root**, **key**)
- se **node** for sentinela (chave não encontrada): (**fim**)
- **child** ← (filho não sentinela de **node**)
- **child.parent** ← **node.parent**
- se **node.parent** é nulo (estamos removendo a raiz):
  - então: **root** ← **child**
  - senão: **node.parent.left** ou **node.parent.right** ← **child**

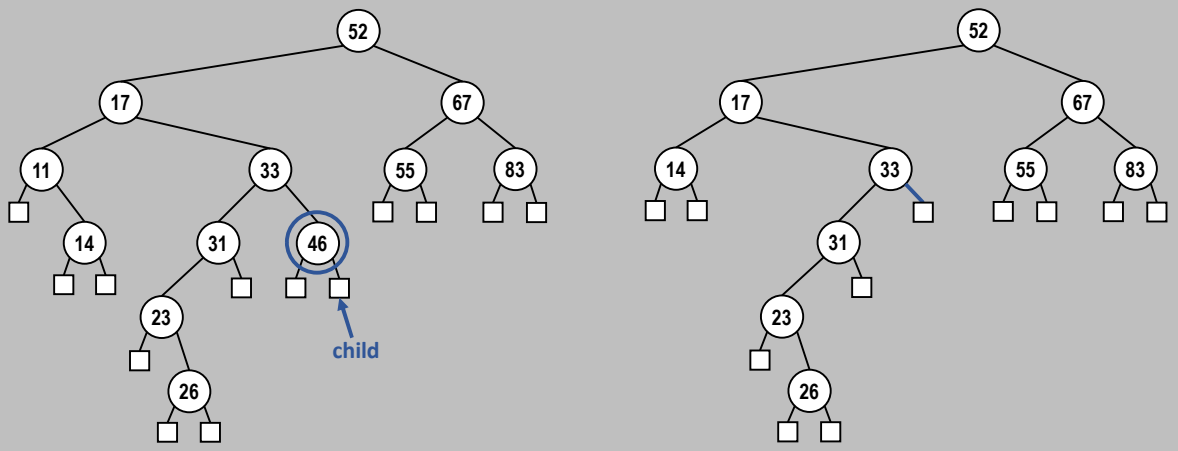
41

remove(11)



42

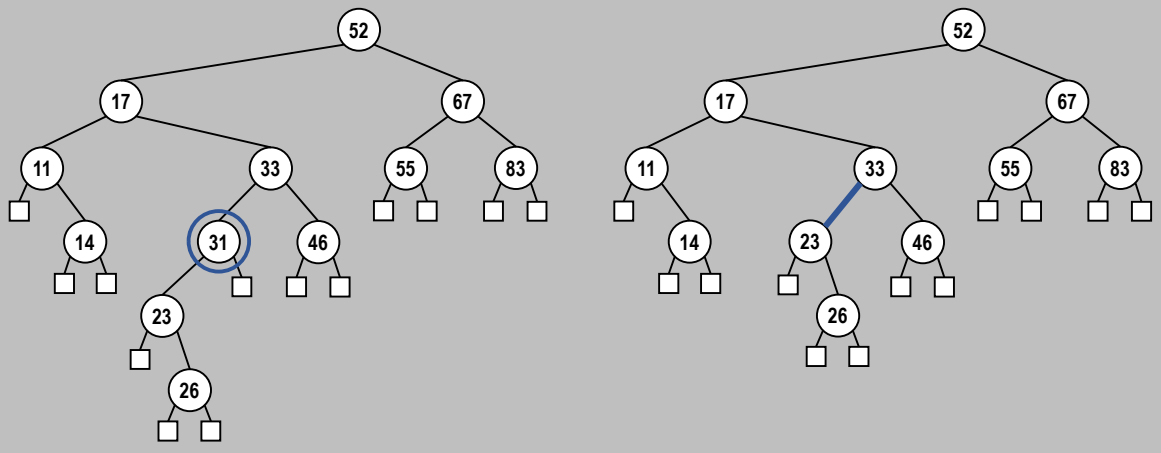
remove(46)



Remoção

Ideia geral válida para nó que possui **no máximo 1 filho**: trocar o nó pela sua subárvore

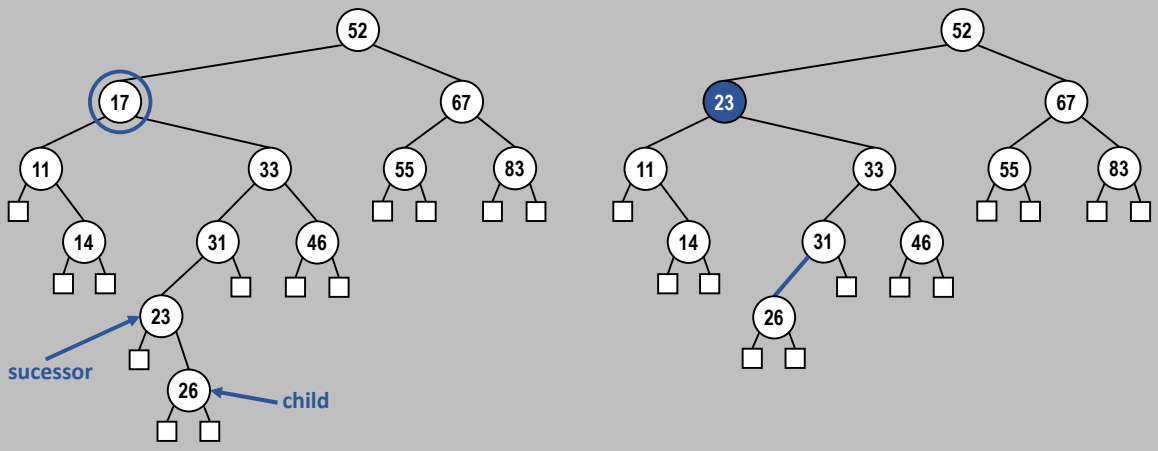
Exemplo: remove(31)



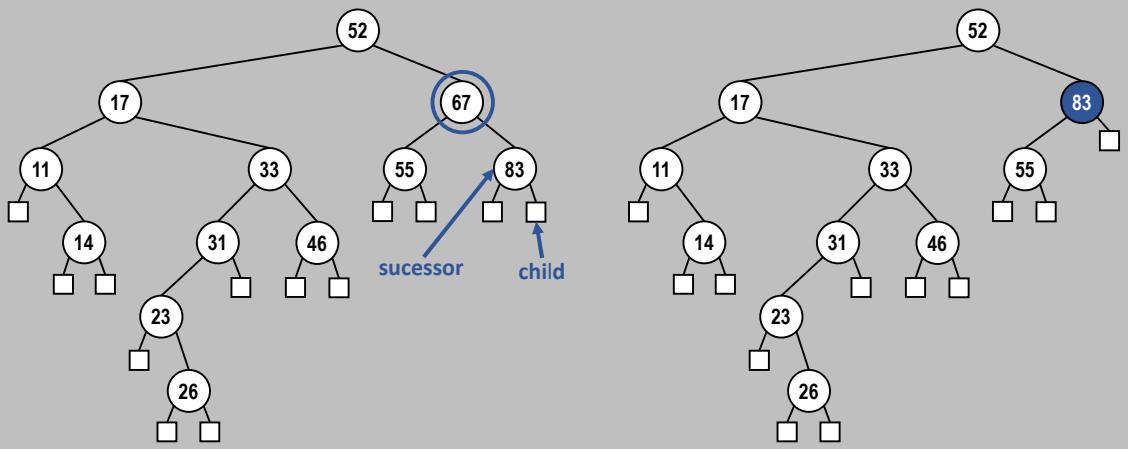
Remoção (nó com dois filhos)

Ideia geral: (1) encontra o sucessor (menor elemento da subárvore à direita), (2) atribui a chave do sucessor ao nó que contém a chave a ser removida, (3) remove o sucessor.

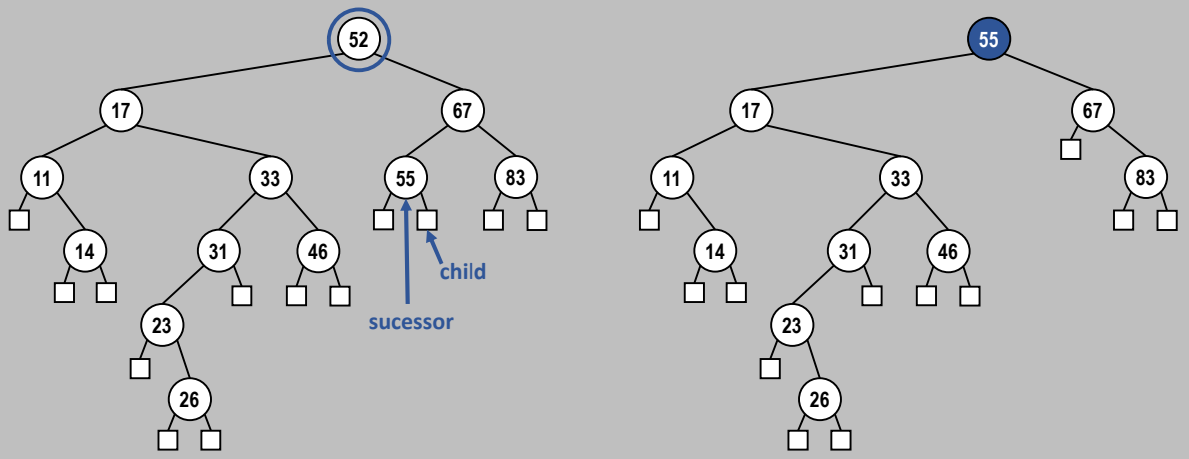
Exemplo: remove(17)



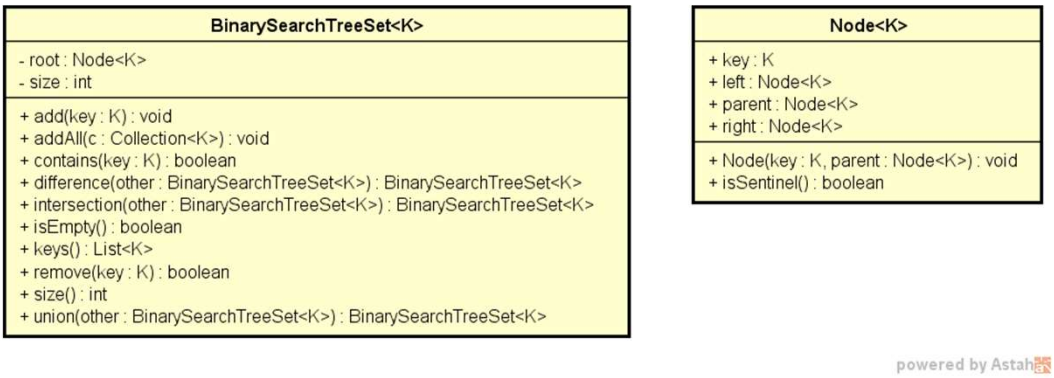
remove(67)



remove(52)



Projeto BST para conjunto



Projeto BST para dicionário