Estrutura de Dados II
Ciência da Computação

Prof. André Kishimoto
2024

1

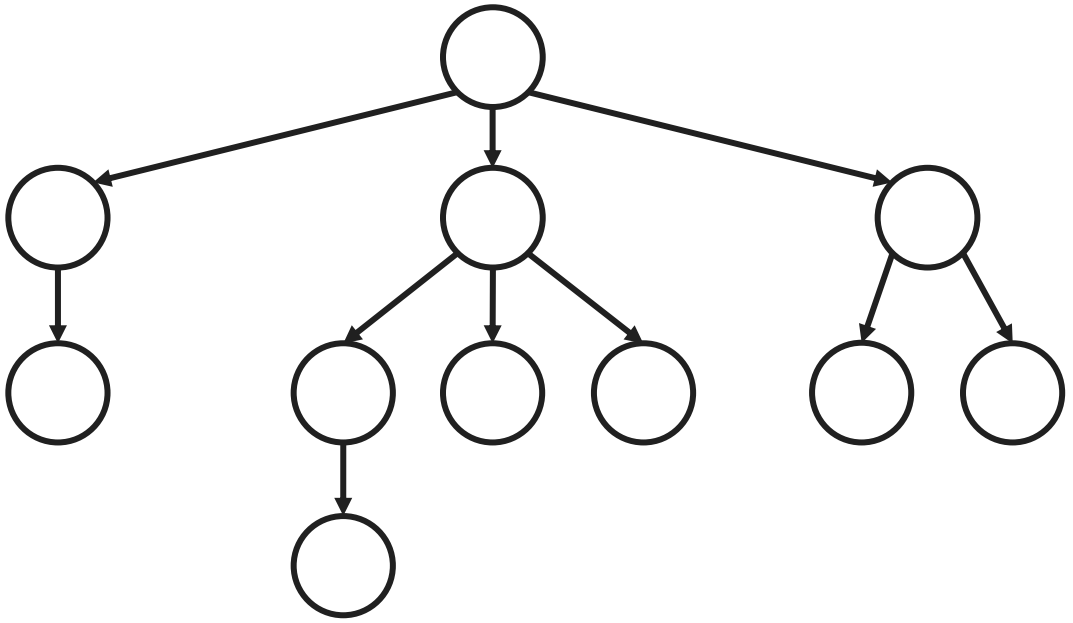
Árvores

Árvores

- Estrutura hierárquica.
  - Estrutura de dados não linear importante!
  - Eficiente para pesquisas.
- Adequada para representação de hierarquias.
  - Melhorar a organização de dados.
  - Facilitar o acesso aos dados.
  - Exemplo: *Como você armazena os trabalhos da faculdade ou as fotos que você tira com o celular na memória (disco) do computador? Existe alguma estrutura ou padrão?*

Árvores

ex.: Brasil



Árvores

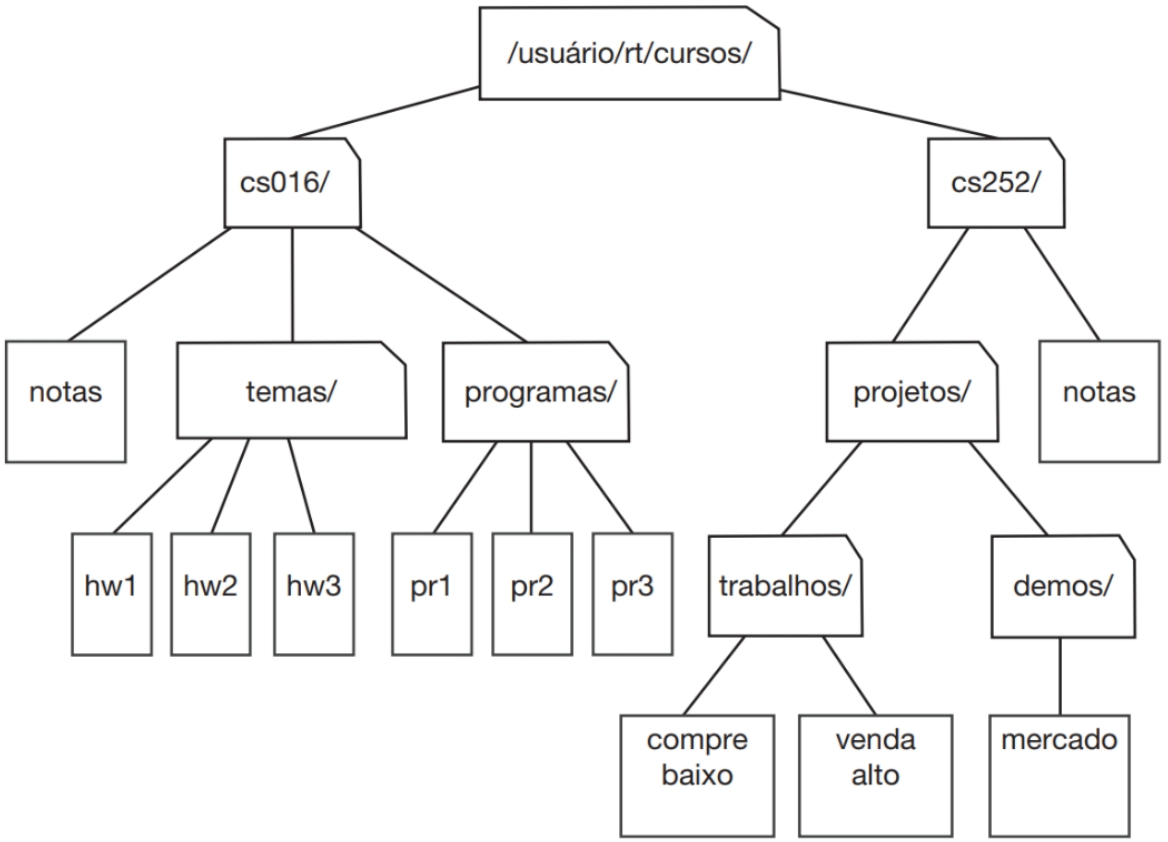


Figura 7.3 Árvore representando parte de um sistema de arquivos.

GOODRICH, M. T.; TAMASSIA, R. Estrutura de Dados e Algoritmos em Java, 5ª ed. Porto Alegre: Bookman, 2013.

Árvores

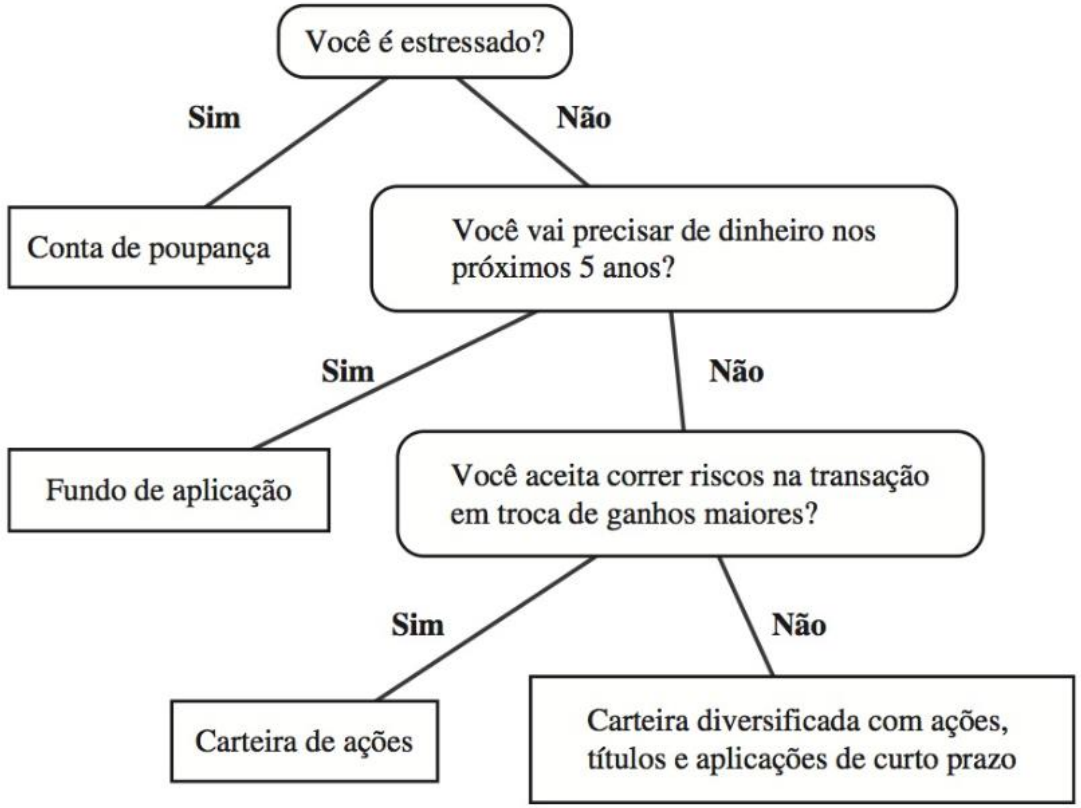


Figura 7.10 Árvore de decisão que fornece dicas de investimento.

GOODRICH, M. T.; TAMASSIA, R. Estrutura de Dados e Algoritmos em Java, 5ª ed. Porto Alegre: Bookman, 2013.

Árvores

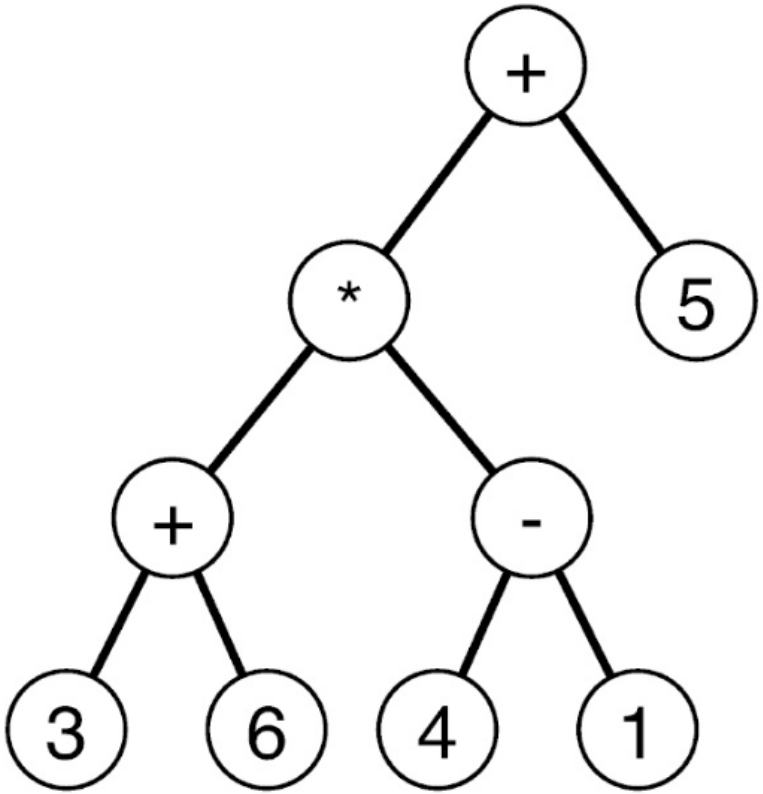


Figura 16.2: Árvore da expressão: $(3 + 6) * (4 - 1) + 5$.

CELES, W.; CERQUEIRA, R.; RANGEL, J. L. Introdução a estrutura de dados: com técnicas de programação em C, 2ª ed. Rio de Janeiro: Elsevier, 2016.

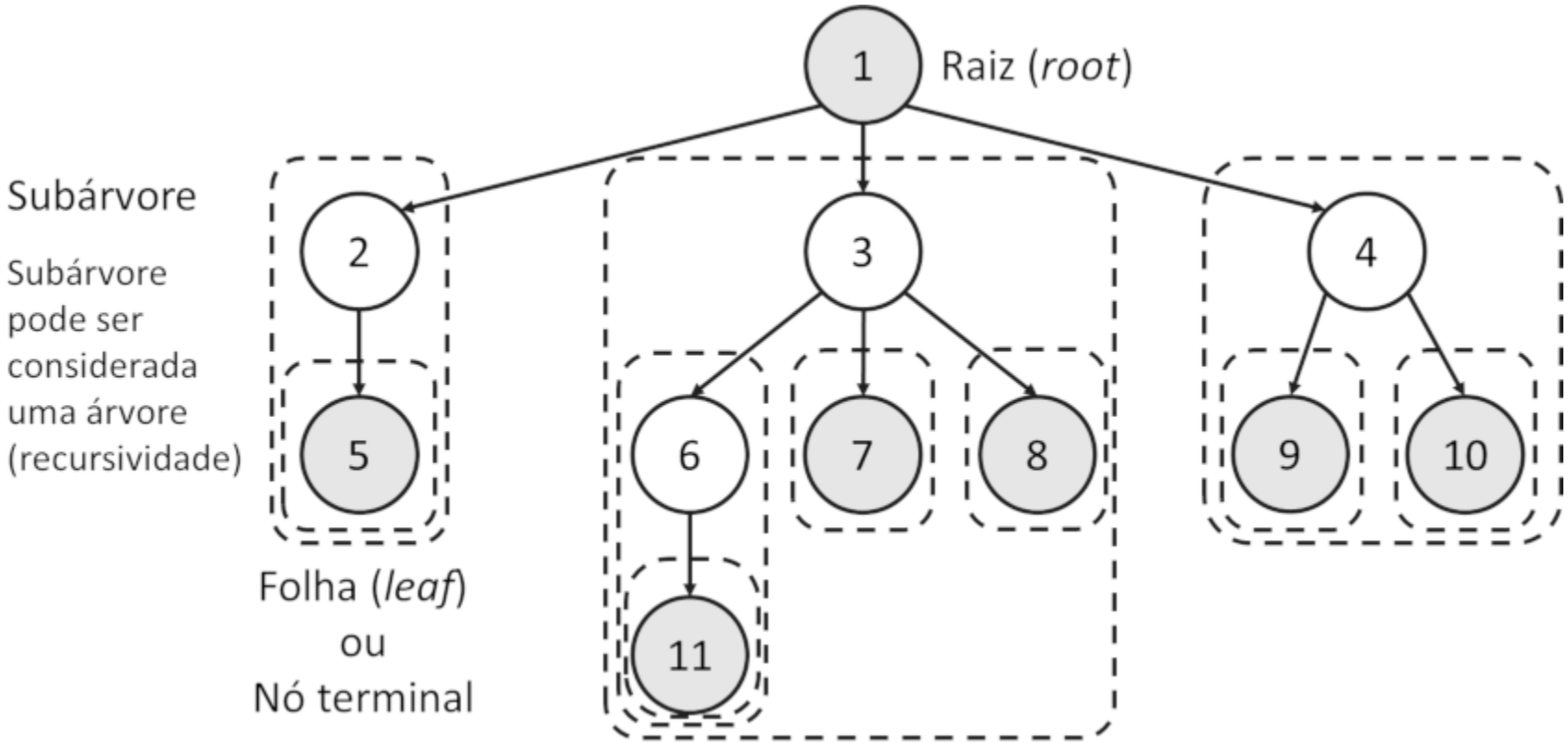
Árvores – conceitos

- Assim como listas, árvores são compostas por um conjunto de **nós** (os dados são armazenados em nós).
- Toda árvore (não-vazia) possui um nó denominado **raiz** (***root***).
- Todo nó de uma árvore é uma **subárvore** (***subtree***).
  - Recursividade!
- Relação de parentesco (hierarquia) entre os nós:
  - Nós raízes das subárvores são **filhos** (***children***, ***child***) do nó **pai** (***parent***).
  - Nós só podem ter um único nó pai, mas podem ter 0+ filhos.
  - O único nó que não tem pai é o nó raiz.

Árvores – conceitos

- Nós que possuem filhos são chamados de **nós internos** (***internal nodes***).
- Nós que não possuem filhos são chamados de **folhas** (***leaves***, ***leaf***, ***leaf node***), **nós externos** (***external nodes***) ou **nós terminais** (***terminal nodes***).
- Nós com mesmo pai são **irmãos** (***siblings***).

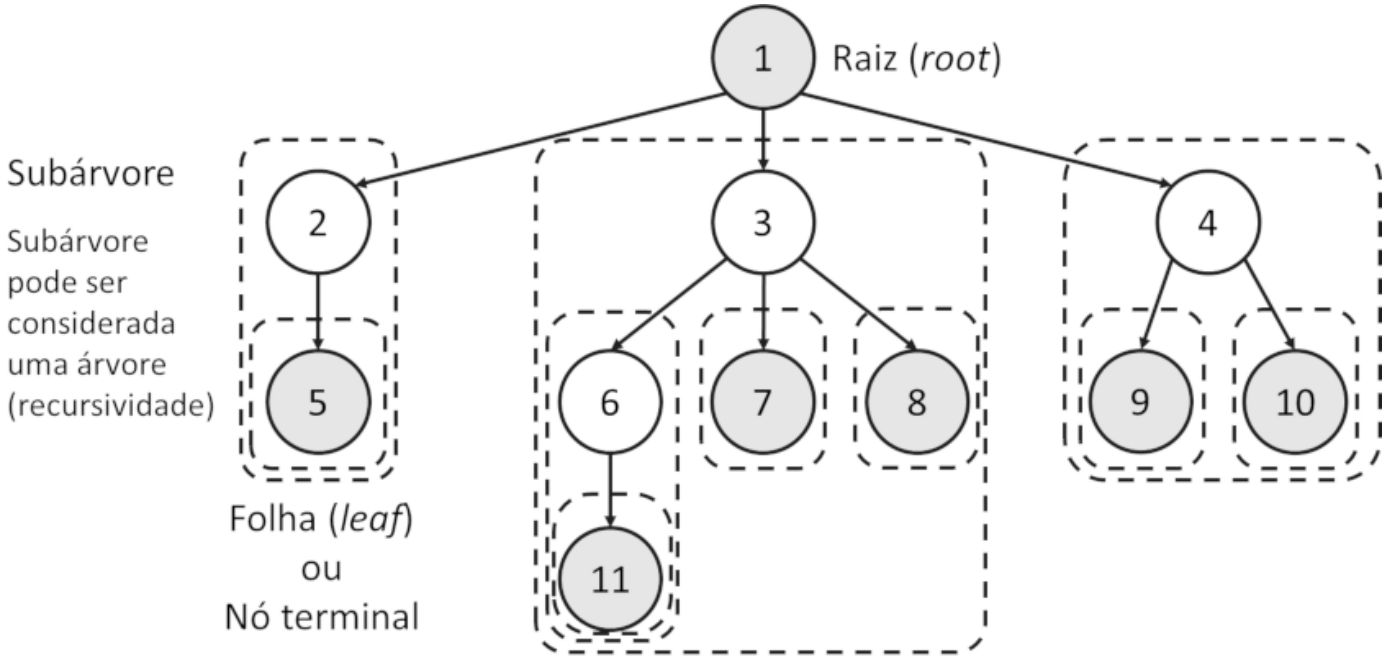
Árvores – conceitos



Árvores – grau

- Todo **nó** possui um **grau** (***degree***), definido pelo número total de filhos do nó.
- Toda **árvore** também possui um **grau**, definido pelo maior grau de um nó pertencente à árvore.

Árvores – grau



Nó	Grau do nó (quantidade de filhos)
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	

E o grau da árvore?

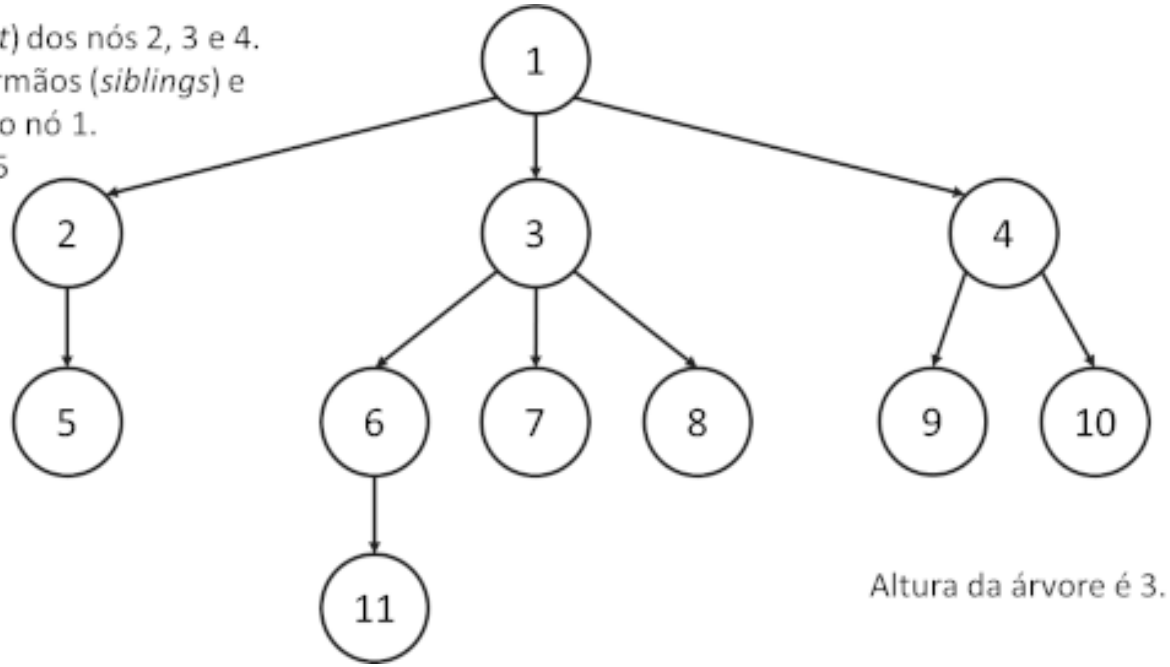
Árvores – altura

- Toda árvore possui uma **altura** (***tree height***), definida pelo caminho mais longo do nó raiz até uma das folhas.
  - Podemos contar as **arestas** (***edges***) que existem no caminho mais longo da raiz até uma das folhas ou
  - Usar o maior **nível** da árvore (nível mais profundo da árvore) – ver adiante.
- Nós também possuem altura (um nó é uma subárvore).
  - **Folha** tem **altura zero** (há zero arestas abaixo desse nó).
  - Se não for folha, calculamos a altura de um nó com:

  h(N) = altura do nó N = 1 + maior altura dos nós filhos de N

Árvores – altura

Nó 1 é pai (*parent*) dos nós 2, 3 e 4.
Nós 2, 3 e 4 são irmãos (*siblings*) e filhos (*children*) do nó 1.
Nó 2 é pai do nó 5 (que é filho do nó 2).
Etc...



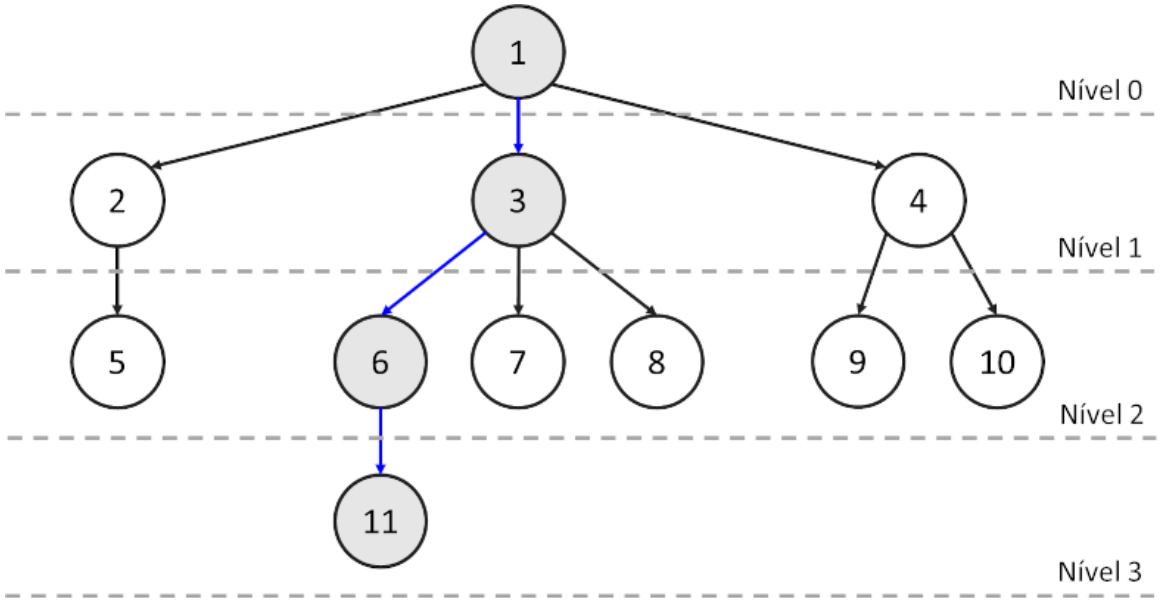
Altura da árvore é 3.

Nó	Altura do nó – h(n)
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	

Árvores – nível dos nós

- Os nós de uma árvore possuem um **nível** (***level***, ***depth***), que pode ser obtido com a equação:

d(N) = nível do nó N = nível do pai de N + 1

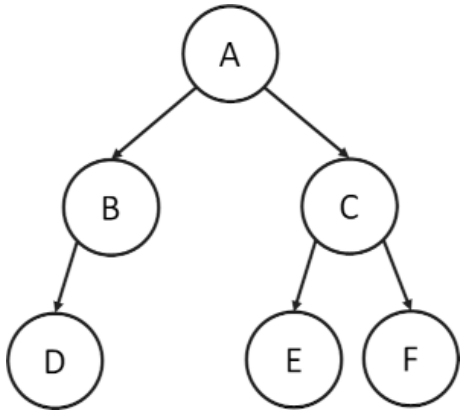


2

Árvore Binária

Árvore Binária

- *Binary tree* é uma especialização de árvore.
- Uma regra principal:
  - **Cada nó da árvore binária tem zero, um ou dois filhos (no máximo).**
  - Filho esquerdo (*left child*) e filho direito (*right child*).

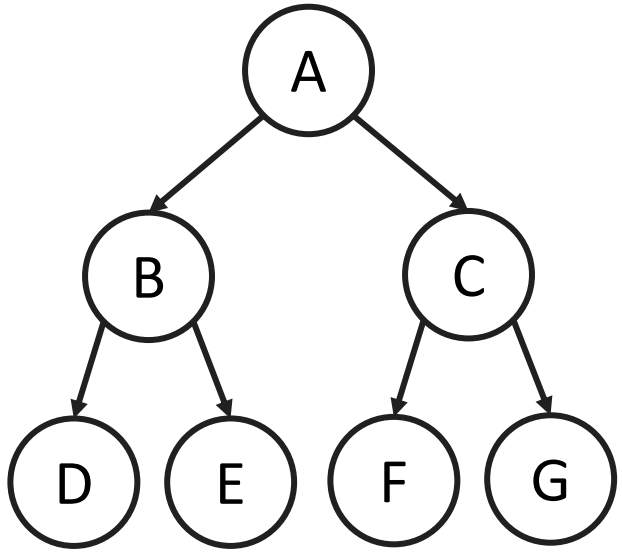


Árvore Binária

Propriedades:

- A **altura** de uma árvore com ***n*** elementos (n > 0) é no **mínimo** $\lfloor \log_2 n \rfloor$ e no **máximo *n-1***.
  - $\log_2 n = x$ (inverso é $2^x = n$)

- Uma árvore binária **não vazia** com altura ***h*** tem no **mínimo** ***h+1*** nós e no **máximo** $2^{h+1} - 1$ nós.



Árvore binária cheia

- Altura mínima e máxima?
- Quantidade mínima e máxima de nós?
- Exemplo de altura máxima?

Árvore Binária

- Podemos definir uma árvore binária de maneira recursiva.
- Uma árvore binária ou é vazia ou consiste em:
  - Um nó raiz da árvore T que armazena um elemento;
  - Uma árvore binária, chamada de subárvore da esquerda de T;
  - Uma árvore binária, chamada de subárvore da direita de T.

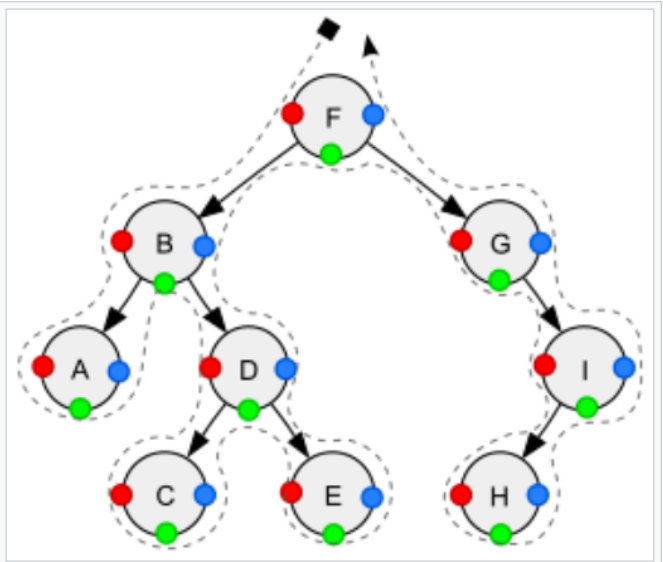
Percurso/travessia em árvore

- Árvores binárias são usadas principalmente para busca.
- É possível percorrer uma árvore:
  - **Pré-ordem:** Visitar a raiz, depois a subárvore esquerda e, por último, a subárvore direita.
  - **Em ordem:** Visitar a subárvore esquerda, depois a raiz e, por último, a subárvore direita.
  - **Pós-ordem:** Visitar a subárvore esquerda, depois a subárvore direita e, por último, a raiz.
  - **Por nível:** Visitar a raiz, depois os nós do próximo nível, da esquerda para a direita, depois os nós do próximo nível, da esquerda para a direita, até percorrer o último nível.

Percurso/travessia em árvore

- Árvores binárias são usadas principalmente para busca.
- É possível percorrer uma árvore:
  - **Pré-ordem:** pré-fixado, pre-order, **NLR** traversal.
  - **Em ordem:** infixado, in-order, **LNR** traversal.
  - **Pós-ordem:** pós-fixado, post-order, **LRN** traversal.
  - **Por nível:** level-order traversal.

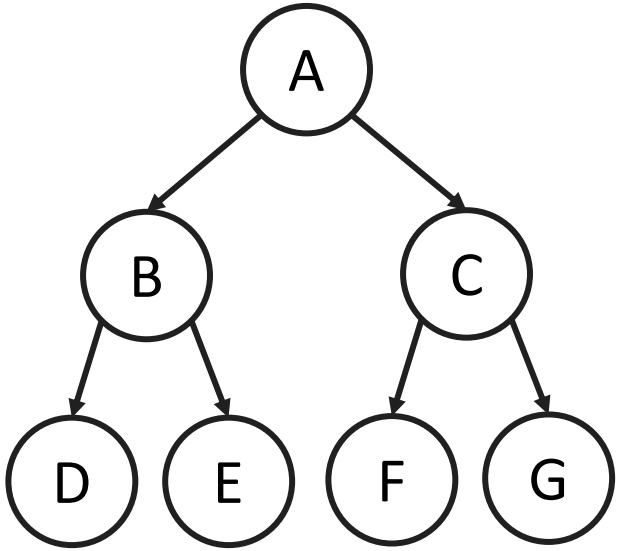
  - N: Node
  - L: Left
  - R: Right



Depth-first traversal (dotted path) of a binary tree:
Pre-order (node visited at position red ●):
F, B, A, D, C, E, G, I, H;
In-order (node visited at position green ●):
A, B, C, D, E, F, G, H, I;
Post-order (node visited at position blue ●):
A, C, E, D, B, H, I, G, F.

https://en.wikipedia.org/wiki/Tree_traversal

Percurso/travessia em árvore



- Percurso em pré-ordem?
- Percurso em ordem?
- Percurso em pós-ordem?
- Percurso por nível?

Percurso/travessia em árvore

- Percurso em pré-ordem?
- Percurso em ordem?
- Percurso em pós-ordem?
- Percurso por nível?

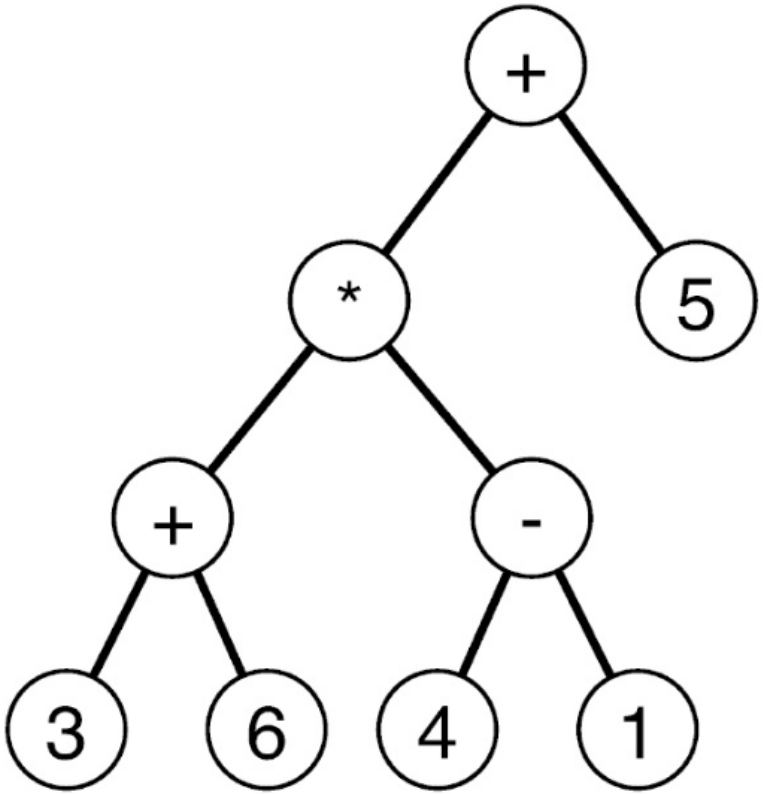


Figura 16.2: Árvore da expressão: $(3 + 6) * (4 - 1) + 5$.

CELES, W.; CERQUEIRA, R.; RANGEL, J. L. Introdução a estrutura de dados: com técnicas de programação em C, 2ª ed. Rio de Janeiro: Elsevier, 2016.

Percurso (...)

Avaliação de uma árvore de expressão

O caminhamento pós-fixado de uma árvore binária pode ser usado para resolver o problema de avaliação de expressões. Nesse problema, dada a árvore de uma expressão aritmética, ou seja, uma árvore binária na qual para cada nodo externo existe um valor associado e para cada nodo interno se associa um operador aritmético (ver Exemplo 7.9), deseja-se calcular o valor da expressão aritmética representada pela árvore.

O algoritmo evaluateExpression, indicado no Trecho de Código 7.24, avalia a expressão associada com a subárvore com raiz no nodo v de uma árvore T, que representa uma expressão aritmética, executando um caminhamento pós-fixado T que se inicia em v. Nesse caso, a ação "de visita" sobre cada nodo consiste na execução de uma operação aritmética simples. Observa-se que se explora o fato de que uma árvore de expressão aritmética é uma árvore binária própria.

```
Algoritmo evaluateExpression(T,v):
  se v é um nodo interno de T então
    seja ○ o operador armazenado em v
    x ← evaluateExpression(T,T.left(v))
    y ← evaluateExpression(T,T.right(v))
    retorna x ○ y
  senão
    retorna o valor armazenado em v
```

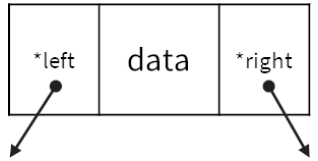
Trecho de Código 7.24 Algoritmo evaluateExpression para calcular a expressão representada pela subárvore de uma árvore T que representa uma expressão aritmética, enraizada no nodo v.

A aplicação de caminhamento pós-fixado na avaliação de expressões aritméticas resulta em um algoritmo que executa em tempo $O(n)$ para avaliar uma expressão aritmética representada por uma árvore binária de n nodos. Na verdade, da mesma forma que o caminhamento pós-fixado genérico, o caminhamento pós-fixado para árvores binárias pode ser aplicado para outros problemas "bottom-up" (como, por exemplo, o problema de cálculo do tamanho apresentado no Exemplo 7.7).

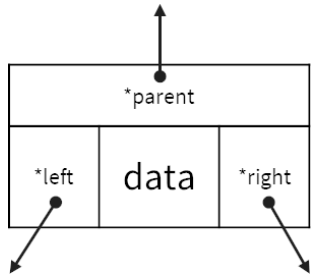
GOODRICH, M. T.; TAMASSIA, R. Estrutura de Dados e Algoritmos em Java, 5ª ed. Porto Alegre: Bookman, 2013.

Implementação

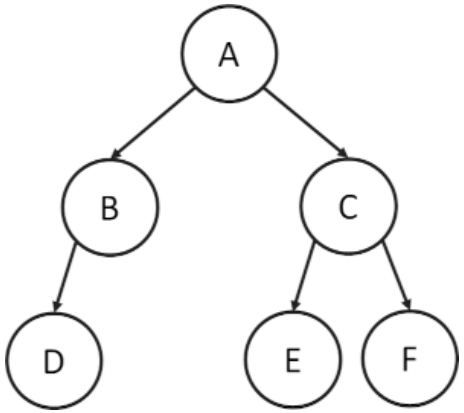
- Uma árvore é composta por um conjunto de nós.
- No caso de uma árvore binária, o mínimo que o nó precisa ter é:
  - A parte dos **dados**, o conteúdo que é armazenado por cada nó (*data*);
  - Um ponteiro para o **nó filho esquerdo** (**left*);
  - Um ponteiro para o **nó filho direito** (**right*).

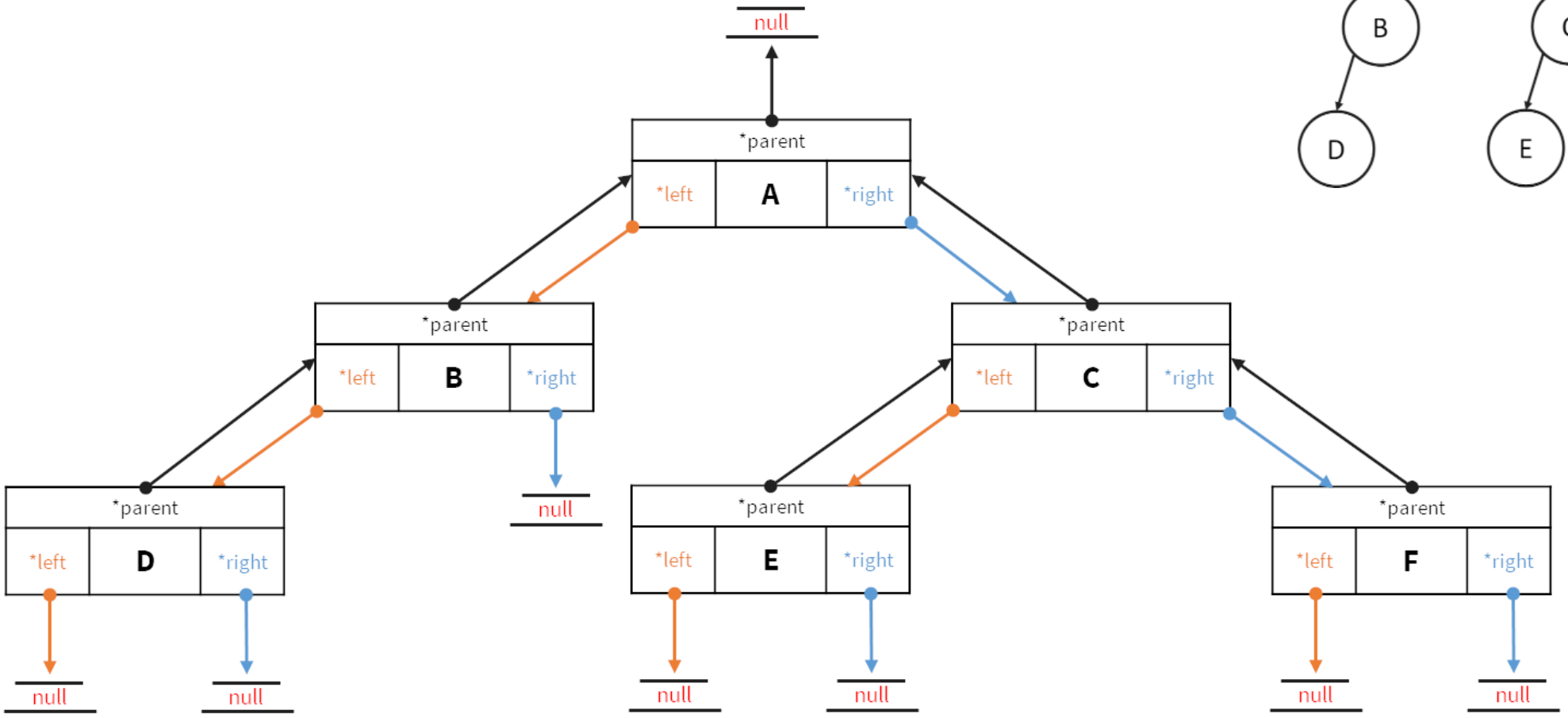


- Podemos incluir um ponteiro que aponta para o **nó pai** (***parent*).
  - Não é obrigatório, mas ter acesso ao nó pai pode facilitar algumas operações.



Implementação



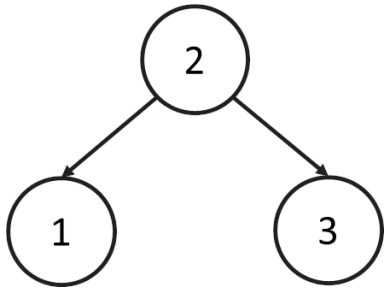


3

Árvore Binária de Busca (BST)

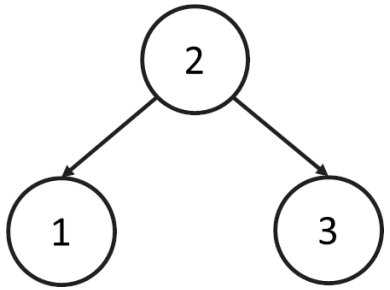
Árvore binária de busca [binária] (BST)

- BST: *Binary Search Tree*.
- Árvore binária com uma nova regra:
  - *Dado um nó T da árvore, todas as subárvores que se encontram à* ***esquerda de T possuem nós com valores menores que T*** *e todas as subárvores que se encontram à* ***direita de T possuem nós com valores maiores que T****.*



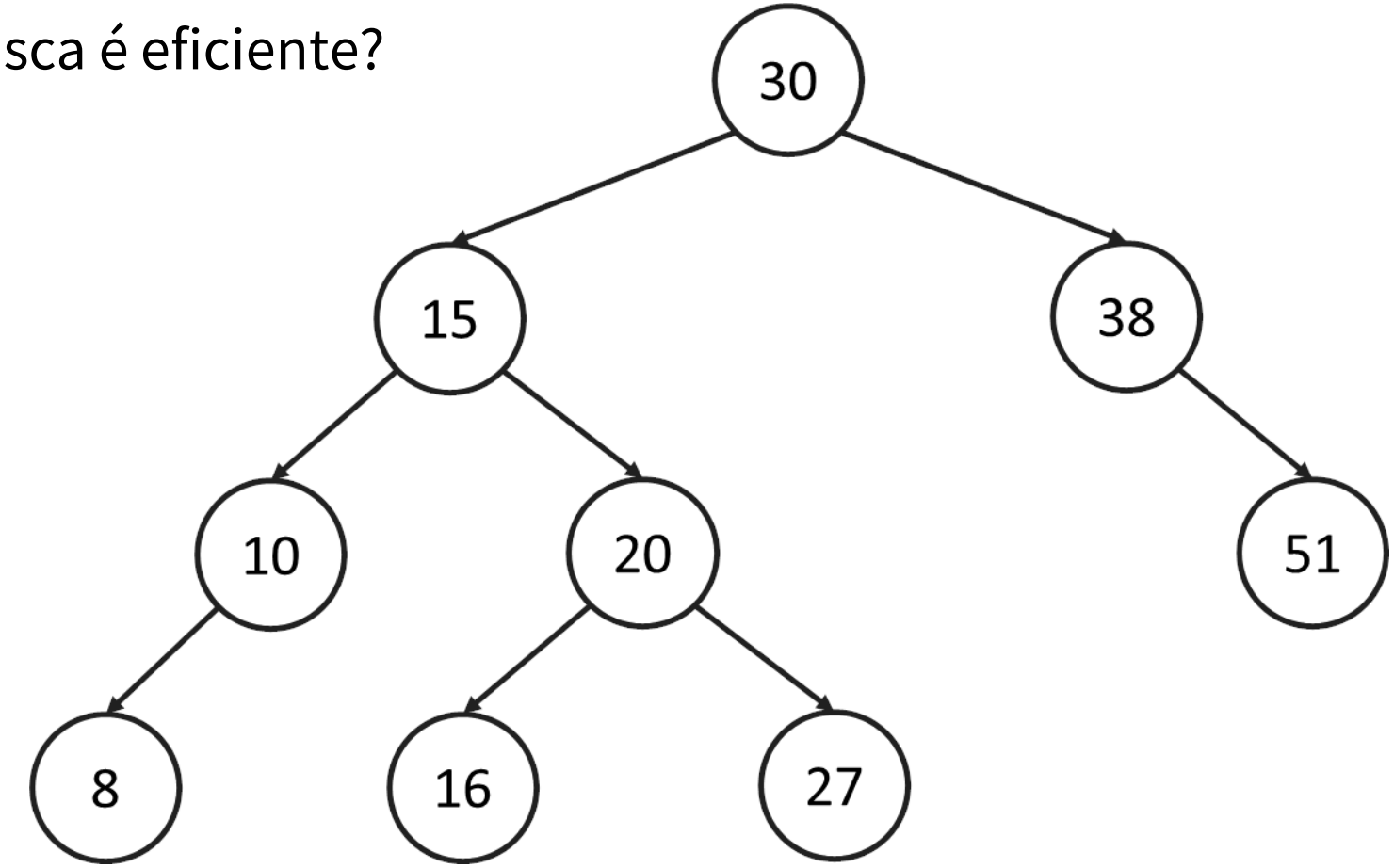
Árvore binária de busca (BST)

- O que define um nó ter um valor (chave) menor ou maior do que outro nó?
  - Depende do contexto, desde que seja possível comparar se um valor é menor, maior ou igual a outro.
  - Igualdade: geralmente, uma BST não tem nós com valores duplicados.
  - Exemplo mais simples: números inteiros e positivos.



Árvore binária de busca (BST)

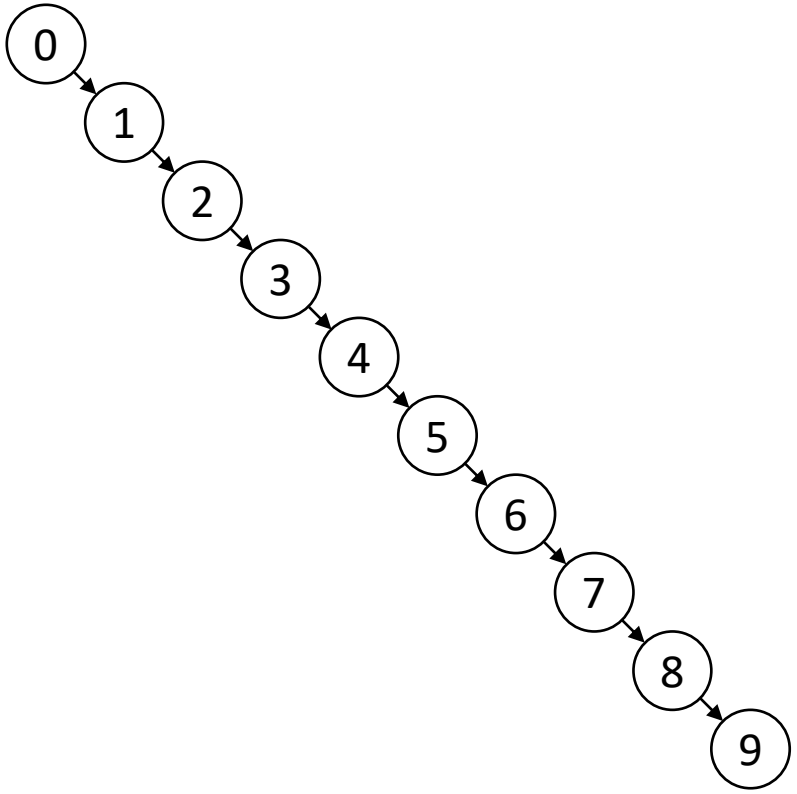
Por que a busca é eficiente?



Árvore binária de busca (BST)

- Vamos inserir os seguintes elementos em uma BST vazia, na ordem apresentada:
- 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
- Como fica a representação visual da árvore?

Árvore binária de busca (BST)



- A ordem da inserção dos elementos afeta a construção da BST...
- Nas próximas aulas, veremos uma outra árvore que resolve esse problema (de balanceamento de árvore).

BST – Busca

- BST é uma árvore binária para realizar buscas binárias.
  - Divisão e conquista.
  - Lembra da busca binária em um vetor de elementos ordenados?
  - A ideia de busca em uma BST é a mesma!
- Começamos pela raiz da árvore.
  - Se a chave da raiz é a que buscamos, fim.
  - Senão, comparamos a chave a ser buscada com a raiz.
  - É menor do que a raiz? Continua a busca pela subárvore esquerda.
  - É maior do que a raiz? Continua a busca pela subárvore direita.
  - Continua até encontrar o nó com a chave OU encontrar um nó nulo (chave não existe na árvore).

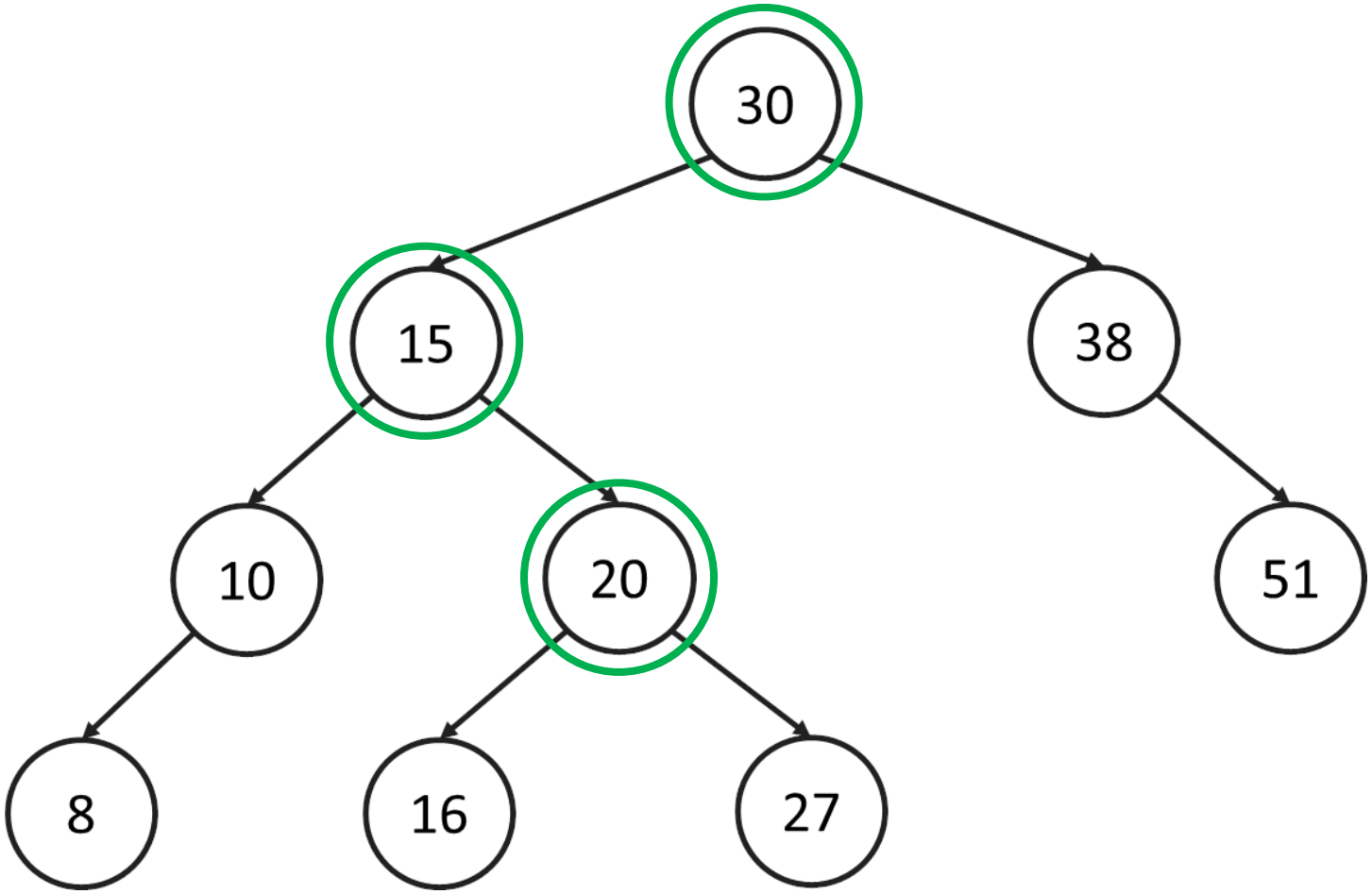
BST – Busca

Exemplo:
Buscar o elemento 20.

Quantas comparações?
3 comparações.

Complexidade:
- Pior caso, O(n)
- Média, O(log n)

n: quantidade de elementos



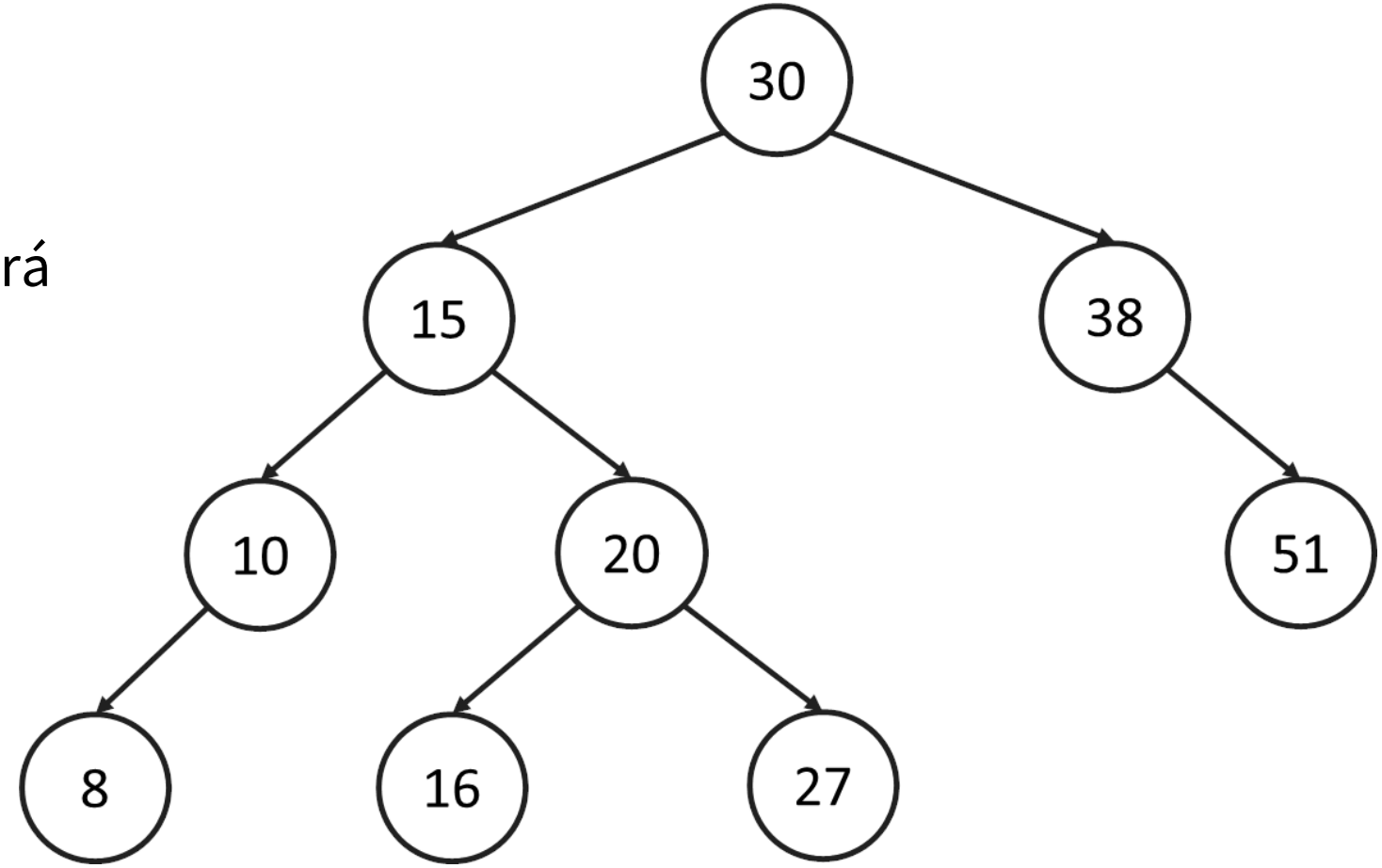
BST – Inserção

- Primeira etapa é similar à busca: procuramos uma posição na árvore para inserir o novo nó, seguindo a regra da BST.
  - Chave menor que um nó T deve ser inserido à esquerda.
  - Chave maior que um nó T deve ser inserido à direita.
- Percorrer a árvore e encontrar um nó nulo:
  - Para a operação de busca: busca concluída sem encontrar a chave especificada.
  - Para a operação inserção: encontramos a posição para a nova chave.

BST – Inserção

Exemplo:
Inserir o elemento 35.

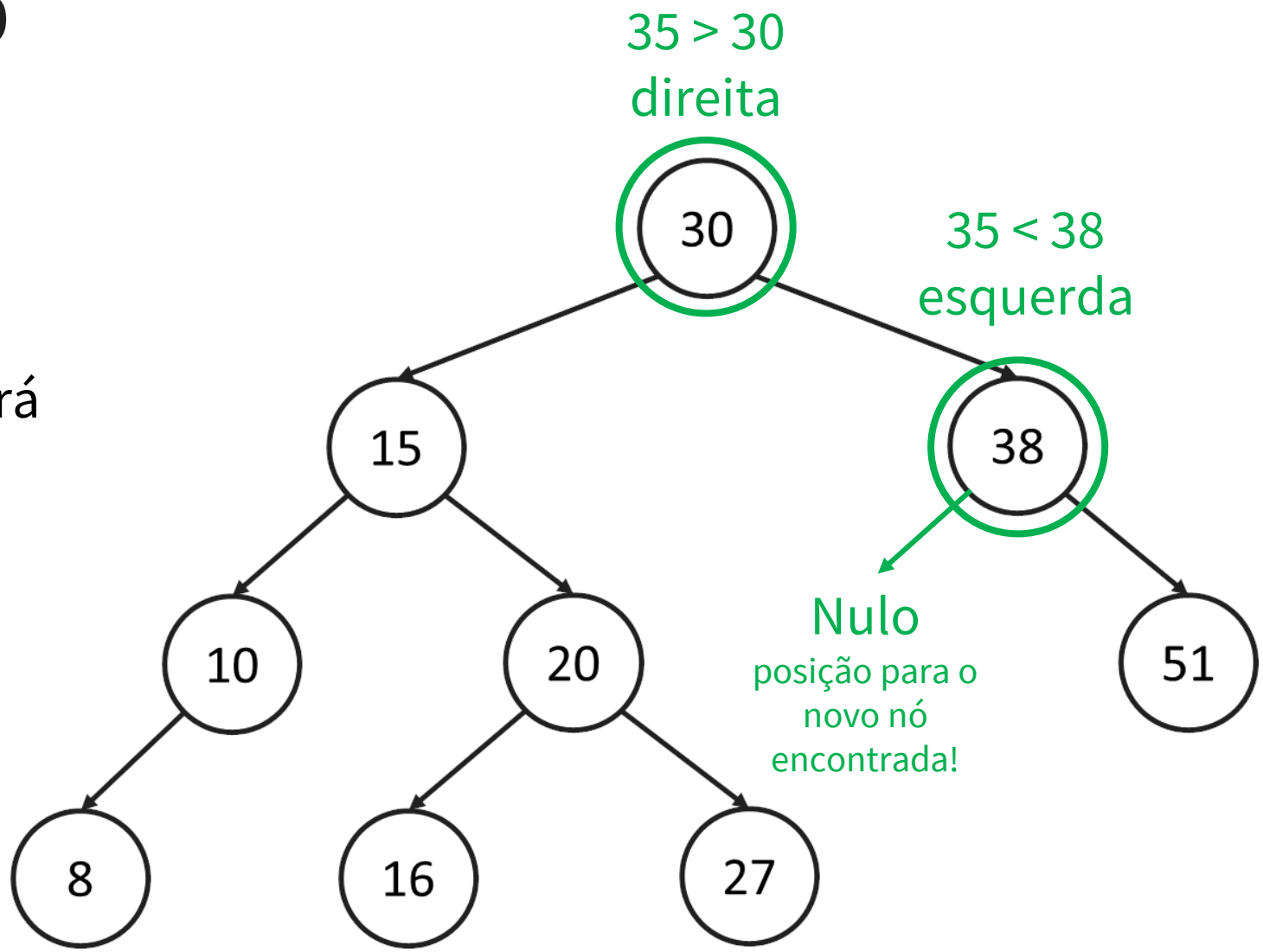
Onde o nó com a chave 35 será inserido na árvore ao lado?



BST – Inserção

Exemplo:
Inserir o elemento 35.

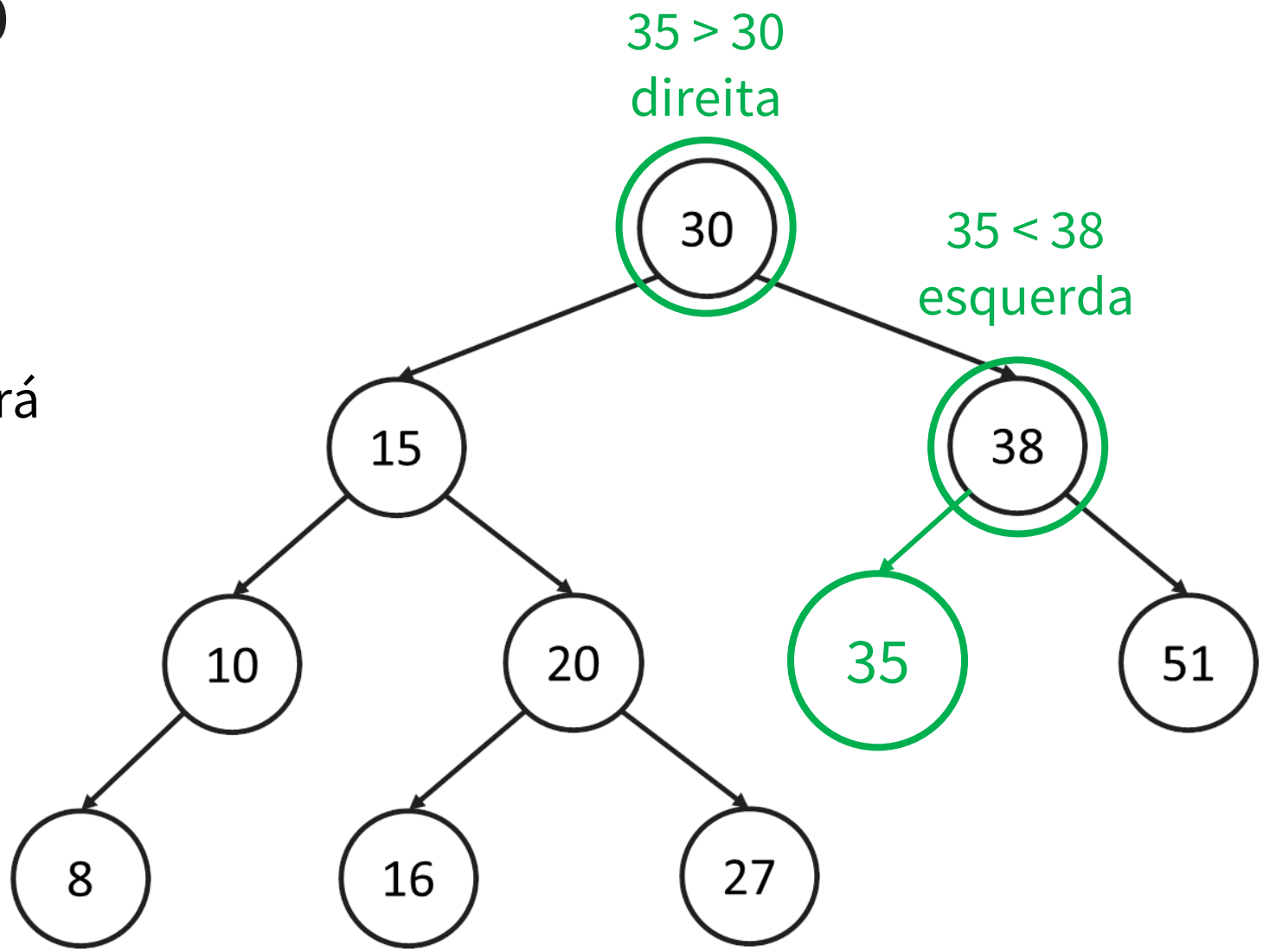
Onde o nó com a chave 35 será inserido na árvore ao lado?



BST – Inserção

Exemplo:
Inserir o elemento 35.

Onde o nó com a chave 35 será inserido na árvore ao lado?



BST – Remoção

- Assim como na busca e inserção, precisamos realizar uma busca na BST para encontrar o nó a ser removido.
- Diferente da busca e inserção, a operação de remoção possui algumas condições que precisamos analisar.
  - Precisamos garantir que, após um nó ser removido, a árvore continue sendo uma BST.

BST – Remoção

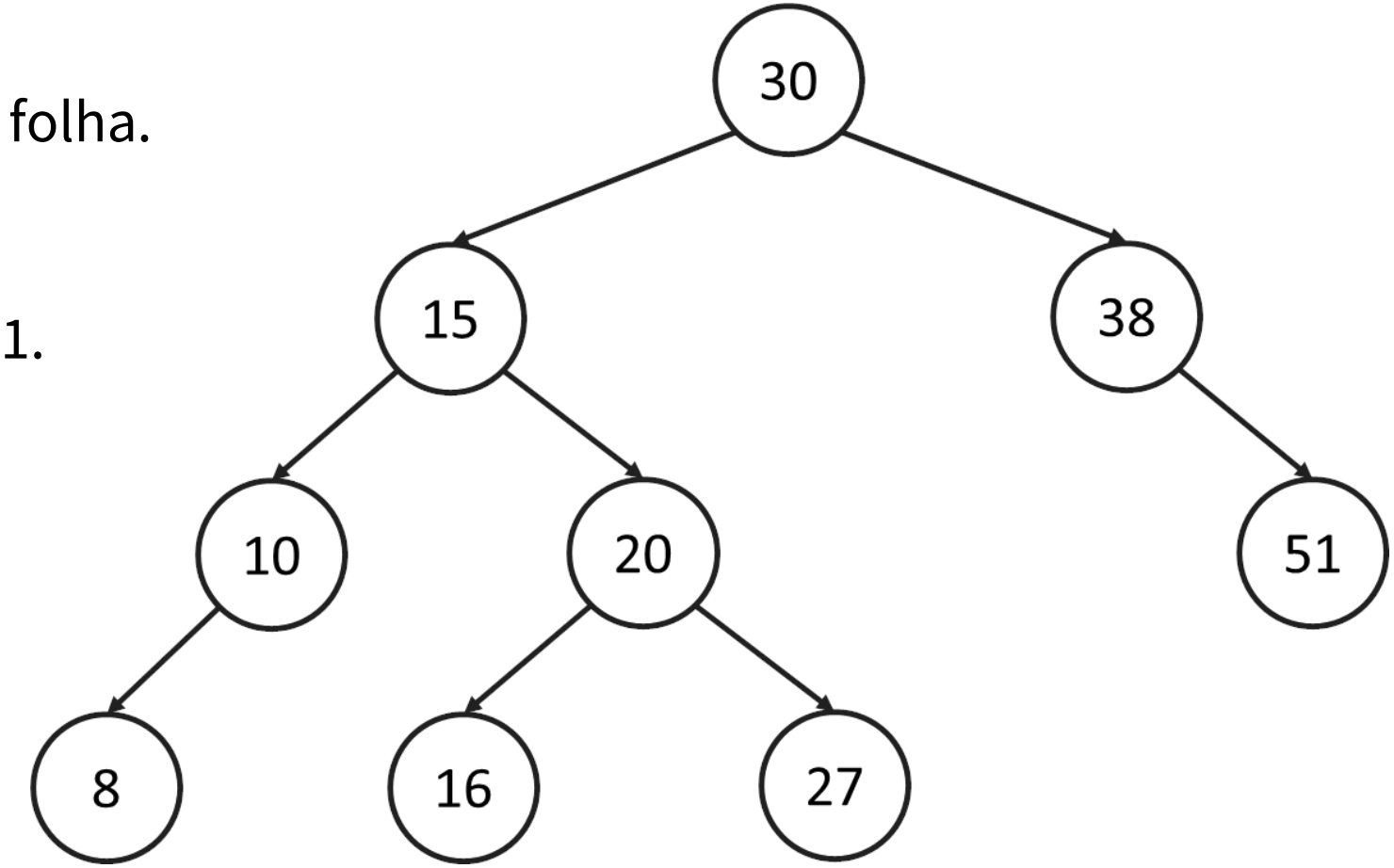
- Existem quatro casos para analisar a remoção de um nó da BST:

1. O nó a ser removido é uma folha.
2. O nó a ser removido não possui filho esquerdo (não possui subárvore da esquerda).
3. O nó a ser removido não possui filho direito (não possui subárvore da direita).
4. O nó a ser removido possui os filhos esquerdo e direito (possui subárvores da esquerda e da direita).

BST – Remoção

Caso 1:
O nó a ser removido é uma folha.

Exemplo:
Remover o nó com chave 51.

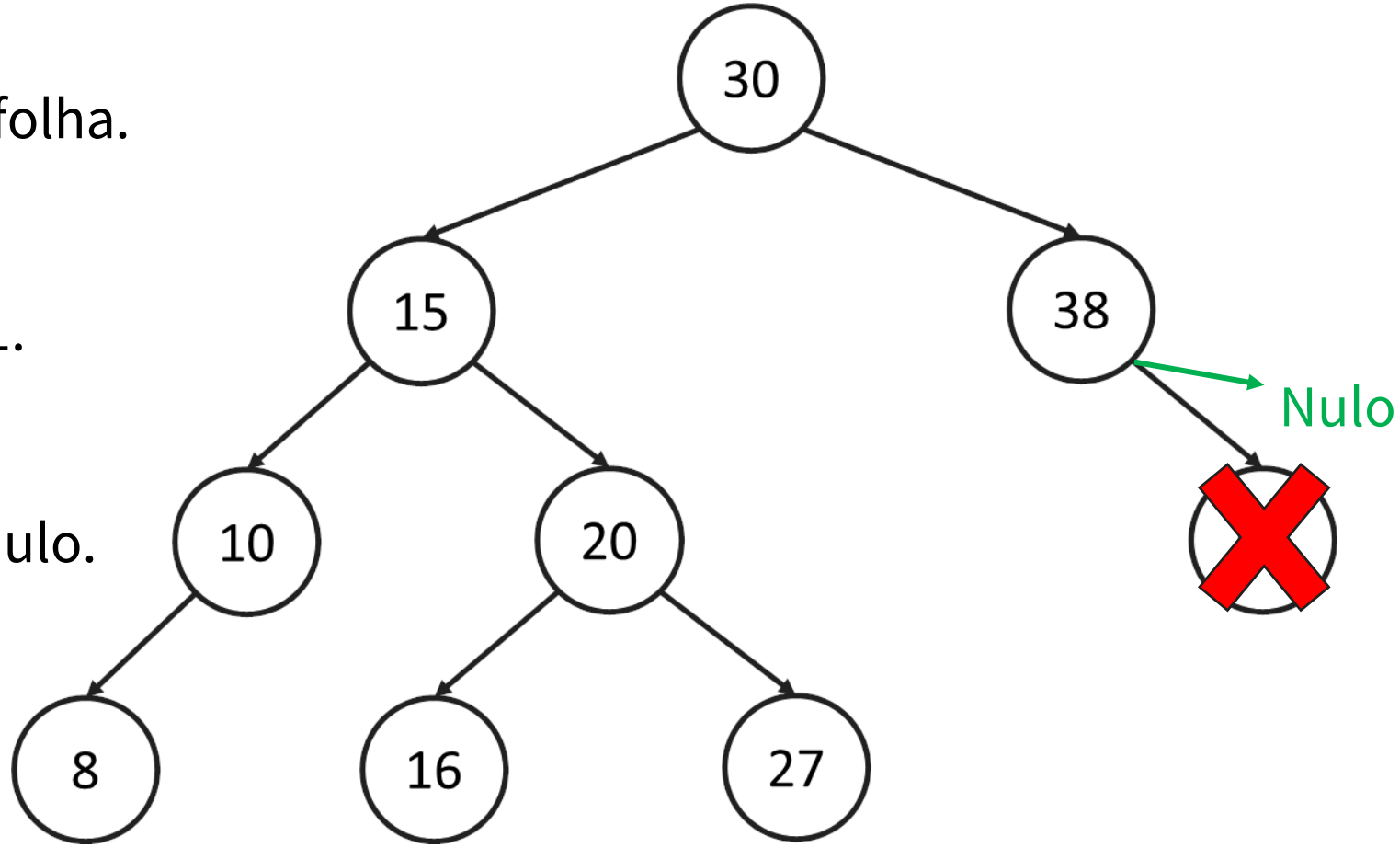


BST – Remoção

Caso 1:
O nó a ser removido é uma folha.

Exemplo:
Remover o nó com chave 51.

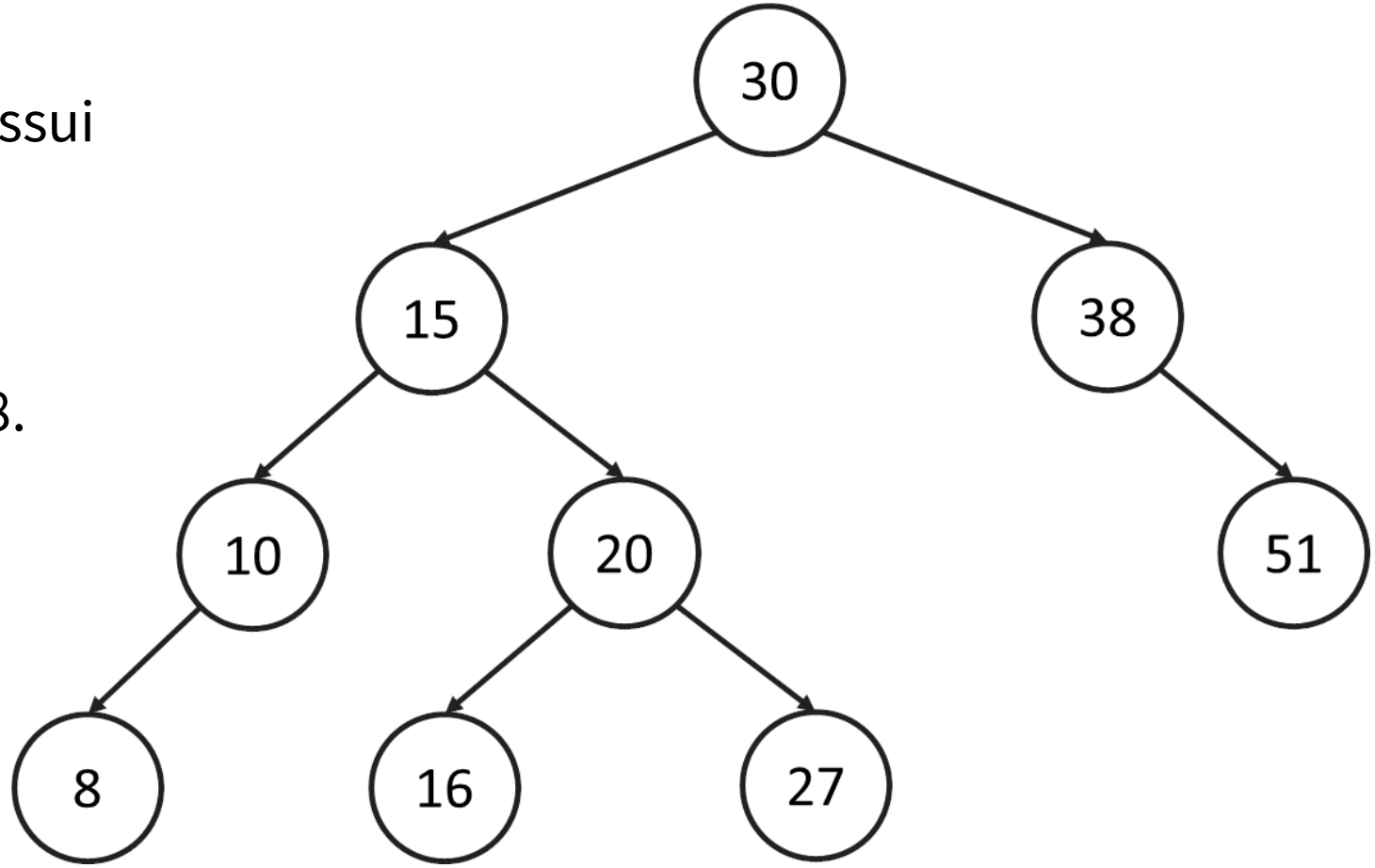
Resultado:
Filho direito de 38 agora é nulo.



BST – Remoção

Caso 2:
O nó a ser removido não possui subárvore esquerda.

Exemplo:
Remover o nó com chave 38.

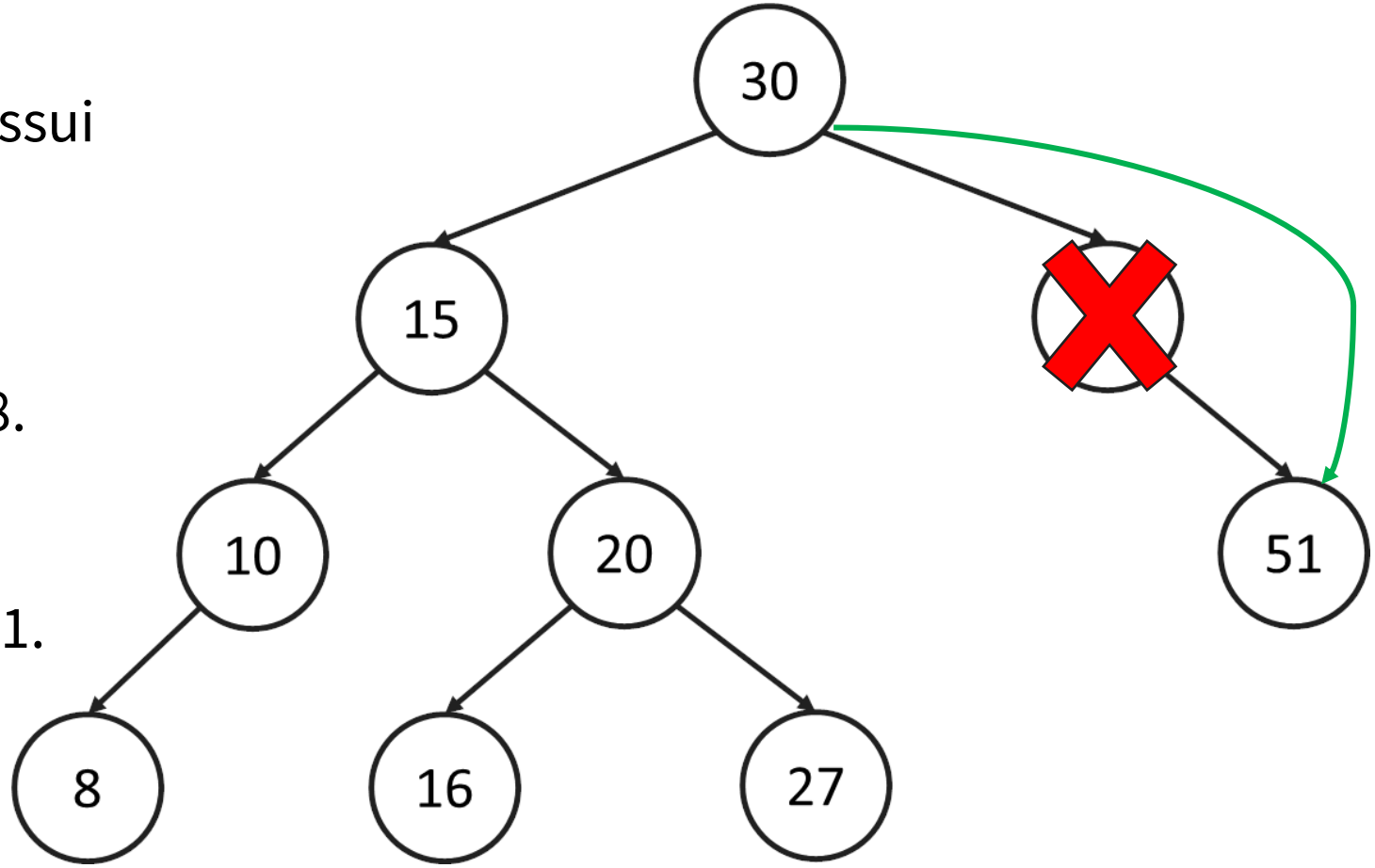


BST – Remoção

Caso 2:
O nó a ser removido não possui subárvore esquerda.

Exemplo:
Remover o nó com chave 38.

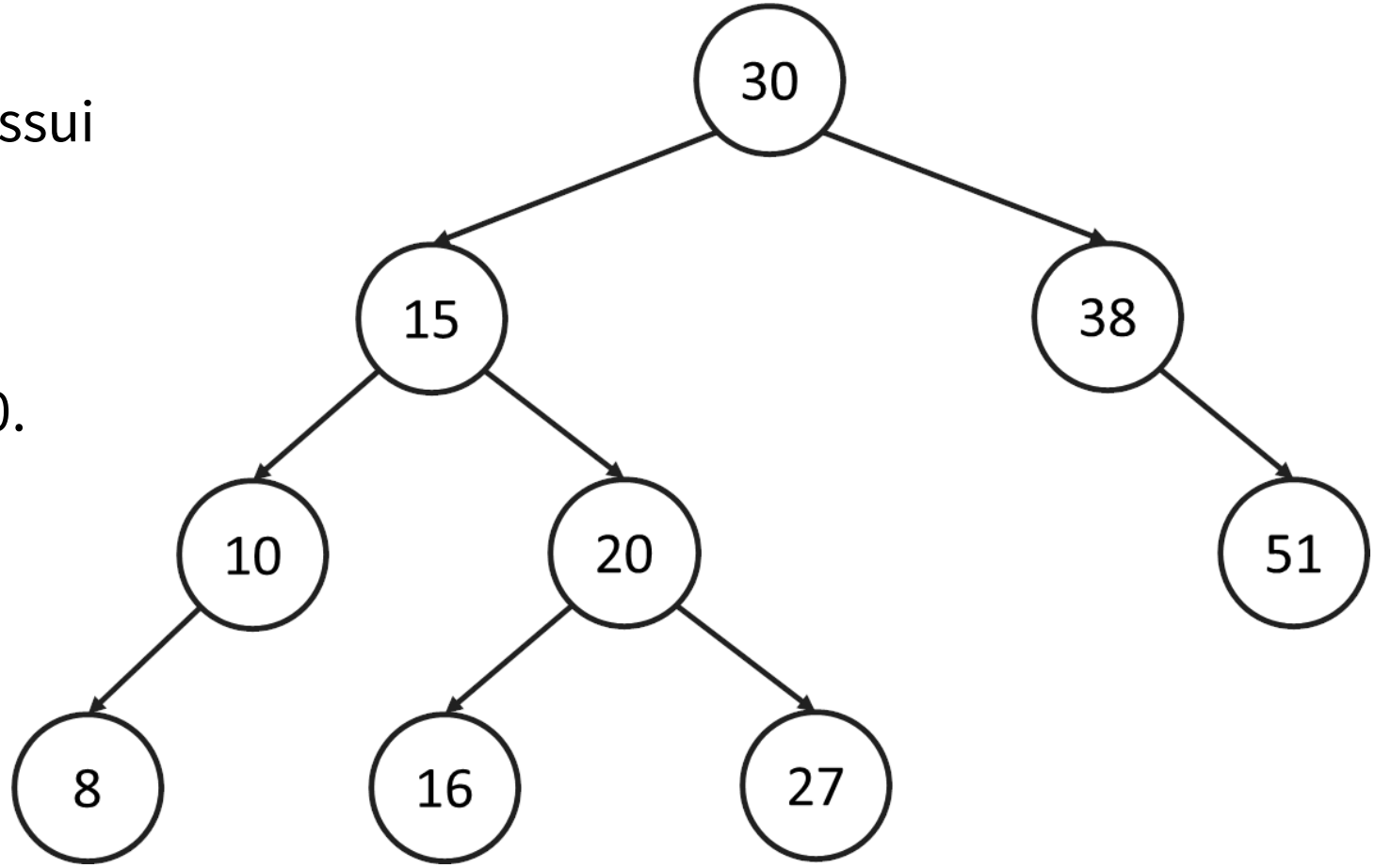
Resultado:
Filho direito de 30 agora é 51.



BST – Remoção

Caso 3:
O nó a ser removido não possui subárvore direita.

Exemplo:
Remover o nó com chave 10.

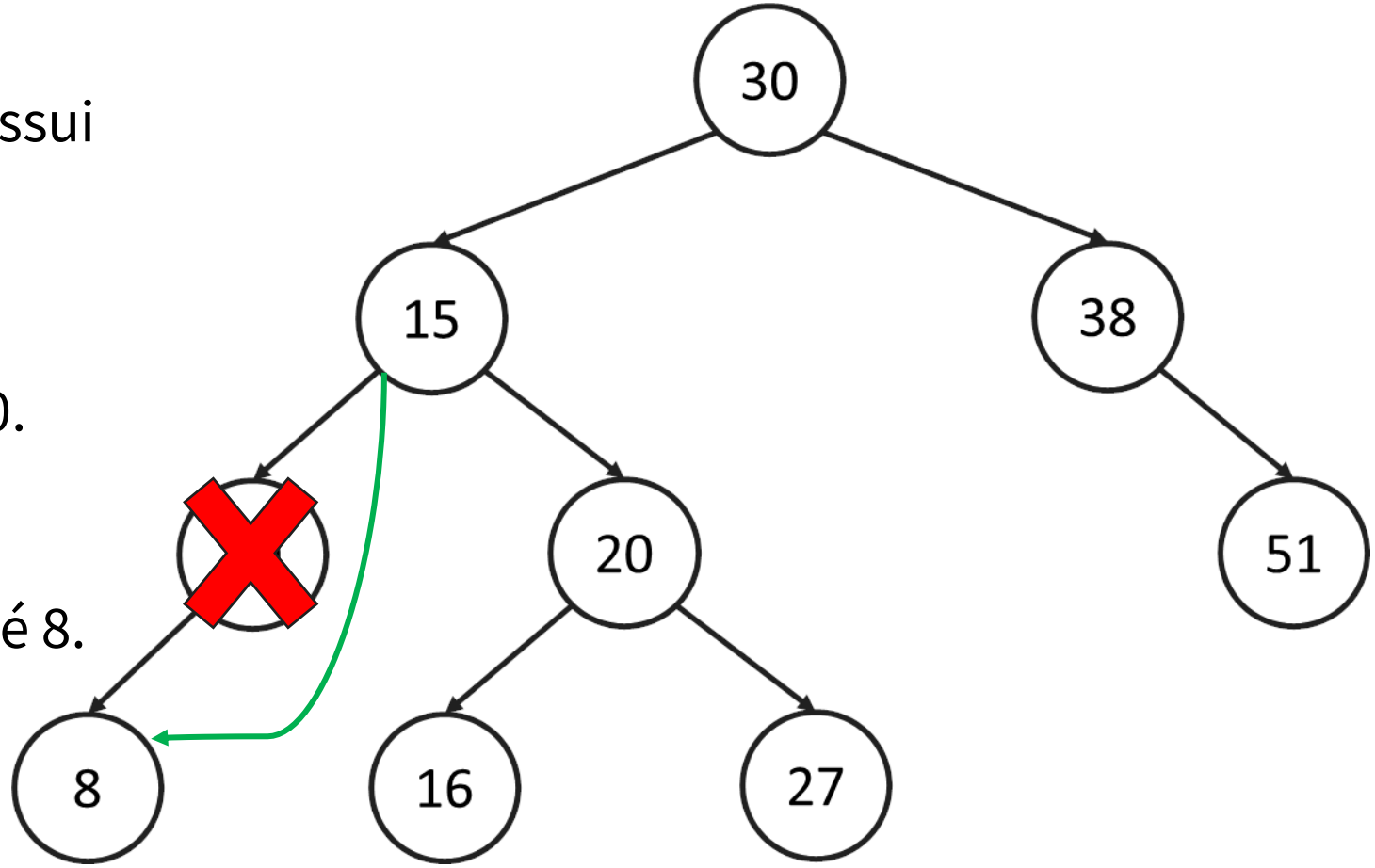


BST – Remoção

Caso 3:
O nó a ser removido não possui subárvore direita.

Exemplo:
Remover o nó com chave 10.

Resultado:
Filho esquerdo de 15 agora é 8.

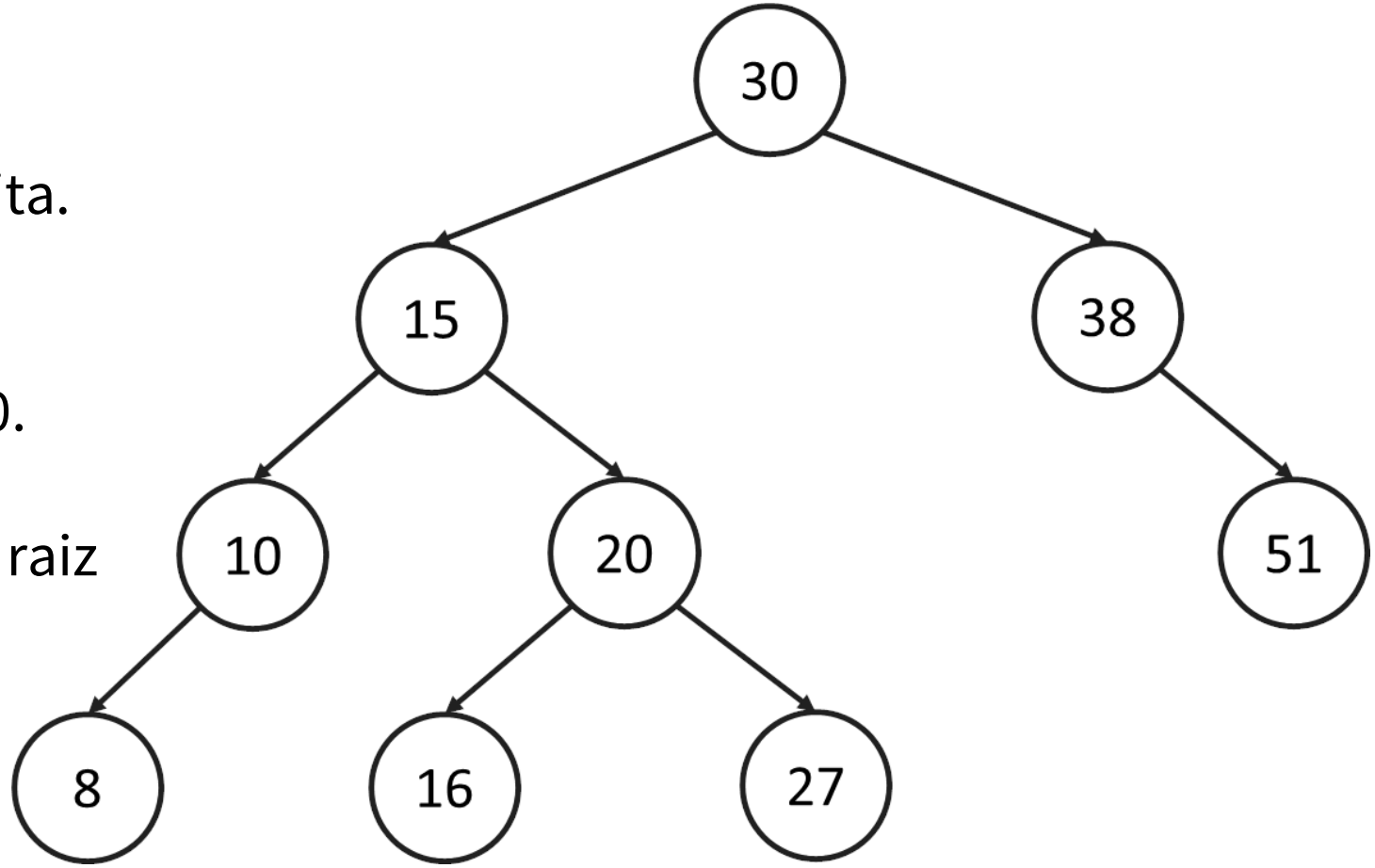


BST – Remoção

Caso 4:
O nó a ser removido possui subárvores esquerda e direita.

Exemplo:
Remover o nó com chave 30.

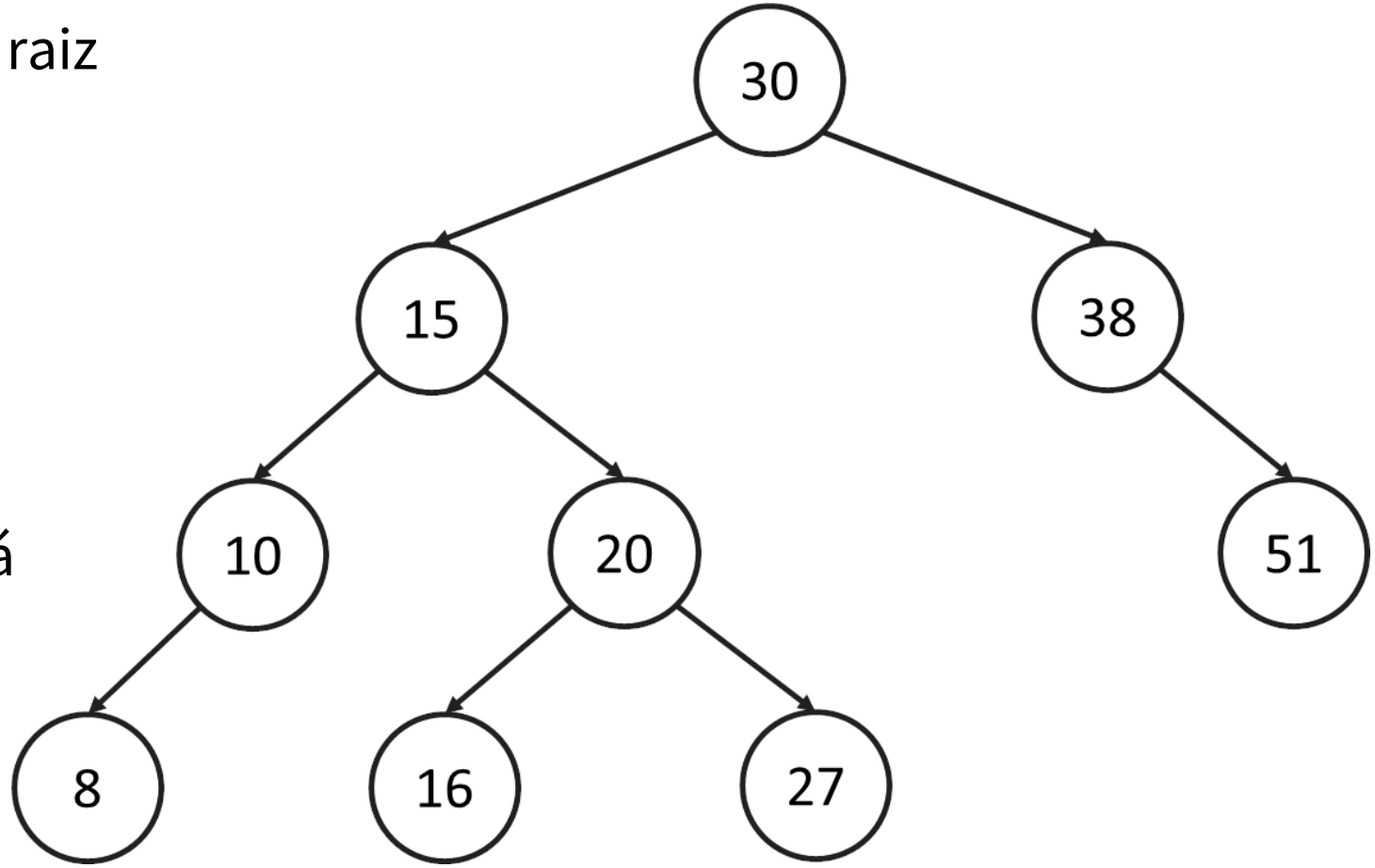
O nó com chave 30 é um nó raiz de uma [sub]árvore.
Qual nó deve virar a raiz da [sub]árvore?



BST – Remoção

O nó com chave 30 é um nó raiz de uma [sub]árvore.
Qual nó deve virar a raiz da [sub]árvore?

Podemos escolher um nó antecessor (27) ou um nó sucessor (38) ao nó que será removido.

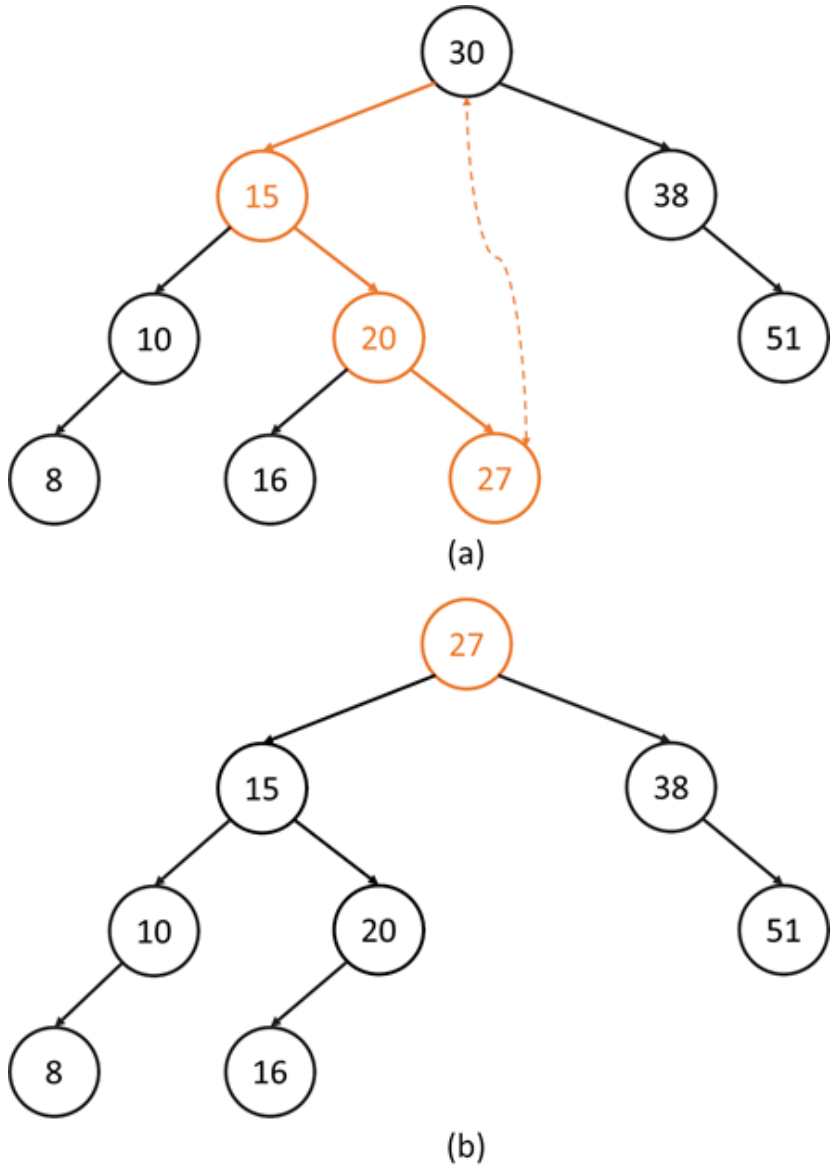


BST – Remoção

Na figura ao lado (a), escolhemos o antecessor do nó a ser removido para se tornar raiz. Nesse exemplo, o nó com chave 27.

Em (b), o nó 27 saiu de sua posição original e tomou o lugar do nó 30, que foi removido.

Observe que a árvore continua sendo uma BST.

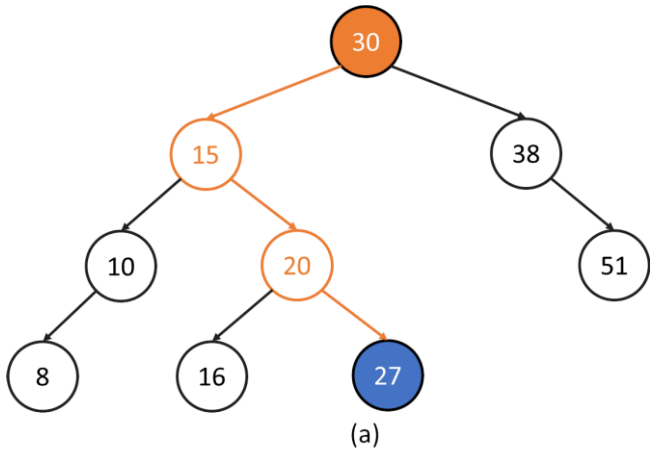


BST – Antecessor de um nó T

(a) Se o nó T possui subárvore esquerda:
- A chave de maior valor dentre as chaves de menor valor (comparado com a chave de T).
- Busca na subárvore esquerda de T, até encontrar um nó sem filho direito.

(b) Se o nó T não possui uma subárvore esquerda:
- Devemos “subir na árvore” até encontrar um nó que possui valor menor que T.

(c) Se o nó T é o menor valor da BST, então não há antecessor de T.



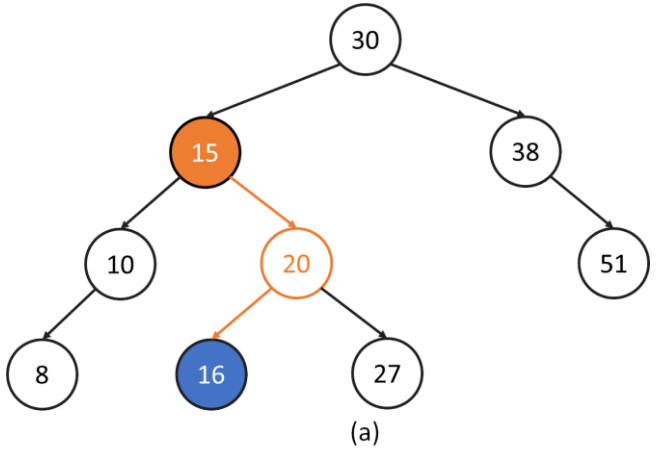
(laranja: nó a ser removido; azul: nó antecessor)

BST – Sucessor de um nó T

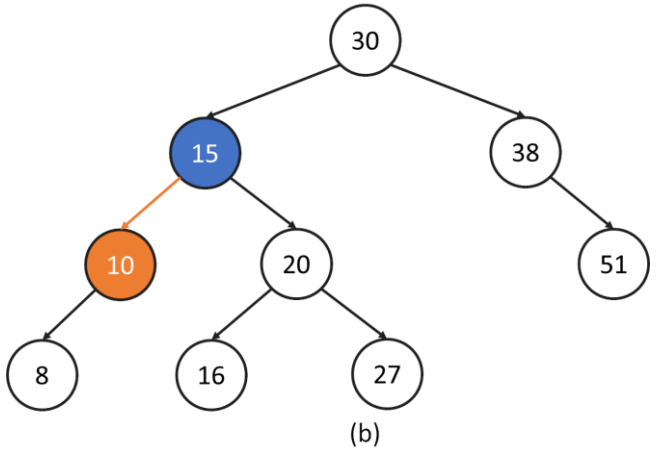
(a) Se o nó T possui uma subárvore direita:
- A chave de menor valor dentre as chaves de maior valor (comparado com a chave de T).
- Busca na subárvore direita de T, até encontrar um nó sem filho esquerdo.

(b) Se o nó T não possui uma subárvore direita:
- Devemos “subir na árvore” até encontrar um nó que possui valor maior que T.

(c) Se o nó T é o maior valor da BST, então não há sucessor de T.



(a)



(b)

(laranja: nó a ser removido; azul: nó sucessor)