Árvores Binárias

Prof. Dr. Eleandro Maschio
Tecnologia em Sistemas para Internet
Câmpus Guarapuava
Universidade Tecnológica Federal do Paraná (UTFPR)

Definição de Árvore Binária

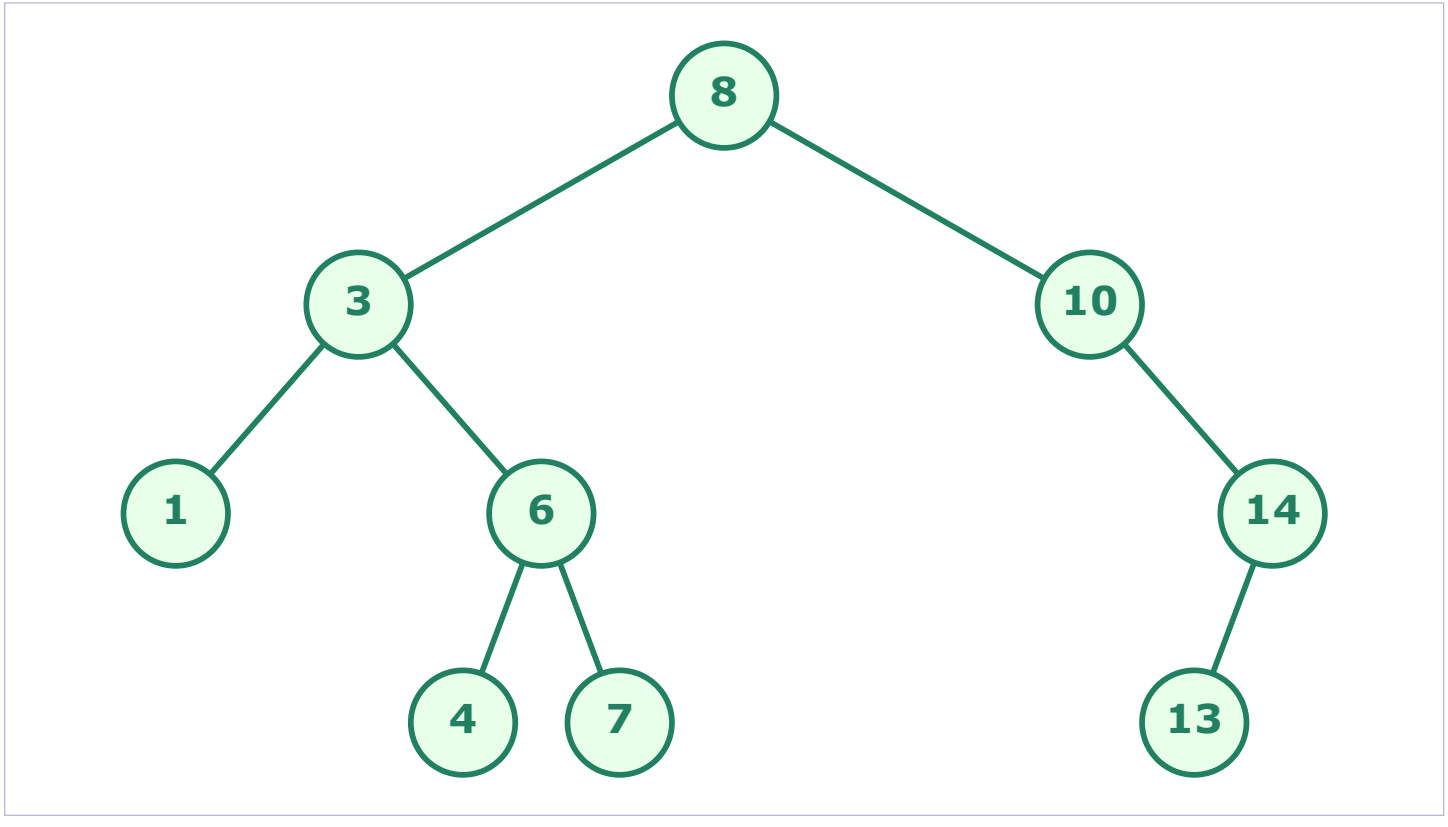
- Trata-se de um **tipo específico** de árvore cujo **grau é 2**, ou seja, cada nó pode ter no máximo dois filhos.
- Desta forma, os nós componentes podem ter **zero** (folhas), **um** ou **dois filhos**.
- Tais filhos são denominados **filho da esquerda** ou **filho da direita** do nó, conforme a posição na representação gráfica.
- Figuram entre os tipos mais **simples** e **úteis** de árvores na Ciência da Computação, tendo principal destaque as **árvores de busca binária**.

Definição de Árvore de Busca Binária

- Um árvore de busca binária, adicionalmente, possui a propriedade de que todo **filho esquerdo** possui valor **menor** do que o de seu pai; e todo **filho direito** possui valor **maior ou igual** ao do seu pai.

- Portanto, a **sub-árvore esquerda** possui somente valores **menores** do que o nó pai, ao passo que a **sub-árvore direita** armazena valores **maiores ou iguais** ao nó pai.

Definição de Árvore de Busca Binária



Definição de Árvore de Busca Binária

Portanto, toda árvore binária possui as seguintes propriedades:

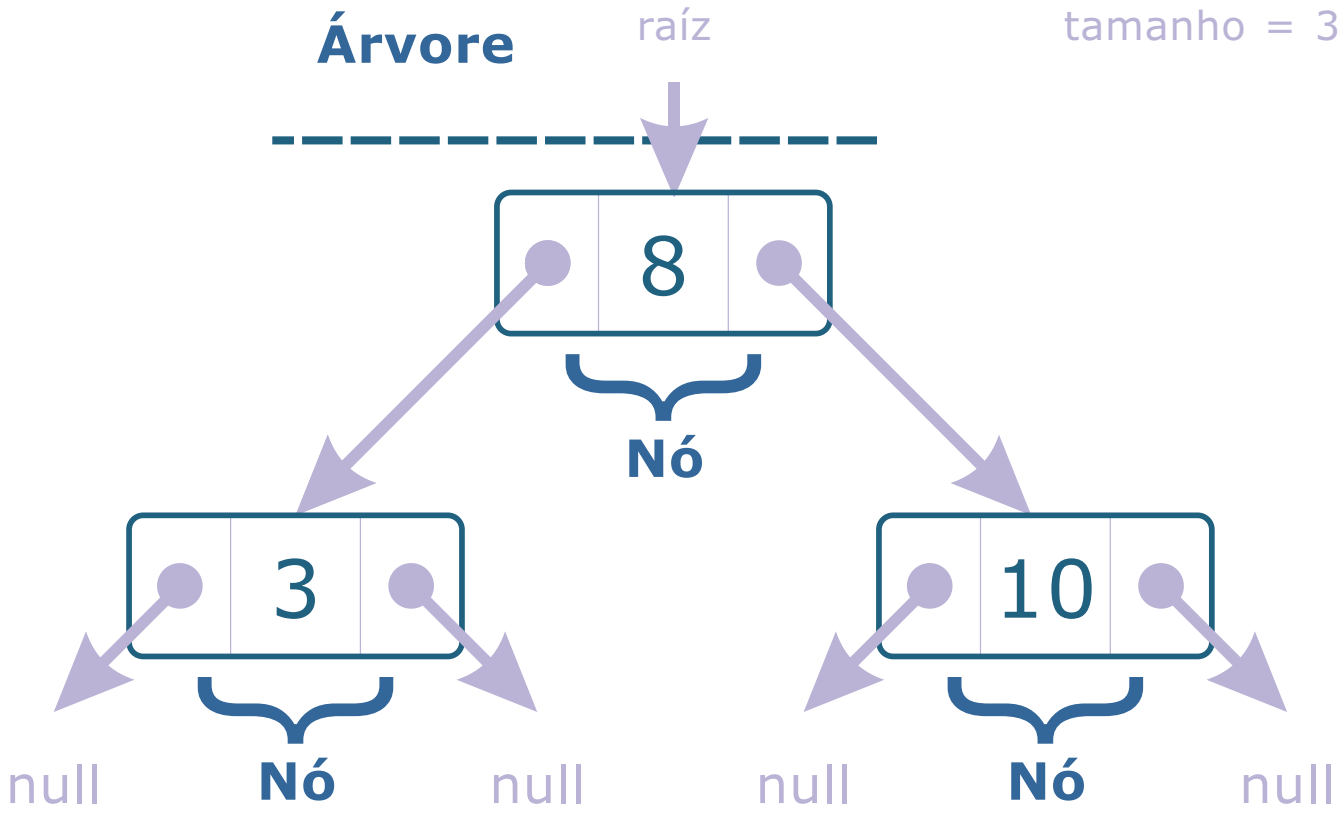
(1) Todos os nós de uma sub-árvore **direita** são **maiores**, ou **iguais**, do que o nó raiz.

(2) Todos os nós de uma sub-árvore **esquerda** são **menores** do que o nó raiz.

(3) Cada **sub-árvore** é também uma **árvore binária**.

(4) O grau de um **nó** representa o respectivo número de sub-árvores.

Modelagem em Representação Encadeada



Modelagem em Representação Encadeada

Consiste basicamente na construção de duas classes:

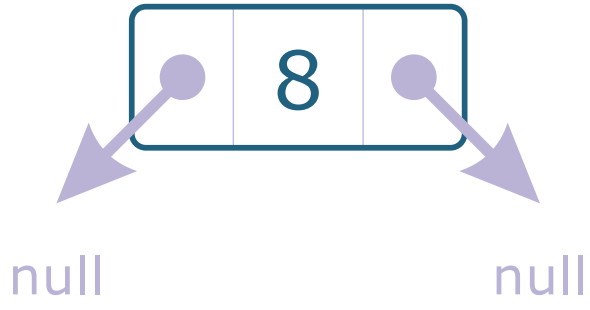
- **Nó**
  Representa individualmente **cada elemento** inserido da árvore.

- **Árvore**
  A **estrutura de dados** propriamente dita. Serve para **coordenar** o **encadeamento** de nós conforme as **regras estruturais** de uma árvore de busca binária.

Classe *Nó*

Classe que modela um objeto com os atributos essenciais:

- **valor:** item (um inteiro, por exemplo) a ser **armazenado** pelo nó.
- **filhoEsquerdo** e **filhoDireito: referências** a outros objetos do tipo *Nó* que correspondem aos **filhos** do nó corrente. Ambos são inicializados como **`null`**.

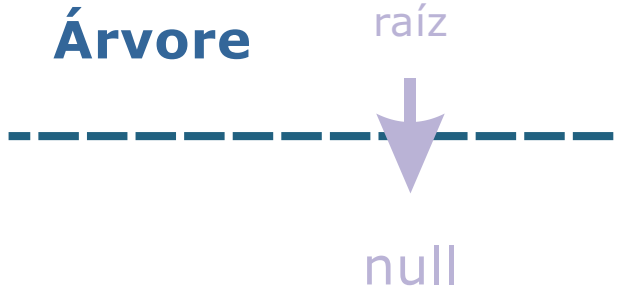



```
public class No
{
    private int valor;
    private No filhoDireito,
              filhoEsquerdo;
    // ...
}
```

Classe *Árvore*

Classe responsável pela **manipulação dos nós** a fim de que constituam uma árvore de busca binária.

- **raiz:** atributo essencial que é uma **referência a um objeto** do tipo *Nó*, inicializada como **`null`**.
- **tamanho:** facilitador que permite a **contagem** dos nós da árvore.




```
public class Arvore
{
    private No raiz;
    private int tamanho;
    // ...
}
```

Operações

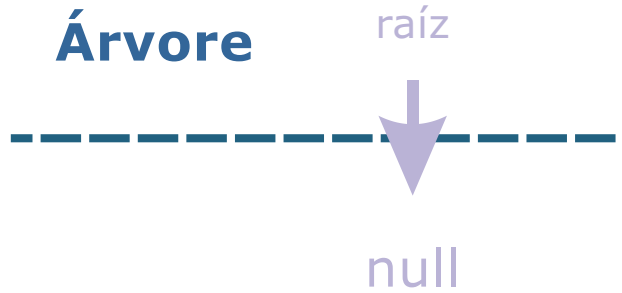
São operações relacionadas às árvores de busca binárias:

- Verificar se a árvore **está vazia**.
- **Inserir** um valor.
- **Buscar** um valor.
- **Percorrer** os nós da árvore (mais basicamente para impressão).
- Retornar o **maior** valor da árvore.
- Retornar o **menor** valor da árvore.
- **Alterar** um valor (em alguns casos).
- **Excluir** um valor.

Verificação de Árvore Vazia

Uma árvore está vazia se não possuir nó algum. Constata-se, portanto:

- se a **raiz** ainda referenciar `null`; ou
- se o **tamanho** for igual a **zero**.



Inserção

- Em uma árvore de busca binária, os elementos são **sempre inseridos** em **nós folhas**.
- Cabe, à inserção, determinar o **local adequado** para adicionar o novo nó.

Inserção

Lógica de Inserção

- Se a árvore estiver **vazia**, então o novo nó será a **raiz da árvore**.
- Caso contrário:
  - se o valor a ser inserido for **menor** do que a **raiz**, siga pela sub-árvore esquerda da raiz, tendo o **filho esquerdo como raiz** da sub-árvore onde o novo nó será inserido.
  - caso contrário, ou seja, se o valor a ser inserido for **maior** (ou **igual**) do que a raiz, siga pela sub-árvore direita da raiz, tendo o **filho direito como raiz** da sub-árvore onde o novo nó será inserido.
- Este é um processo **recursivo** (mas pode ser implementado iterativamente) e o novo nó **será inserido** quando uma sub-árvore estiver **raiz vazia**, ou seja, quando um nó folha for encontrado.

Inserção

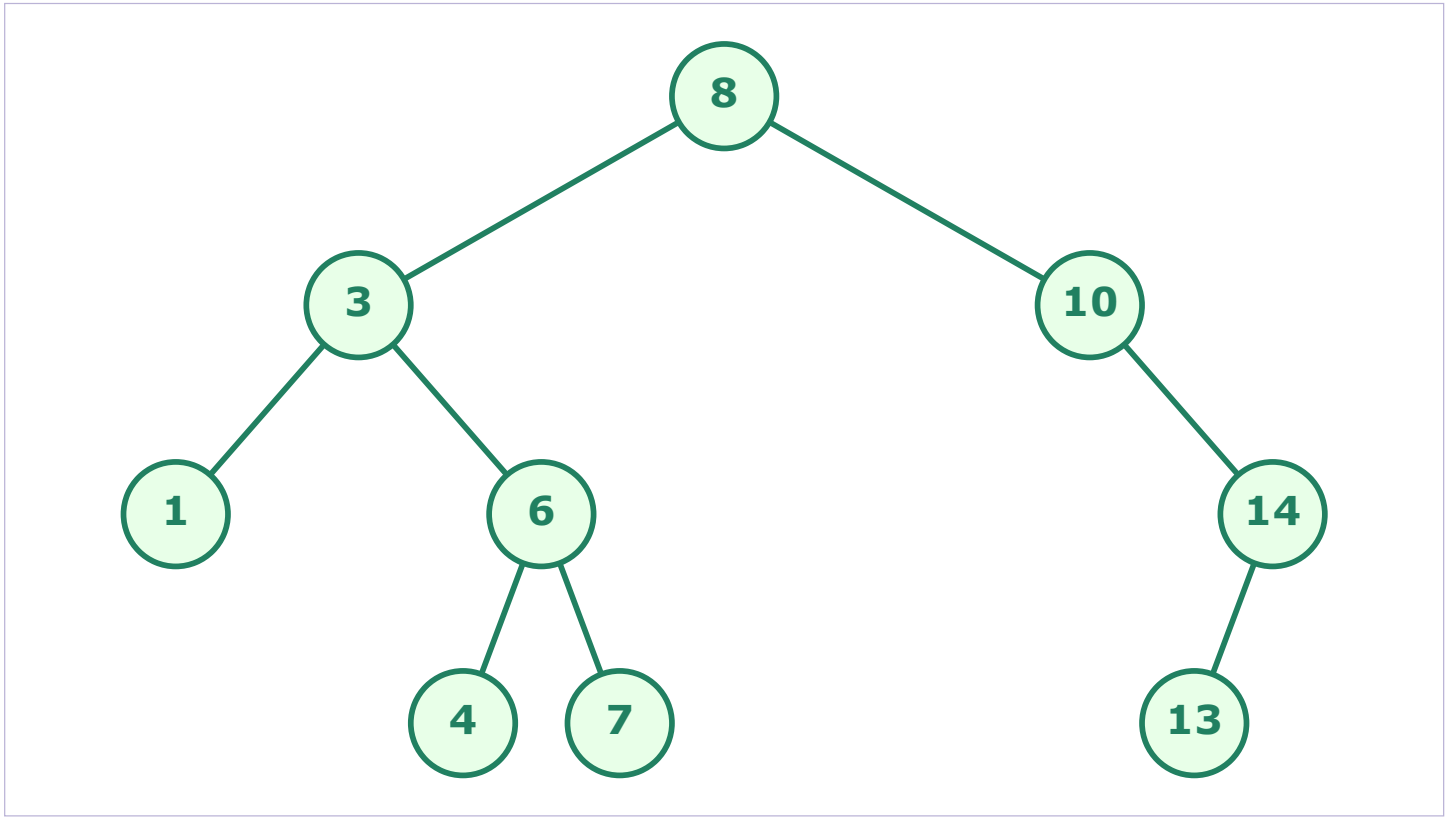
Lógica de Inserção

Simulemos, em uma representação simbólica de árvore binária de busca, a inserção dos inteiros:

50, 126, 28, 77, 12, 429, 39, 84, 256, 31.

Retorno do Maior Elemento

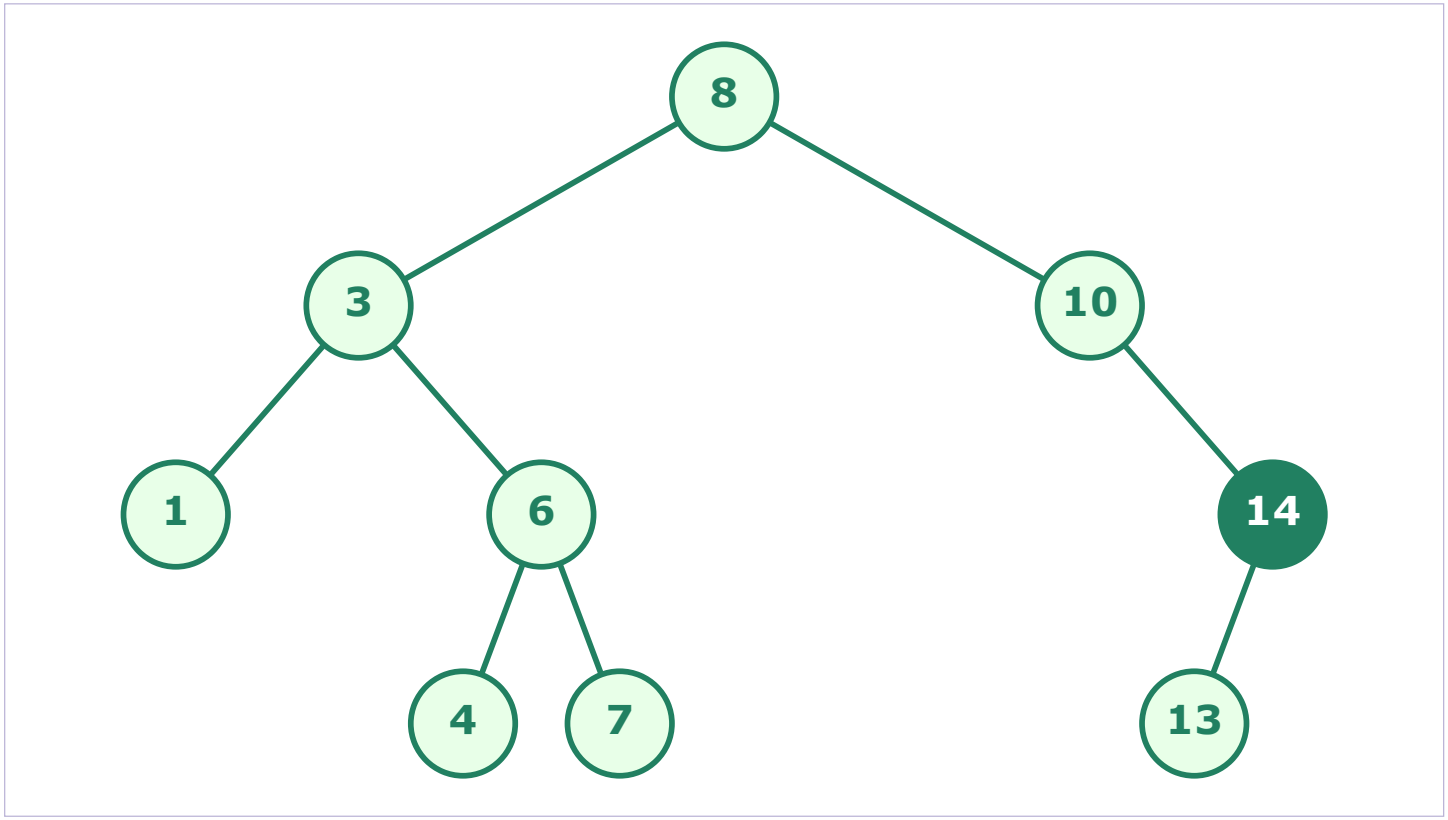
Lógica de Retorno do Maior Elemento



Retorno do Maior Elemento

Lógica de Retorno do Maior Elemento

Numa árvore não-vazia, siga a partir da raiz, sempre pela **sub-árvore direita**, até **não haver filho** da direita. Será o maior elemento.



Percorrimento

- Ou **percurso**.
- Consiste em **processar**, de forma sistemática, todos os nós de uma árvore, passando por cada nó **apenas uma vez**, partindo da raiz.
- Existem diversos motivos para se percorrer uma árvore. A **impressão** é o mais básico destes.
- Existem três tipos básicos de percurso:
    - Em ordem (simétrica).
    - Em pré-ordem.
    - Em pós-ordem.

Percorrimento

Lógica de Percurso Em Ordem (Simétrica)

- Remetem às expressões infixas:

  ```
  ( a + b ) * ( c - d )
  ```

- Se a raiz estiver vazia, retorne.
- Caso contrário:
  - Percorra em ordem a sub-árvore **esquerda**.
  - Processe/exiba a **raiz**.
  - Percorra em ordem a sub-árvore **direita**.

Percorrimento

```
public class No
{
    // ...

    public String toString()
    {
        return Integer.toString(this.valor);
    }
}
```

Percorrimento

```
public class Arvore
{
    // ...

    public void percorre()
    {
        this.percorre(this.raiz);
    }

    private void percorre(No raizAtual)
    {
        if (raizAtual != null)
        {
            this.percorre(raizAtual.getFilhoEsquerdo());
            System.out.println(raizAtual.toString());
            this.percorre(raizAtual.getFilhoDireito());
        }
    }
}
```

Percorrimento

Lógica de Percurso Em Pré-Ordem

- Remetem às expressões pré-fixas:

  `* + a b - c d`

- Se a raiz estiver vazia, retorne.
- Caso contrário:
  - Processe/exiba a **raiz**.
  - Percorra em ordem a sub-árvore **esquerda**.
  - Percorra em ordem a sub-árvore **direita**.

Lógica de Percurso Em Pós-Ordem

- Remetem às expressões pós-fixas:

```
a b + c d - *
```

- Se a raiz estiver vazia, retorne.
- Caso contrário:
  - Percorra em ordem a sub-árvore **esquerda**.
  - Percorra em ordem a sub-árvore **direita**.
  - Processe/exiba a **raiz**.

Busca

- Cabe à busca encontrar um elemento **baseada nas propriedades estruturais** da árvore de busca binária.
- Ou seja, se o elemento procurado não for a raiz da árvore (ou da sub-árvore), deve-se **direcionar** a pesquisa para o lado **esquerdo** ou **direito**.

Busca

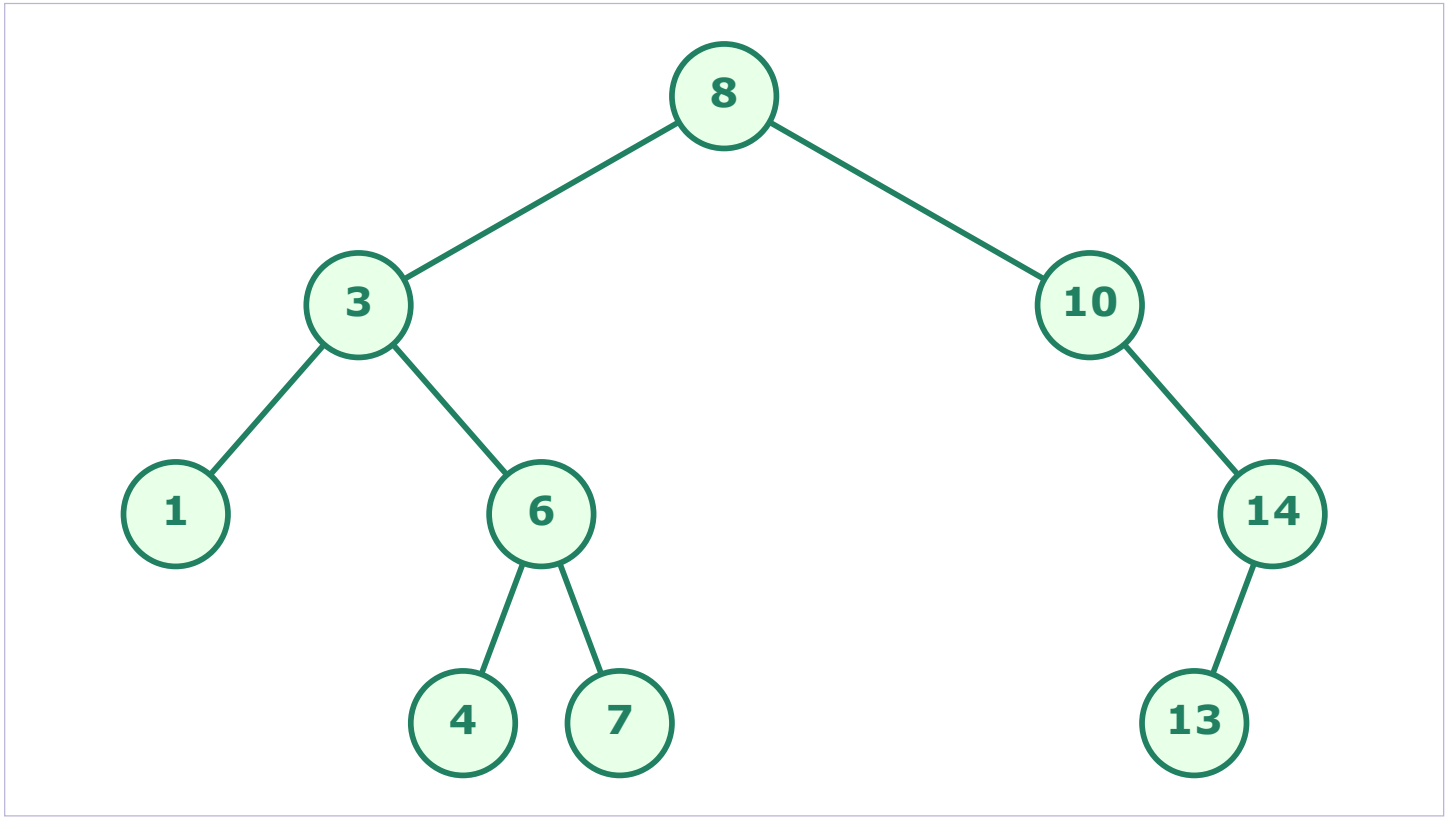
Lógica de Busca

- Se a raiz da árvore for **nula**, a árvore está vazia, portanto não possui o valor procurado.
- Caso contrário, se o valor armazenado na **raiz for igual** àquele procurado, a busca foi bem sucedida.
- Se o valor procurado é **menor** do que a raiz, siga procurando pela **sub-árvore esquerda** da raiz. Trabalhe recursivamente esta busca tendo o **filho esquerdo** como **raiz** da sub-árvore a ser pesquisada.
- Caso contrário, se o valor procurado é **maior ou igual** à raiz, a siga procurando pela **sub-árvore direita** da raiz. Trabalhe recursivamente esta busca **tendo o filho direito** como **raiz** da sub-árvore a ser pesquisada.

Lógica de Busca

- Continue a busca, recursivamente, até que o **valor seja encontrado** ou que a raiz da sub-árvore pesquisada seja **nula**.

Lógica de Busca



Pesquisar os inteiros **4** e **26**.

Alteração

- Caso o único dado armazenado em um nó seja o respectivo valor, **critério de alocação** deste nó na inserção, alterá-lo impactaria em fazer com que a árvore deixasse de ser binária de busca, na maioria das vezes.

- Nas situações em que houver **mais dados associados**, além do valor, pode ser interessante permitir a alteração destes outros mediante a busca do nó pelo respectivo valor.

Exclusão

Para a exclusão de um nó, consideram-se **três casos** distintos:

(1) Exclusão de um nó **folha**.

(2) Exclusão de um nó com um **único filho**.

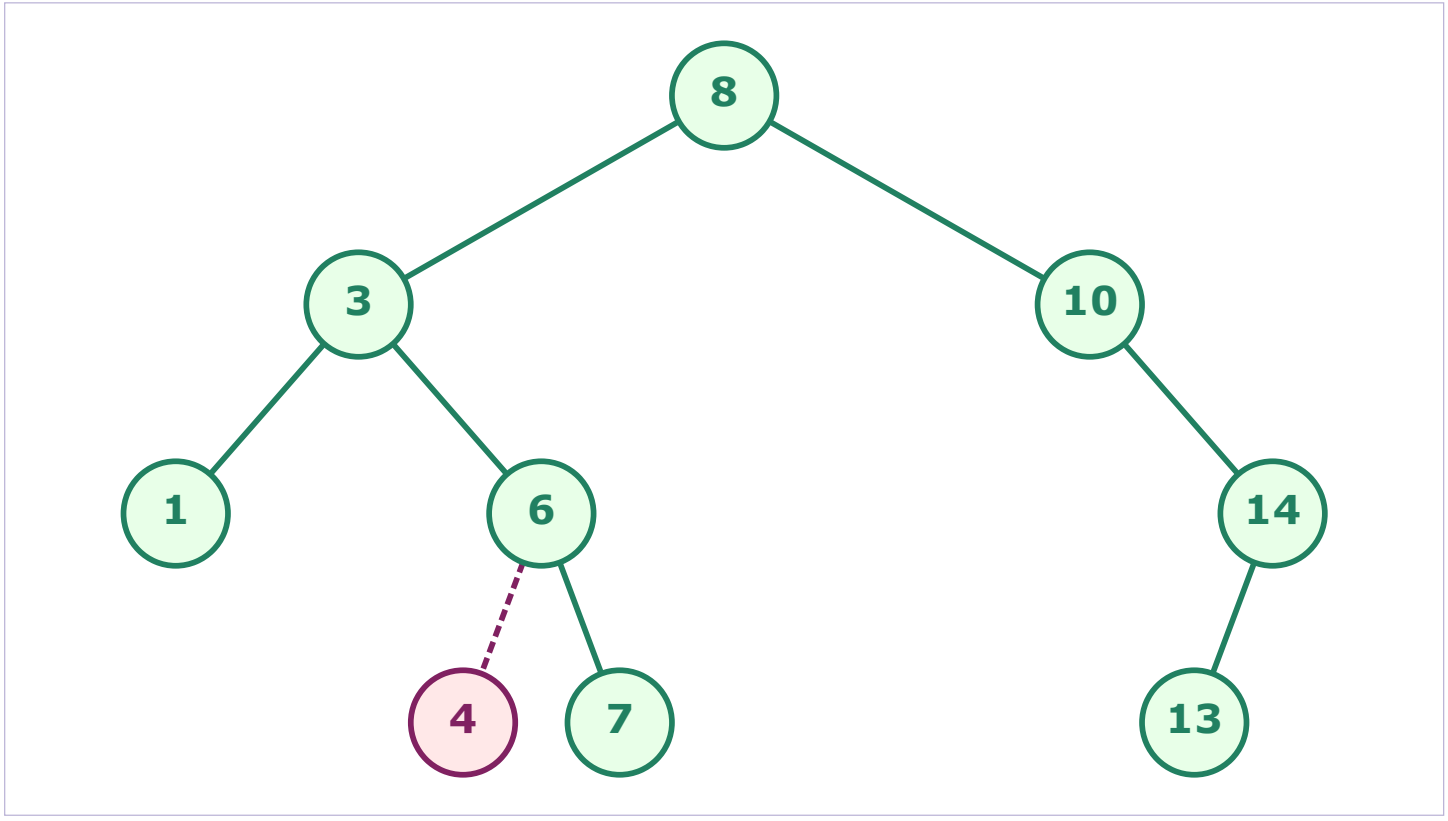
(3) Exclusão de um nó com **dois filhos**.

Lembre-se, contudo, que primeiro precisamos chegar até o nó (**busca**).

Exclusão

Lógica para Exclusão de um Nó Folha

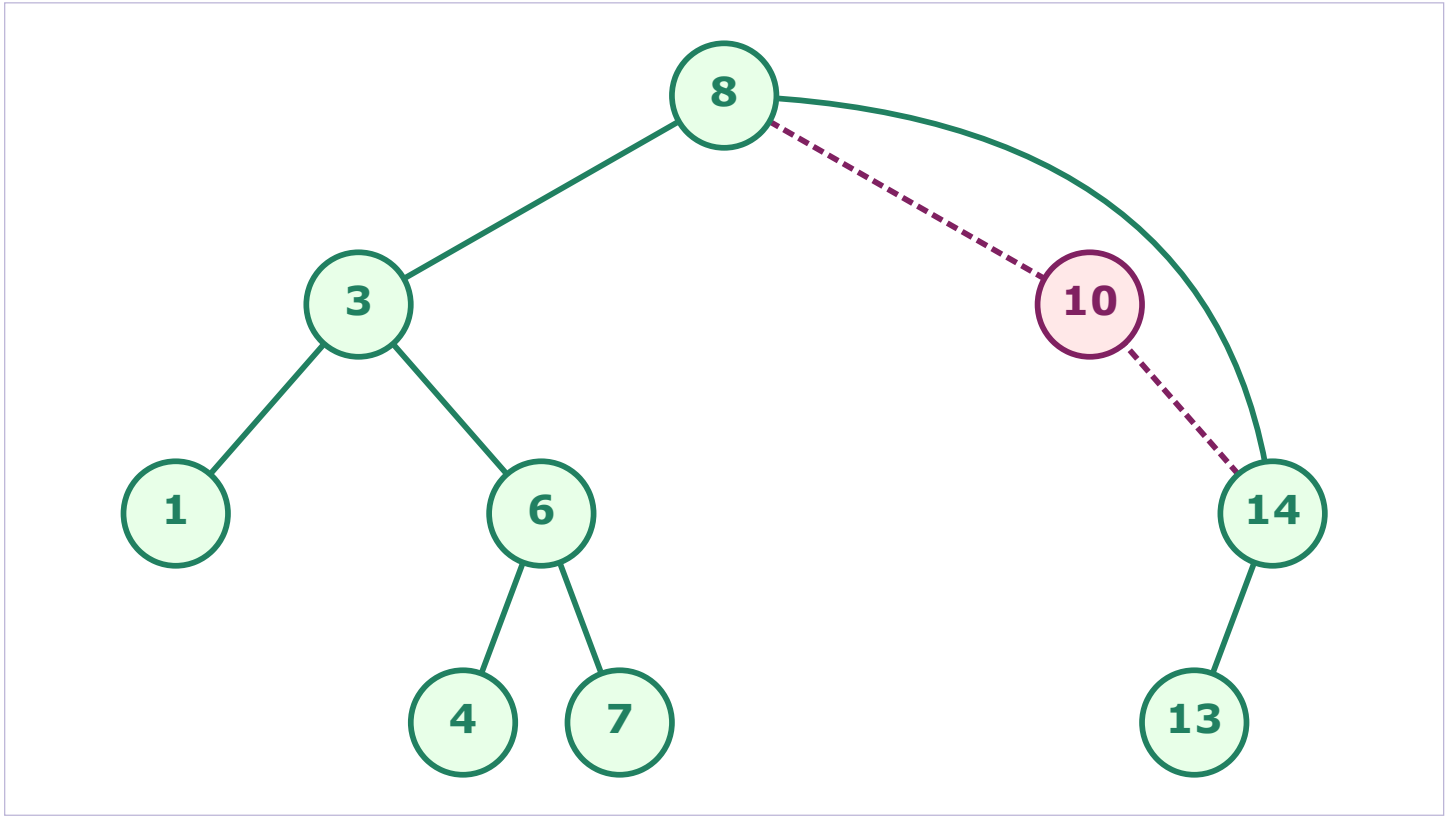
Basta removê-lo da árvore, ou seja, **anule a referência** para o nó em seu pai.



Exclusão

Lógica para Exclusão de um Nó com um Único Fillho

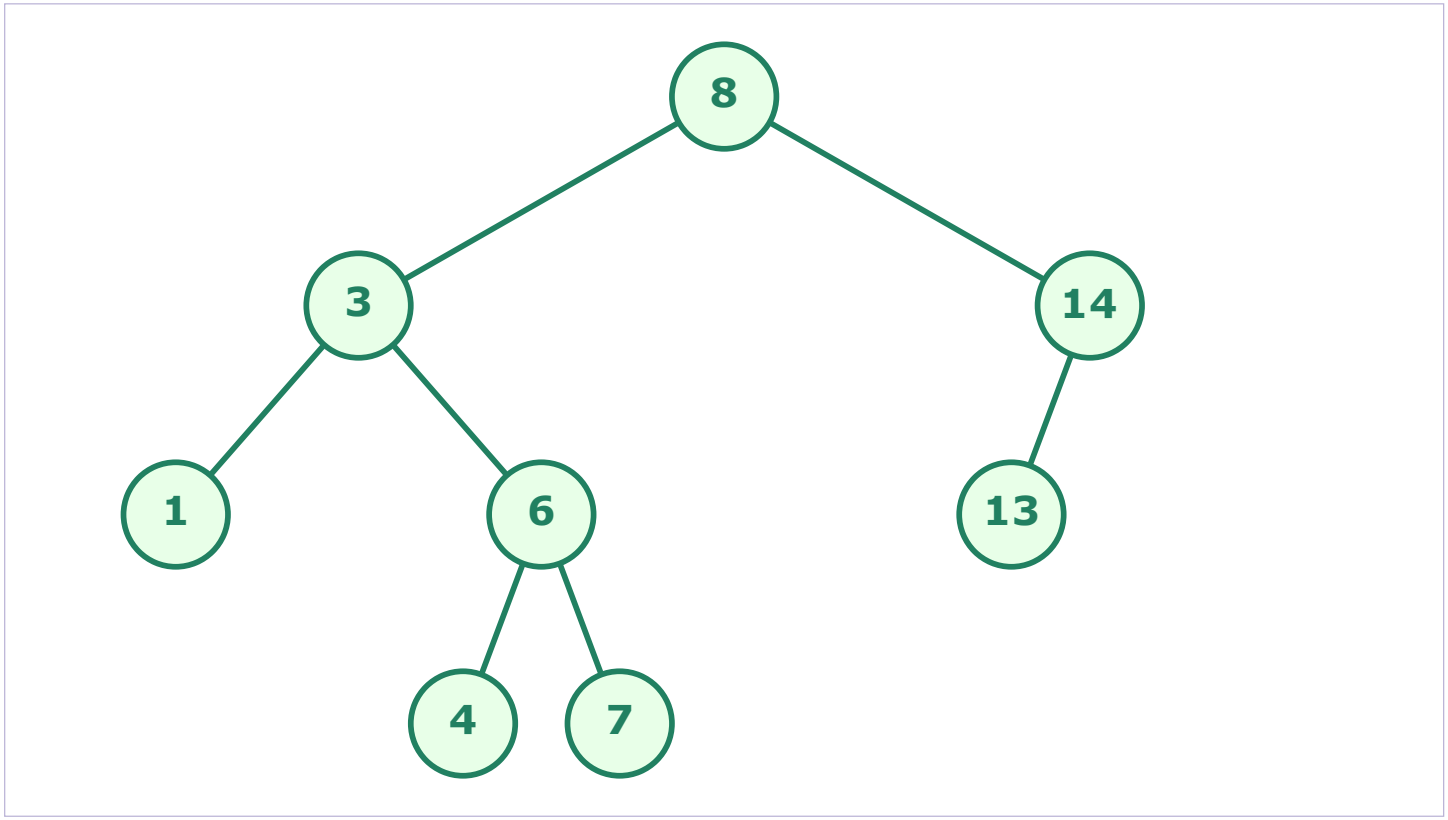
Substitua o nó pelo seu único filho. Ou seja, a referência para o nó, no pai, é substituída pela **referência ao seu filho**.



Exclusão

Lógica para Exclusão de um Nó com um Único Fillho

Substitua o nó pelo seu único filho. Ou seja, a referência para o nó, no pai, é substituída pela **referência ao seu filho**.

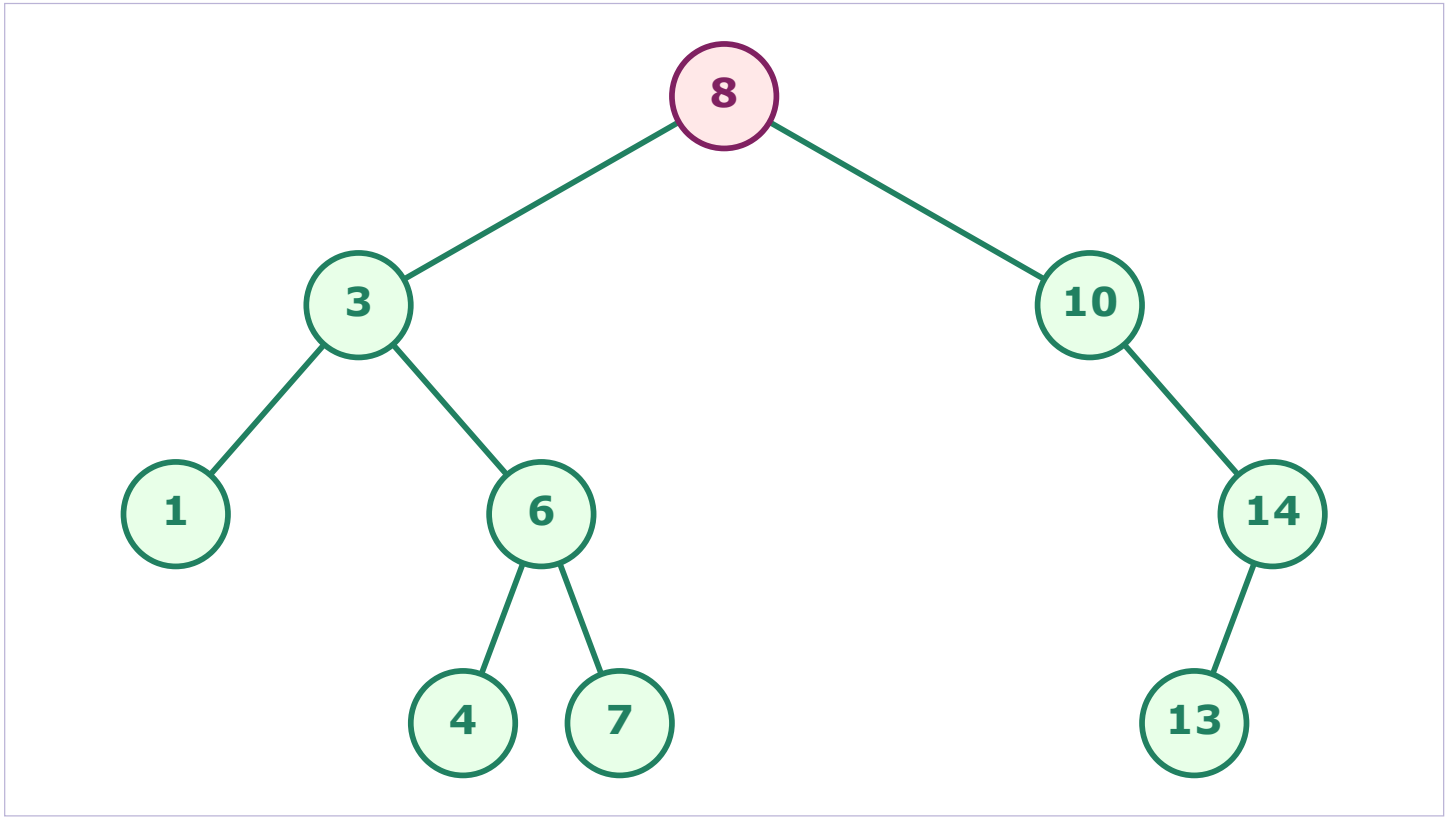


Exclusão

Lógica para Exclusão de um Nó com Dois Fillhos

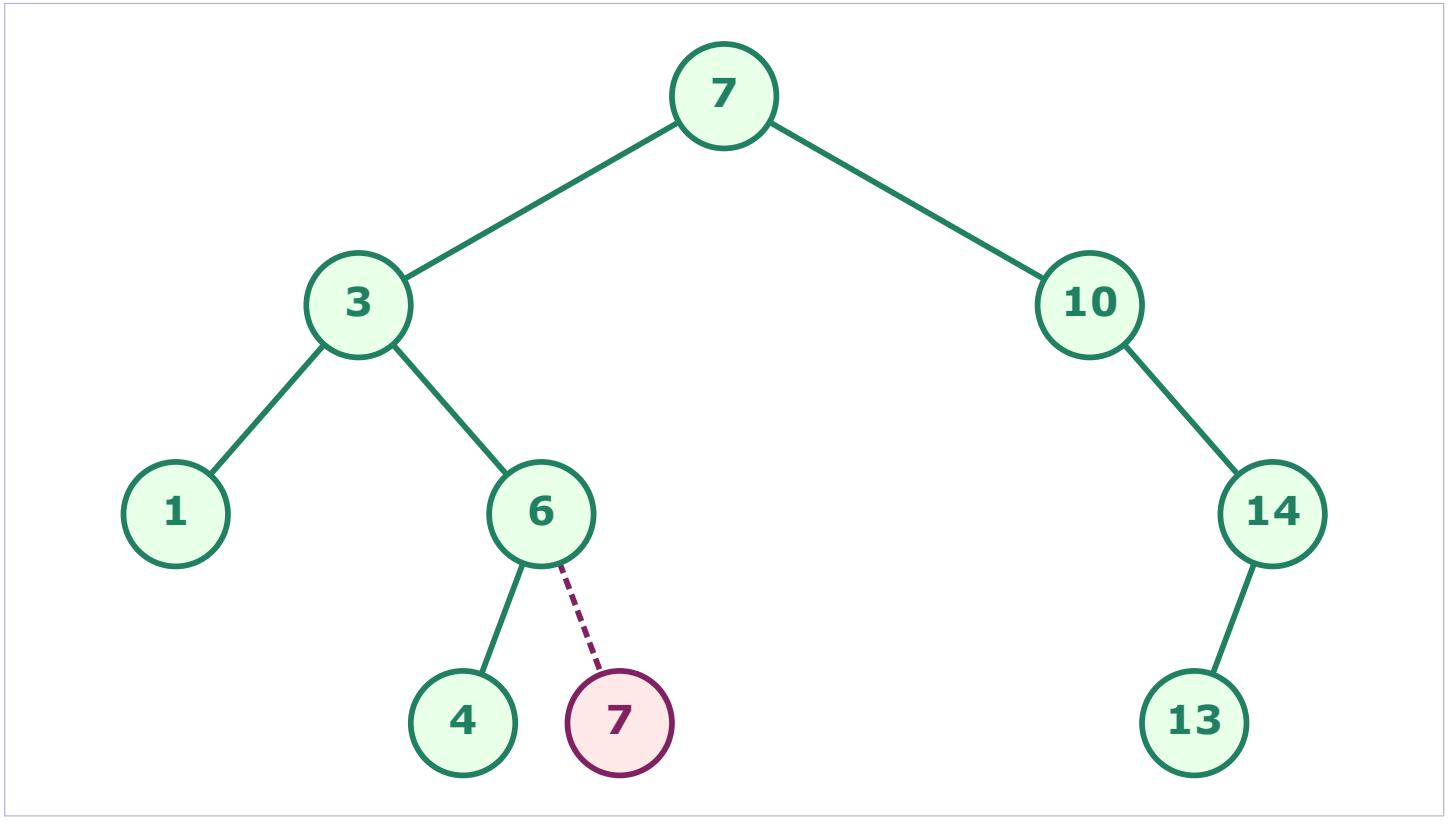
- **Substitua** o valor do nó a ser removido pelo valor **sucessor** ou **antecessor** deste nó.
- O valor sucessor está no **nó mais à esquerda** da **sub-árvore direita**, enquanto o valor antecessor fica no nó mais à **direita** da **sub-árvore esquerda**.
- **Remova**, então, o sucessor que, inevitavelmente, estará enquadrado em um dos casos anteriores:
  - nó **folha**; ou
  - nó com **um filho**.

Lógica para Exclusão de um Nó com Dois Fillhos



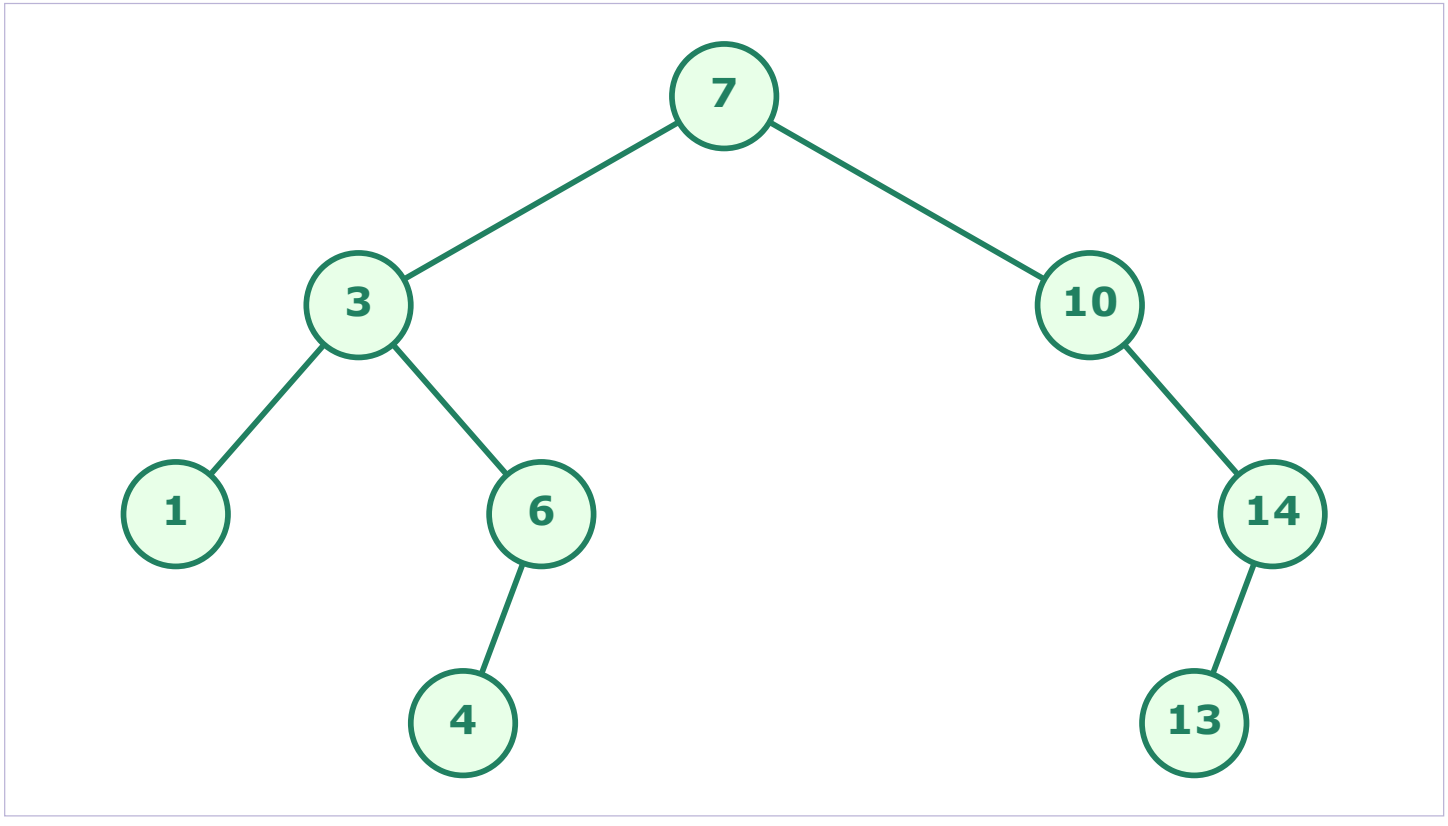
Excluindo o **8**.

Lógica para Exclusão de um Nó com Dois Fillhos



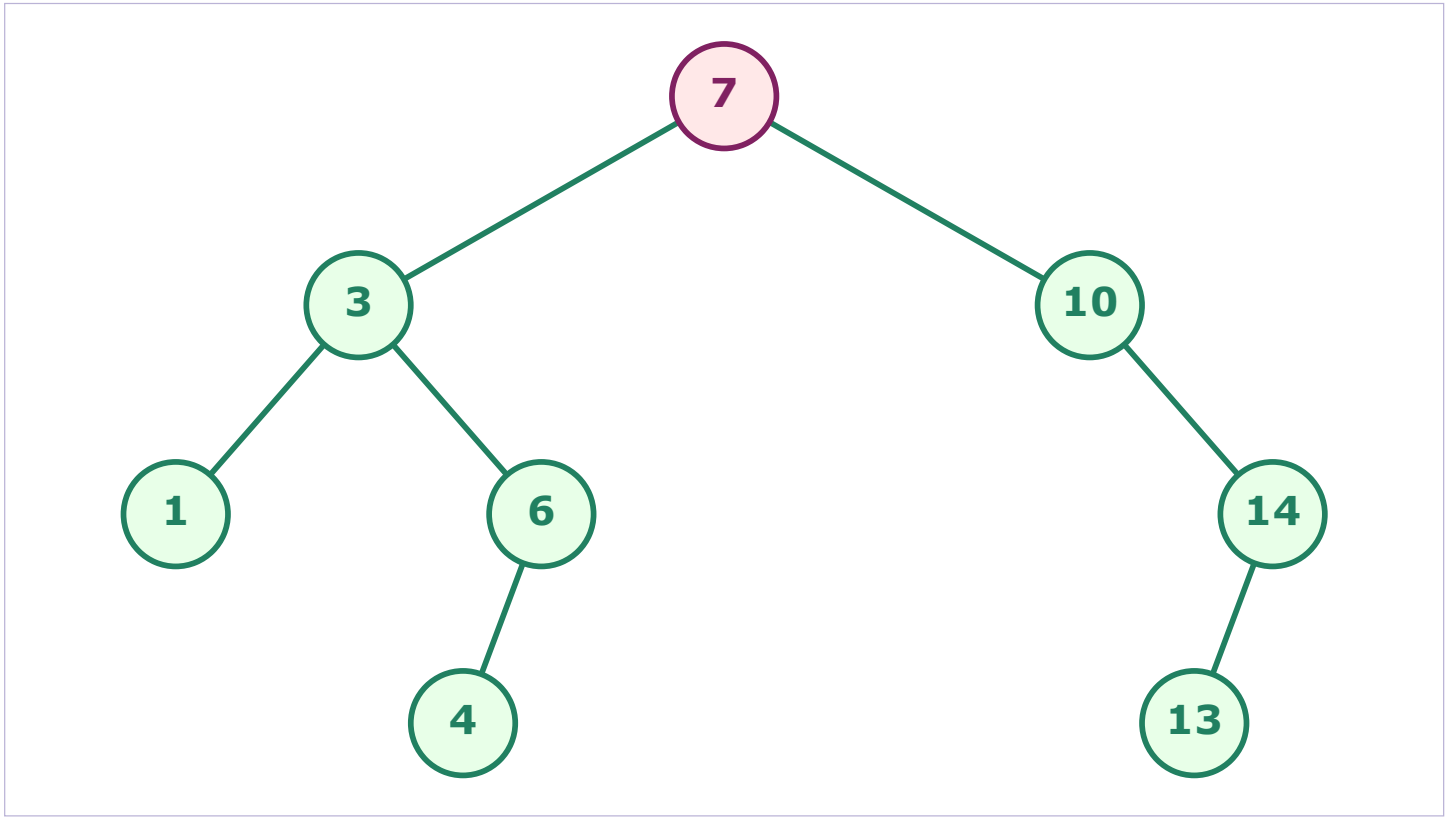
Excluindo o **8**: o antecessor é um nó folha.

Lógica para Exclusão de um Nó com Dois Fillhos



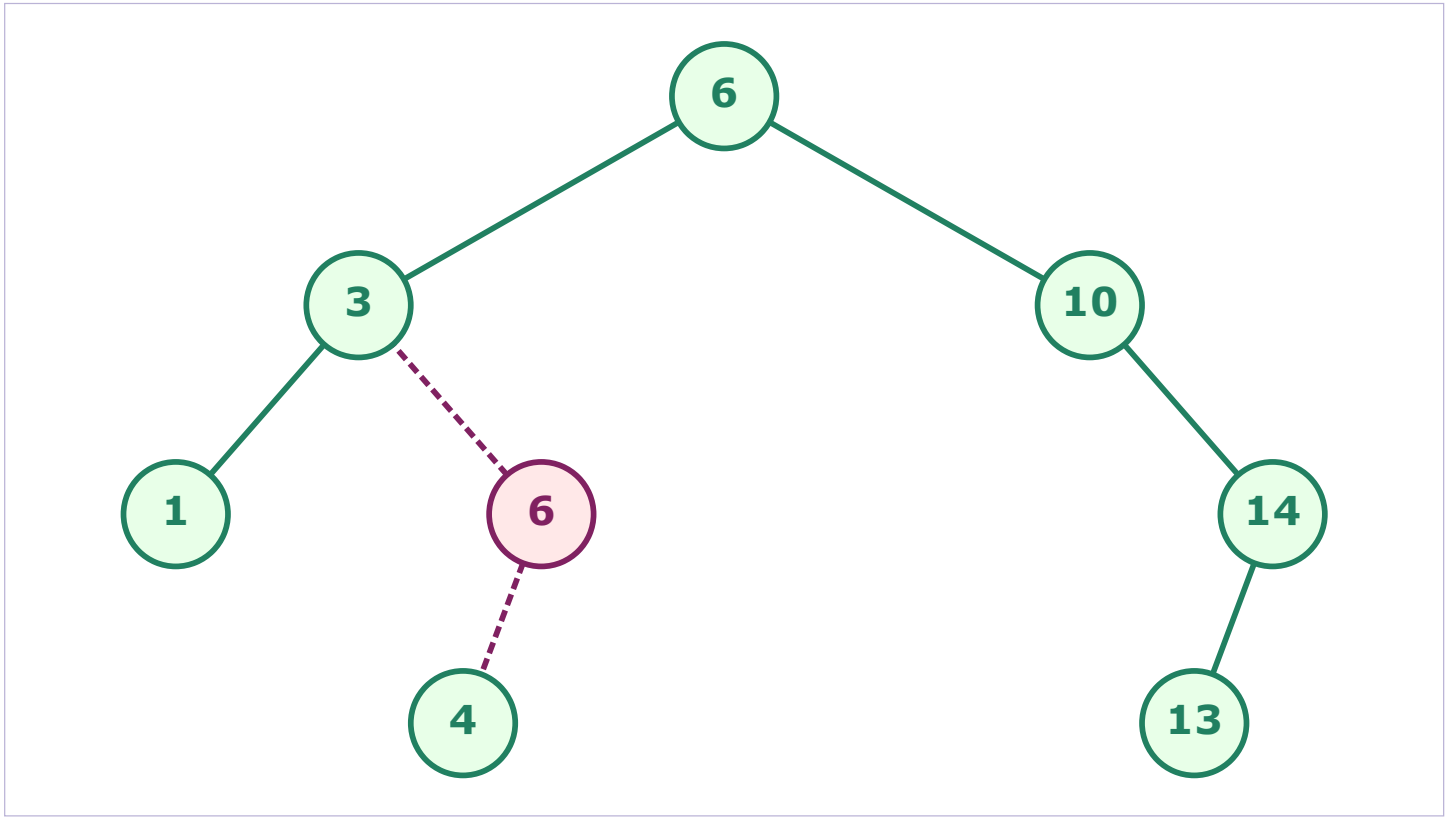
8 excluído.

Lógica para Exclusão de um Nó com Dois Fillhos



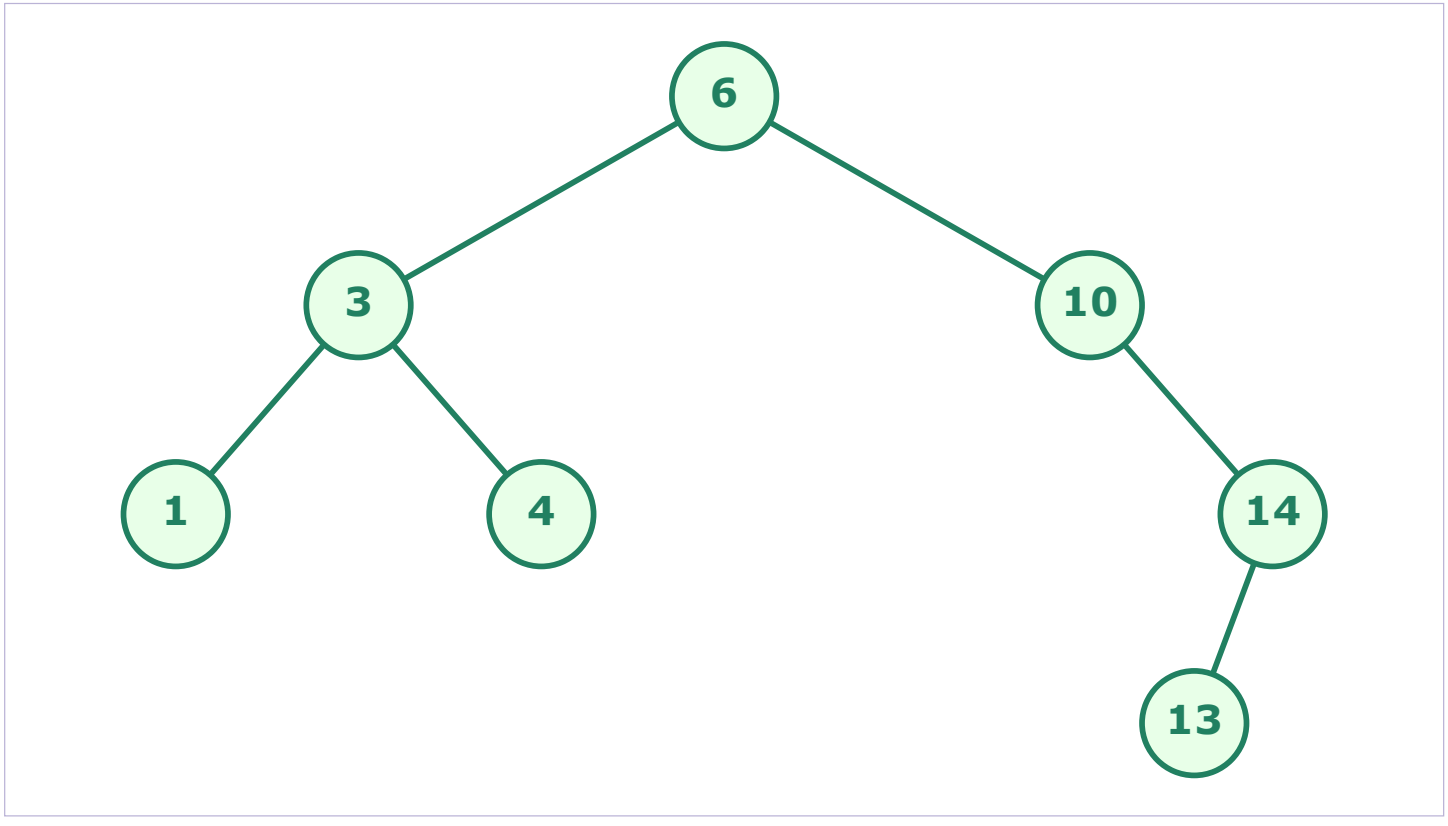
Excluindo o **7**.

Lógica para Exclusão de um Nó com Dois Fillhos



Excluindo o **7**: o antecessor é um nó com um único filho.

Lógica para Exclusão de um Nó com Dois Fillhos



7 excluído.