Teoria dos Grafos

Integrantes:

- **Guilherme Vizzoni Haidmann**
- **Rafael Christian Silva Wernesbach**
- **Rian Novelli Barcellos**
- **Samuel Barbosa Santos**
- **Tales Paiva Calvi**

Introdução

Objetivo

- Visualizar e compreender Árvores Binárias de Busca (ABB) e Árvores Genéricas em Python.

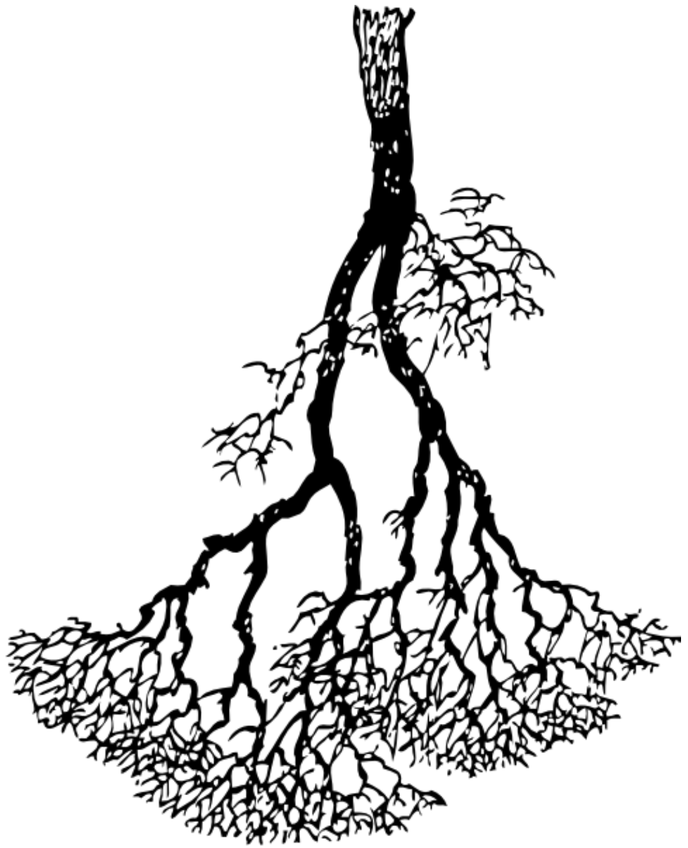
O que será abordado?

- Teoria e conceitos das árvores.
- Implementação em Python.
- Demonstração prática com interface gráfica.

Teoria das Estruturas

- **Teoria Geral Sobre árvores:**

Uma árvore é uma estrutura de dados hierárquica composta por nós conectados por arestas, onde há um nó especial chamado raiz e cada nó pode ter múltiplos filhos. Diferente de grafos, as árvores são estruturas sem ciclos e possuem apenas um caminho único entre dois nós quaisquer.



- **Árvores Binárias**

Uma árvore binária é um tipo especial de árvore onde cada nó pode ter no máximo dois filhos, chamados de filho esquerdo e filho direito. Essa restrição permite uma estrutura mais organizada e facilita operações como busca, inserção e remoção de elementos.

Teoria das Estruturas

- **Árvores Binárias De Busca:**

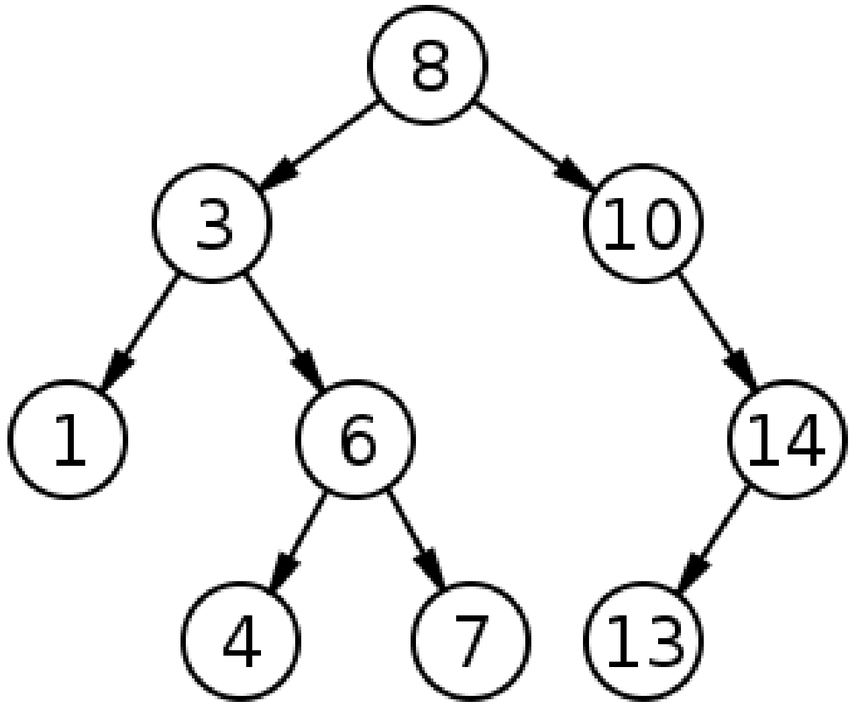
Uma Árvore Binária de Busca (ABB) é uma estrutura de dados que mantém seus elementos organizados de forma ordenada, facilitando operações de busca, inserção e remoção.

A ABB possui as seguintes propriedades:

Para cada nó da árvore:

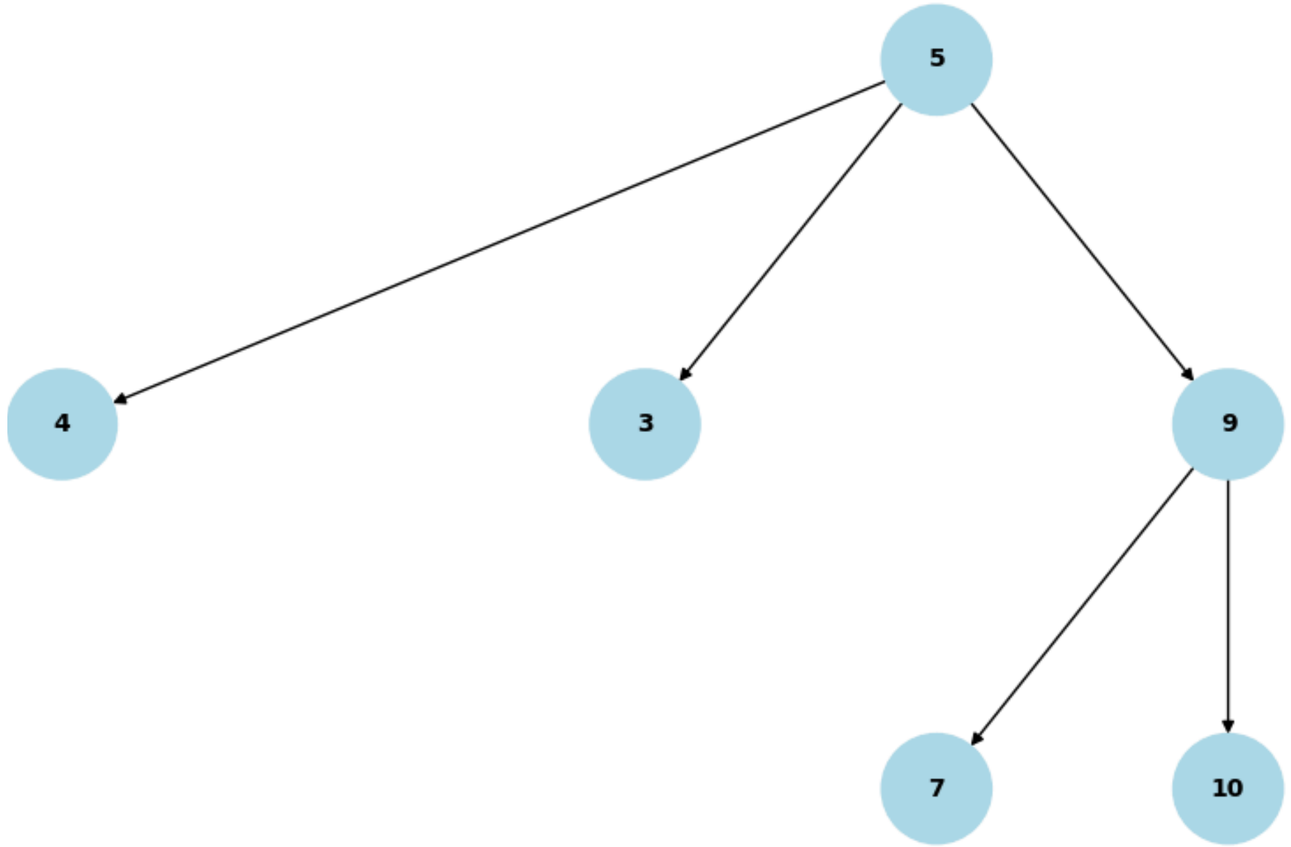
- Os valores menores ficam na subárvore esquerda.
- Os valores maiores ficam na subárvore direita.

Essa estrutura permite que a busca por um elemento seja mais eficiente do que em listas não ordenadas, com uma complexidade O(log n) em árvores balanceadas.



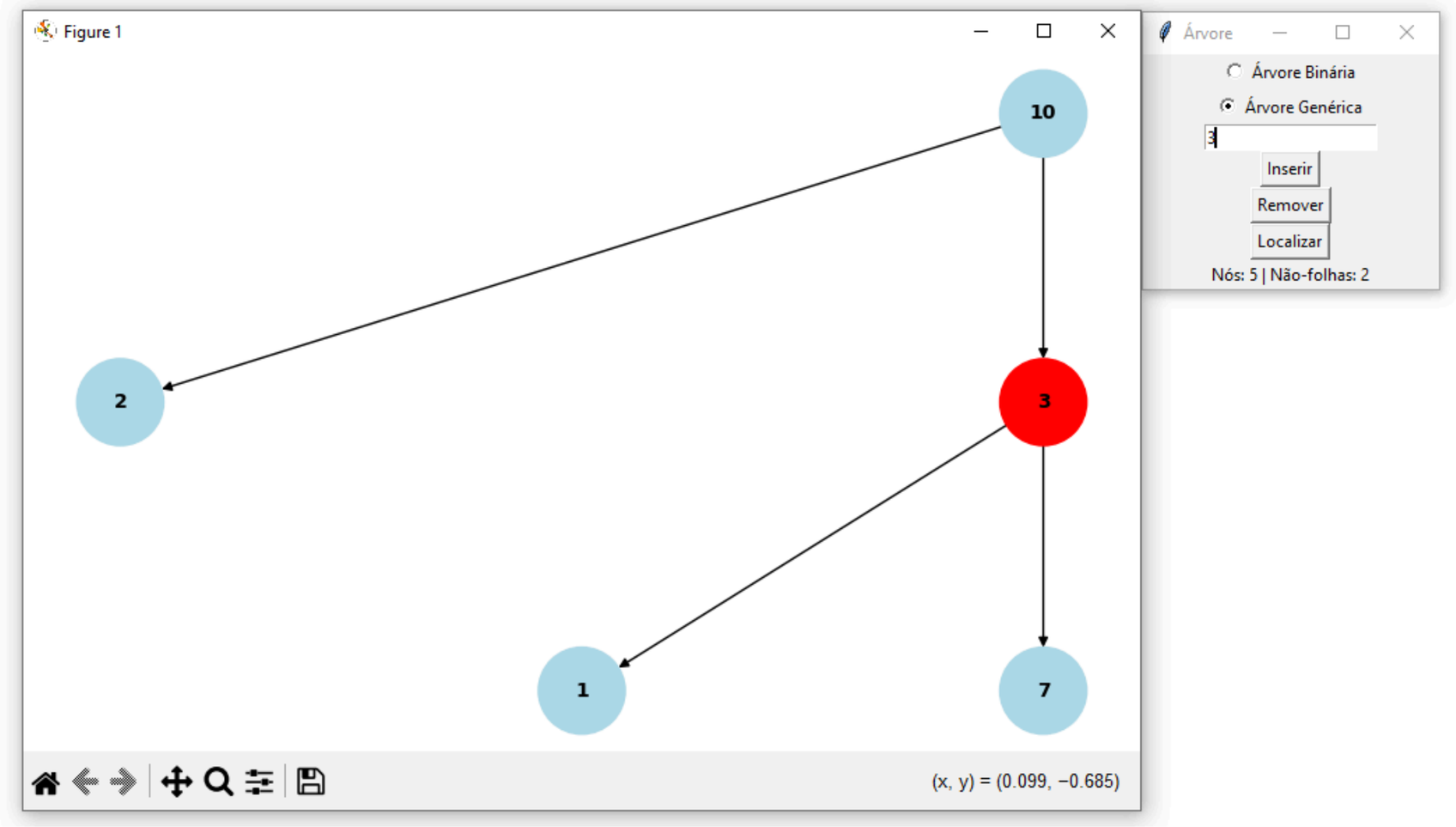
Exemplo Árvore Genérica

- **Adicionando Nós Filhos**



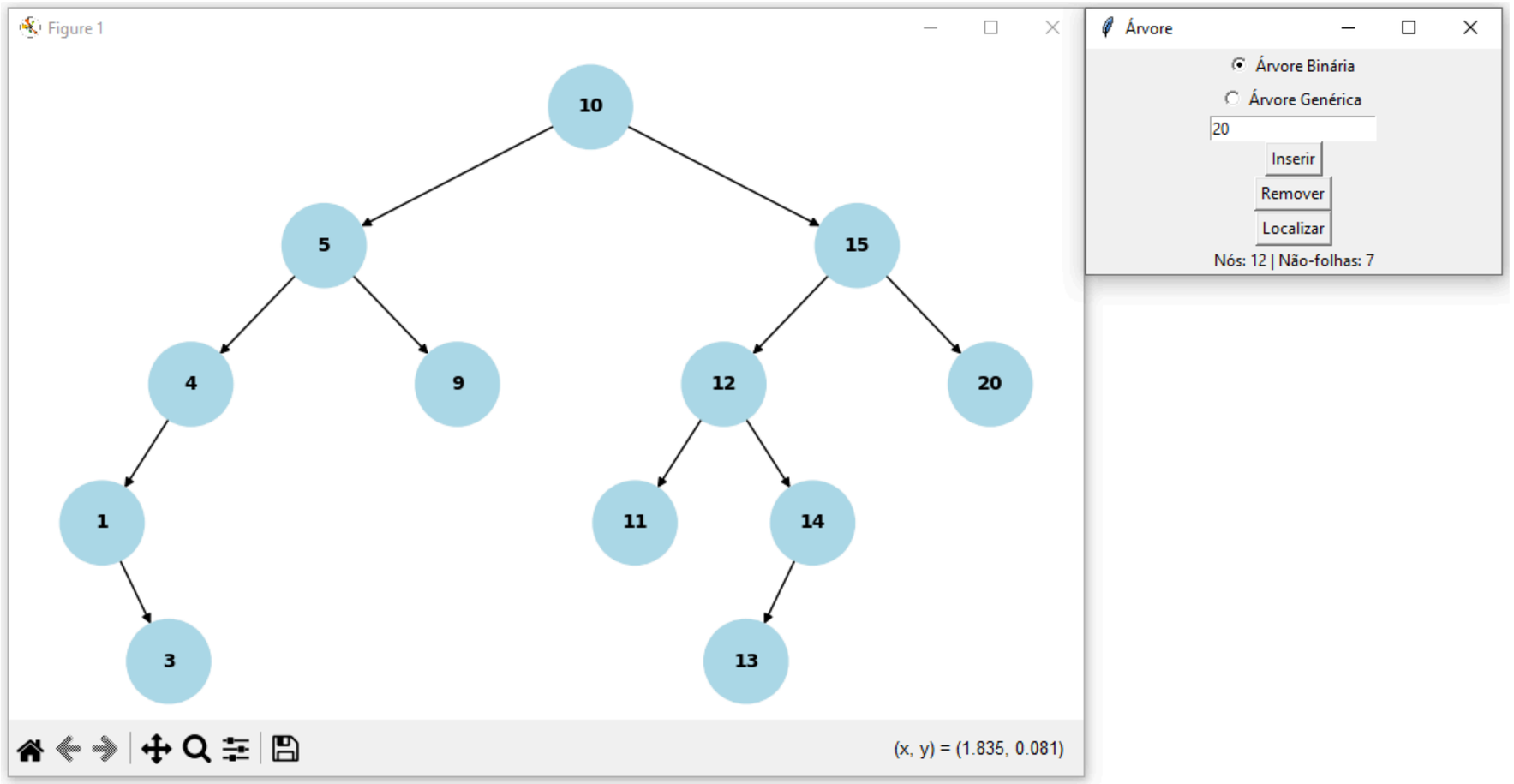
Exemplo Árvore Genérica

- **Encontrando o valor na Interface**



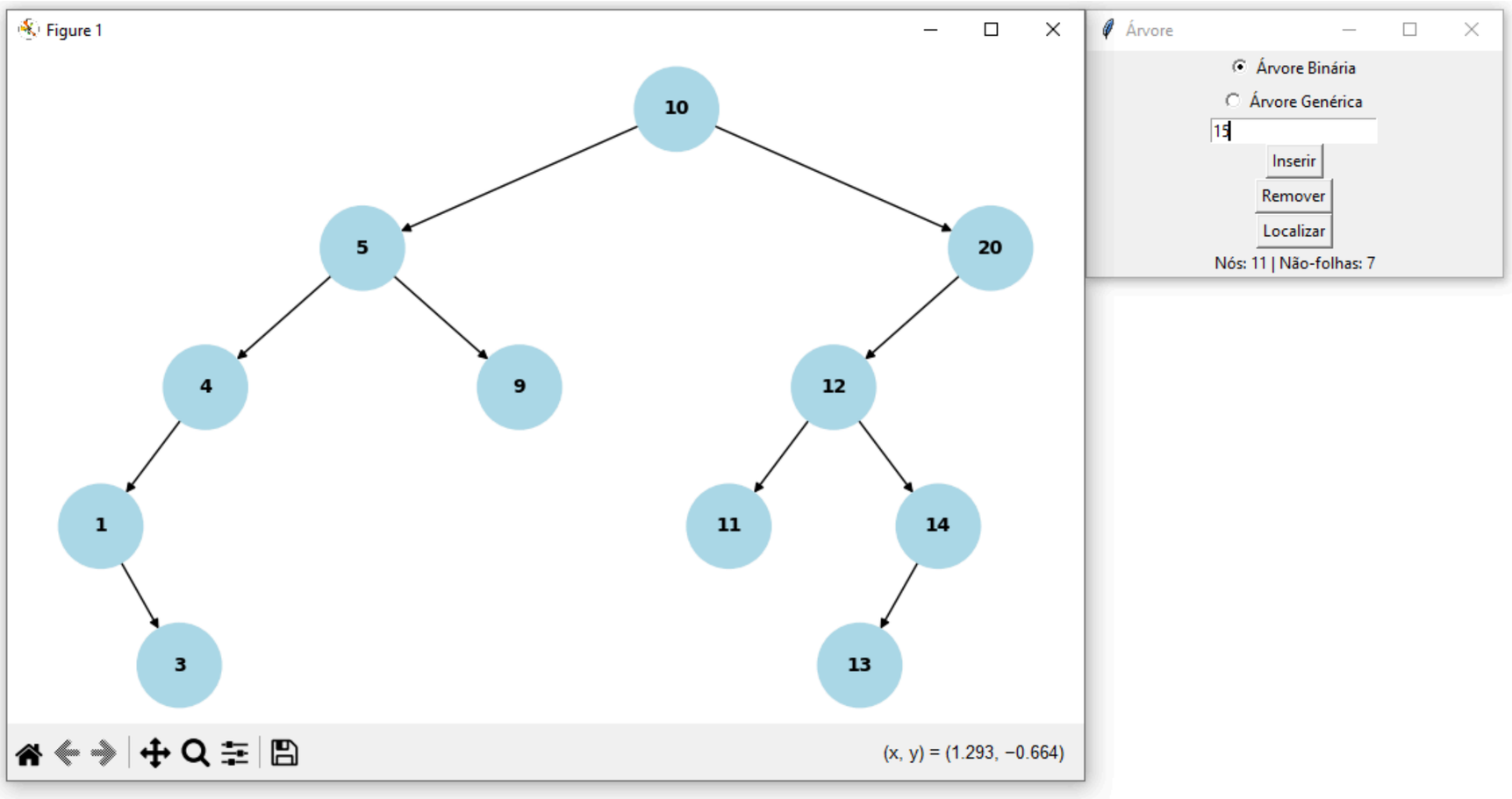
Árvore Binária de Busca

- **Inserindo um Valor na Interface**



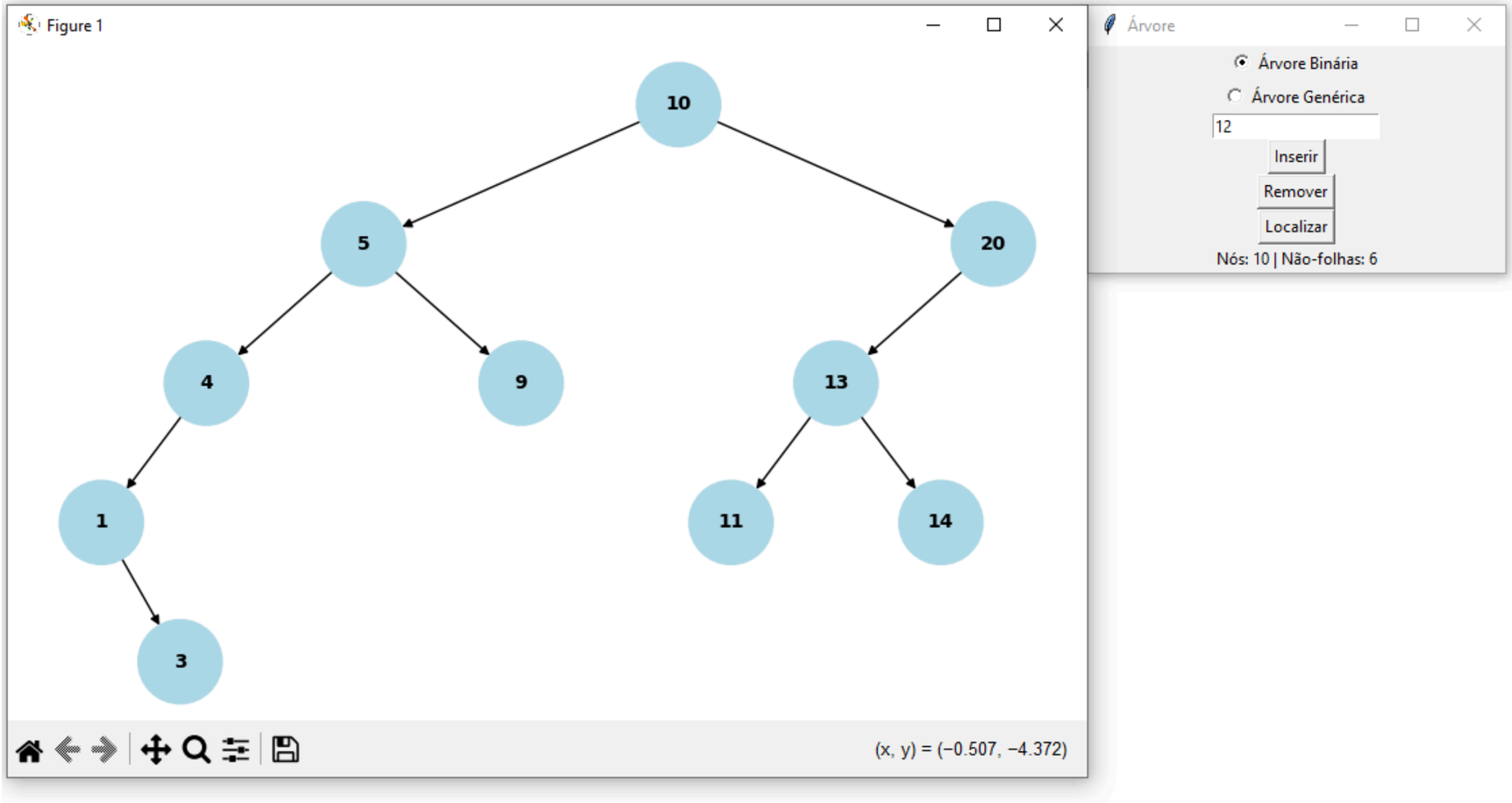
Árvore Binária de Busca

- **Removendo Valor na Interface**



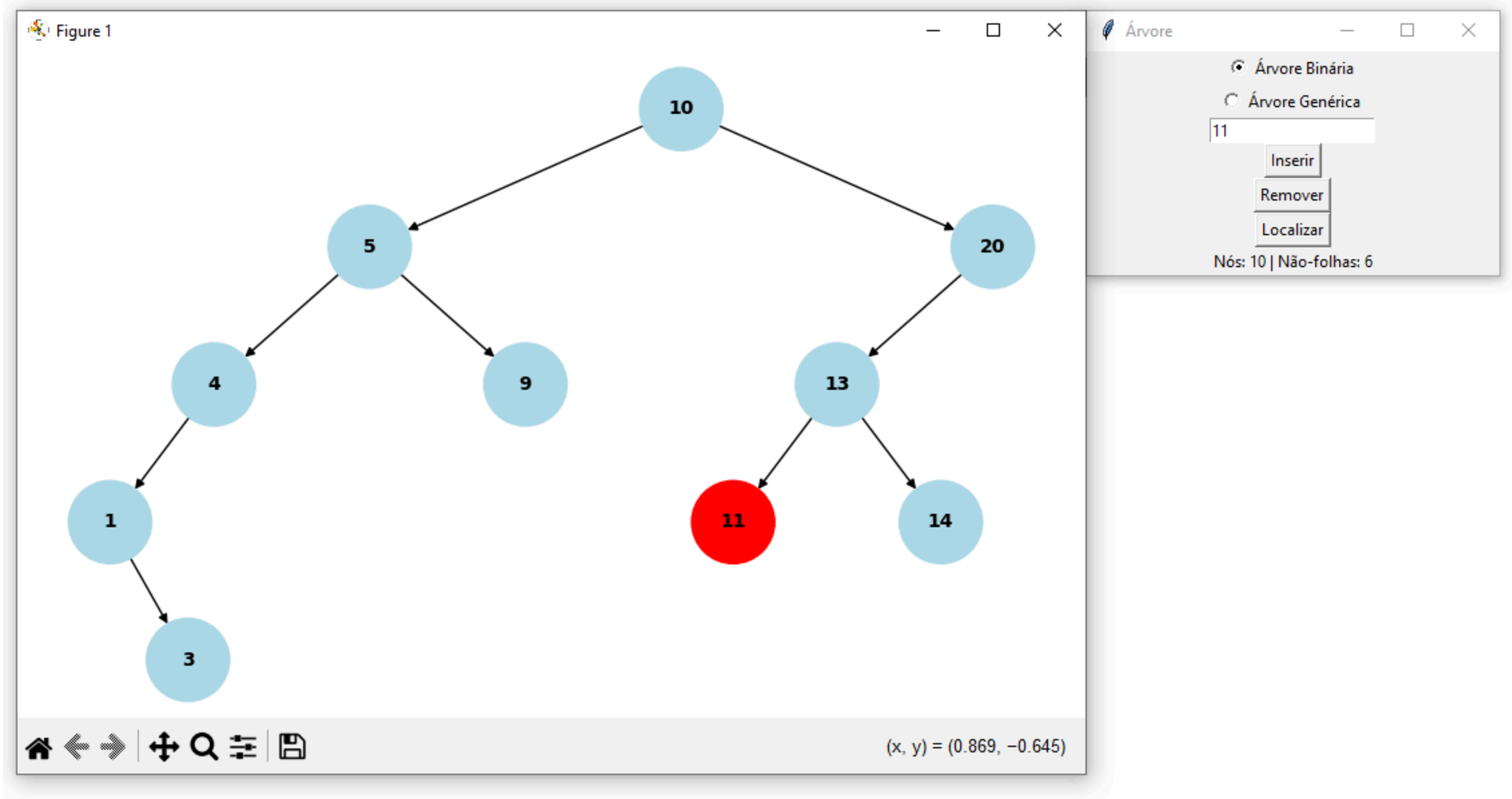
Árvore Binária de Busca

- **Removendo Valor na Interface**



Árvore Binária de Busca

- **Localizando na interface**



Código

A partir de agora abordaremos a implementação em python do algoritmo da ABB

```
    if self.tipo_arvore.get() == "binaria":
        total_nos = self.arvore_binaria.contar_nos()
        total_nao_folhas = self.arvore_binaria.contar_nao_folhas()
    else:
        total_nos = self.arvore_generica.contar_nos()
        total_nao_folhas = self.arvore_generica.contar_nao_folhas()
    self.contagem_label.config(text=f"Nós: {total_nos} | Não-folhas: {total_nao_folhas}"

def adicionar_valor(self):
    try:
        if self.tipo_arvore.get() == "binaria":
            valor = int(self.entrada.get())
            if self.arvore_binaria.buscar(valor) is not None:
                raise ValueError
            self.arvore_binaria.inserir(valor)
        else:
            valores = self.entrada.get().split(',')
            valor = int(valores[0])
            chave_pai = int(valores[1]) if len(valores) > 1 else None
            if chave_pai is not None and self.arvore_generica.buscar(self.arvore_generic
                raise ValueError
            self.arvore_generica.inserir(valor, chave_pai)
        self.atualizar_contagem()
        self.desenhar_arvore()
    except ValueError:
        messagebox.showerror("Erro", "Insira um valor válido no formato correto")

def remover_valor(self):
    try:
        valor = int(self.entrada.get())
        if self.tipo_arvore.get() == "binaria":
            self.arvore_binaria.excluir(valor)
        else:
```

Código

- **Classe No: Representa um nó da árvore binária. Cada nó contém:**
- **Uma chave (valor armazenado).**
- **Um ponteiro para a subárvore esquerda (self.esquerda).**
- **Um ponteiro para a subárvore direita (self.direita).**
- **Classe Arvore: Representa a árvore binária como um todo, possuindo um único atributo:**
- **raiz: Inicialmente None, indicando que a árvore começa vazia.**

```python
class No:
    def __init__(self, chave):
        self.chave = chave
        self.esquerda = None
        self.direita = None

class Arvore:
    def __init__(self):
        self.raiz = None
```

Código

As funções inserir(self, chave) e _inserir(self, raiz, chave) insere um novo nó na árvore, mantendo a propriedade da BST:

Se a chave for menor que a raiz.chave, ela é inserida na subárvore esquerda.

Se for maior, é inserida na subárvore direita.
O processo é recursivo até encontrar uma posição vazia (None).

A função _excluir(self, raiz, chave) remove um nó da árvore:
Busca o nó a ser removido recursivamente.

Caso base: Se o nó não for encontrado (raiz is None), retorna None.
Caso 1: Nó sem filhos → Retorna None, removendo o nó.
Caso 2: Nó com um filho → Retorna o único filho, substituindo o nó removido.
Caso 3: Nó com dois filhos → Substitui a chave pelo menor valor da subárvore direita, garantindo que a estrutura da BST seja mantida.

```python
def inserir(self, chave):
    if self.raiz is None:
        self.raiz = No(chave)
    else:
        self._inserir(self.raiz, chave)

def _inserir(self, raiz, chave):
    if chave < raiz.chave:
        if raiz.esquerda is None:
            raiz.esquerda = No(chave)
        else:
            self._inserir(raiz.esquerda, chave)
    else:
        if raiz.direita is None:
            raiz.direita = No(chave)
        else:
            self._inserir(raiz.direita, chave)

def excluir(self, chave):
    self.raiz = self._excluir(self.raiz, chave)

def _excluir(self, raiz, chave):
    if raiz is None:
        return raiz
    if chave < raiz.chave:
        raiz.esquerda = self._excluir(raiz.esquerda, chave)
    elif chave > raiz.chave:
        raiz.direita = self._excluir(raiz.direita, chave)
    else:
        if raiz.esquerda is None:
            return raiz.direita
        elif raiz.direita is None:
            return raiz.esquerda
        temp = self._menor_no(raiz.direita)
        raiz.chave = temp.chave
        raiz.direita = self._excluir(raiz.direita, temp.chave)
    return raiz
```

Código

A função _menor_no(self, raiz) encontra o nó com o menor valor da árvore:

Percorre a subárvore esquerda até encontrar o nó mais à esquerda.

Esse nó representa o menor valor da BST, pois, por definição, valores menores estão sempre na esquerda.

```
def _menor_no(self, raiz):
    atual = raiz
    while atual.esquerda is not None:
        atual = atual.esquerda
    return atual

def buscar(self, chave):
    return self._buscar(self.raiz, chave)

def _buscar(self, raiz, chave):
    if raiz is None or raiz.chave == chave:
        return raiz
    if chave < raiz.chave:
        return self._buscar(raiz.esquerda, chave)
    return self._buscar(raiz.direita, chave)
```

A função buscar(self, chave) procura um nó com a chave desejada na árvore.
Ela chama a função auxiliar recursiva _buscar(self, raiz, chave), que:

Caso base: **Retorna o nó se for encontrado ou None se a chave não existir na árvore.**
Se a chave for menor, a busca continua na subárvore esquerda.
Se a chave for maior, a busca continua na subárvore direita.

Código

```python
def contar_nos(self):
    return self._contar_nos(self.raiz)

def _contar_nos(self, raiz):
    if raiz is None:
        return 0
    return 1 + self._contar_nos(raiz.esquerda) + self._contar_nos(raiz.direita)

def contar_nao_folhas(self):
    return self._contar_nao_folhas(self.raiz)

def _contar_nao_folhas(self, raiz):
    if raiz is None or (raiz.esquerda is None and raiz.direita is None):
        return 0
    return 1 + self._contar_nao_folhas(raiz.esquerda) + self._contar_nao_folhas(raiz.direita)
```

A função contar_nos(self) chama o método auxiliar _contar_nos(self, raiz), passando a raiz da árvore como parâmetro.

No método auxiliar, o código verifica se a raiz é None, o que indica que a árvore (ou subárvore) está vazia.
Nesse caso, retorna 0.
Caso contrário, retorna 1 (representando o nó atual) somado com a quantidade de nós na subárvore esquerda e na subárvore direita, calculadas recursivamente

A função contar_nao_folhas(self) chama o método auxiliar _contar_nao_folhas(self, raiz) passando a raiz da árvore.

No método auxiliar, primeiro é verificado se o nó atual é None ou se é uma folha (ou seja, se não possui filhos à esquerda e à direita). Se for esse o caso, o retorno é 0, pois folhas não são contadas aqui
Se o nó não for uma folha, o código retorna 1 (contando o nó atual como não-folha) somado com os nós não-folhas da subárvore esquerda e da subárvore direita