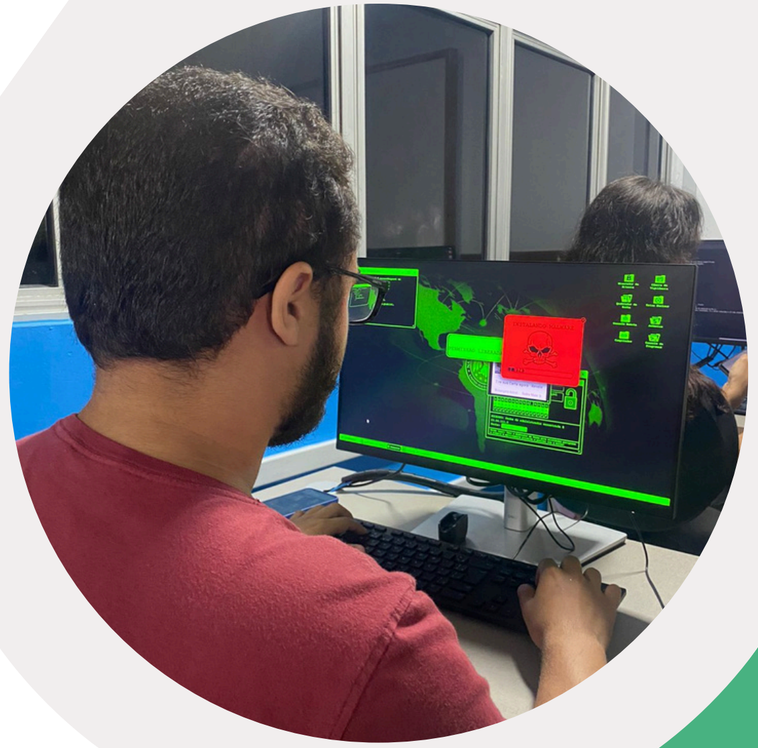
ÁRVORE BINÁRIA DE BUSCA (ABB)

Implementação em Python



Grupo: Lorenzo Simonassi, Robson Junior, Bruno Almonde, Pedro Cravo, Henrique Miranda.

Tópicos

- Introdução
- Estrutura da Árvore
- Operação - Inserção
- Operação - Exclusão
- Operação - Alteração
- Outras Funções
- Enfim o Código
- Código em Ação
- Conclusão

Introdução

O QUE É UMA ABB?

Estrutura de dados usada para organizar informações.

A principal característica de uma ABB é que, para cada nó, os valores dos nós na subárvore esquerda são menores que o valor do nó, e os valores dos nós na subárvore direita são maiores.

Tempo médio de execução: O(log n) quando balanceada.

OBJETIVO DO TRABALHO

Implementar uma ABB com operações de INSERÇÃO, REMOÇÃO, ALTERAR e EXIBIR.

Descartar Valores duplicados.

Função para calcular altura da árvore e mostrar no terminal.

Estrutura da Árvore

```python
class No:
    def __init__(self, key, dir=None, esq=None):
        self.item = key
        self.dir = dir
        self.esq = esq

class Tree:

    def __init__(self):
        self.root = None  # A raiz começa como None,
                          # representando uma árvore vazia.
```

Regra Básica	Menores à esquerda, maiores à direita.
Classe *Tree*	Gerencia a árvore inteira. Começa com raiz vazia (*root = None*).
Classe *No*	Nó da árvore. **Armazena**: valor (*item*), filho esquerdo (*esq*), filho direito (*dir*).

Operação - Inserção

MÉTODO *INSERIR*

Cria um novo nó com o valor informado.

Se árvore vazia, novo nó vira a raiz.
Senão, compara o no criado com os demais.
↳ Se menor, desce à esquerda até a folha.
↳ Se maior, desce à direita até a folha.

COMPLEXIDADE

O(h), onde h é a altura.

```python
def inserir(self, v):
    novo = No(v)  # Cria um novo nó
    if self.root is None:  # Se a árvore estiver vazia
        self.root = novo
    else:
        atual = self.root
        while True:
            if v < atual.item:  # Se o valor for menor, ir para a esquerda
                if atual.esq is None:
                    atual.esq = novo
                    return
                else:
                    atual = atual.esq  # Continua para a esquerda
            elif v > atual.item:  # Se for maior ou igual, ir para a direita
                if atual.dir is None:
                    atual.dir = novo
                    return
                else:
                    atual = atual.dir  # Continua para a direita
            else:
                return  # Não insere o valor se ele já existir
```

```python
def excluir(self, v):
    self.root = self._excluir(self.root, v)

def _excluir(self, raiz, v):
    # Caso base: se a árvore estiver vazia
    if raiz is None:
        return raiz

    # Se o valor a ser excluído for menor que o valor do nó atual
    if v < raiz.item:
        raiz.esq = self._excluir(raiz.esq, v)

    # Se o valor a ser excluído for maior que o valor do nó atual
    elif v > raiz.item:
        raiz.dir = self._excluir(raiz.dir, v)

    # Caso em que o valor a ser excluído é igual ao valor do nó atual
    else:
        # Caso 1: O nó não tem filhos (é uma folha)
        if raiz.esq is None and raiz.dir is None:
            return None

        # Caso 2: O nó tem um filho à esquerda
        elif raiz.dir is None:
            return raiz.esq

        # Caso 3: O nó tem um filho à direita
        elif raiz.esq is None:
            return raiz.dir

        # Caso 4: O nó tem dois filhos
        else:
            # Encontra o nó com o menor valor na subárvore direita
            minimo = self._minimo(raiz.dir)
            raiz.item = minimo  # Substitui o valor do nó a ser excluíd
            raiz.dir = self._excluir(raiz.dir, minimo)  # Exclui o nó m

    return raiz
```

Operação - Exclusão

MÉTODO *EXCLUIR*

1. **Nó é folha:** remove o nó direto (substitui por None).
2. **Tem um filho:** substitui pelo filho.
3. **Tem dois filhos:** usa o sucessor (menor da subárvore direita).

AUXILIAR *_MINIMO*

Encontra o menor valor descendo à esquerda.

COMPLEXIDADE

Semelhante ao de inserção O(h).

Operação - Alteração

MÉTODO *ALTERAR*

Remove o valor antigo (excluir).
Insere o novo valor (inserir).

COMPLEXIDADE

O(h) + O(h) = O(h).

```python
def alterar(self, v_antigo, v_novo):
    # Exclui o nó com o valor antigo
    self.excluir(v_antigo)
    # Insere o novo valor na posição correta
    self.inserir(v_novo)
```

Outras Funções

```
def altura(self, no):
    """Calcula a altura da árvore"""
    if no is None:
        return -1  # Altura de uma árvore vazia é -1
    else:
        altura_esq = self.altura(no.esq)
        altura_dir = self.altura(no.dir)
        return 1 + max(altura_esq, altura_dir)

def show_tree (self, no, level=0):
    if no:
        self.show_tree (no.dir, level + 1)  # Chama r
        print("      " * level + f"({no.item})")  # Ex
        self.show_tree (no.esq, level + 1)  # Chama r
```

Altura (*altura*)	Exibição (*show_tree*)
• Calcula a altura da árvore recursivamente. • Árvore vazia: -1. • Senão: 1 + maior altura entre subárvores.	• Mostra a árvore em formato visual. • Usa recursão para percorrer os nós.

Enfim o Código

```python
class No:
    def __init__(self, key, dir=None, esq=None):
        self.item = key
        self.dir = dir
        self.esq = esq

class Tree:

    def __init__(self):
        self.root = None  # A raiz começa como None, representando uma árvore vazia.

    def inserir(self, v):
        novo = No(v)  # Cria um novo nó
        if self.root is None:  # Se a árvore estiver vazia
            self.root = novo
        else:
            atual = self.root
            while True:
                if v < atual.item:  # Se o valor for menor, ir para a esquerda
                    if atual.esq is None:
                        atual.esq = novo
                        return
                    else:
                        atual = atual.esq  # Continua para a esquerda
                elif v > atual.item:  # Se for maior ou igual, ir para a direita
                    if atual.dir is None:
                        atual.dir = novo
                        return
                    else:
                        atual = atual.dir  # Continua para a direita
                else:
                    return  # Não insere o valor se ele já existir
```

```python
def excluir(self, v):
    self.root = self._excluir(self.root, v)

def _excluir(self, raiz, v):
    # Caso base: se a árvore estiver vazia
    if raiz is None:
        return raiz

    # Se o valor a ser excluído for menor que o valor do nó atual
    if v < raiz.item:
        raiz.esq = self._excluir(raiz.esq, v)

    # Se o valor a ser excluído for maior que o valor do nó atual
    elif v > raiz.item:
        raiz.dir = self._excluir(raiz.dir, v)

    # Caso em que o valor a ser excluído é igual ao valor do nó atual
    else:
        # Caso 1: O nó não tem filhos (é uma folha)
        if raiz.esq is None and raiz.dir is None:
            return None

        # Caso 2: O nó tem um filho à esquerda
        elif raiz.dir is None:
            return raiz.esq

        # Caso 3: O nó tem um filho à direita
        elif raiz.esq is None:
            return raiz.dir
```

Enfim o Código

```
        # Caso 4: O nó tem dois filhos
        else:
            # Encontra o nó com o menor valor na subárvore direita
            minimo = self._minimo(raiz.dir)
            raiz.item = minimo  # Substitui o valor do nó a ser excluído pelo mínimo da subárvore direita
            raiz.dir = self._excluir(raiz.dir, minimo)  # Exclui o nó mínimo da subárvore direita

    return raiz

def _minimo(self, raiz):
    atual = raiz
    while atual.esq is not None:
        atual = atual.esq
    return atual.item

def alterar(self, v_antigo, v_novo):
    # Exclui o nó com o valor antigo
    self.excluir(v_antigo)
    # Insere o novo valor na posição correta
    self.inserir(v_novo)
```

```
def altura(self, no):
    """Calcula a altura da árvore"""
    if no is None:
        return -1  # Altura de uma árvore vazia é -1
    else:
        altura_esq = self.altura(no.esq)
        altura_dir = self.altura(no.dir)
        return 1 + max(altura_esq, altura_dir)

def show_tree(self, no, level=0):
    if no:
        self.show_tree(no.dir, level + 1)  # Chama recursivamente a direita
        print("      " * level + f"({no.item})")  # Exibe o item do nó
        self.show_tree(no.esq, level + 1)  # Chama recursivamente a esquerda
```

```
Oque você deseja?
 1: Inserir
 2: Exibir
 3: Excluir
 4: Alterar
 5: Sair do programa
-> 2
     (15)
(10)
     (5)
A altura da árvore é: 1
```

```
 Oque você deseja?
  1: Inserir
  2: Exibir
  3: Excluir
  4: Alterar
  5: Sair do programa
 -> 2
       (15)
 (10)
 A altura da árvore é: 1
```

Código em Ação

Exemplo de Uso:

- Inserir: 10, 5, 15
- Exibir Árvore.
- Excluir: 5.

>Fim<

Referências

https://pythonhelp.wordpress.com/2015/01/19/arvore-binaria-de-busca-em-python/

https://gist.github.com/divanibarbosa/a8662693e44ab9ee0d0e8c2d74808929

https://www.freecodecamp.org/portuguese/news/arvores-binarias-de-busca-bst-explicada-com-exemplos/