

# **LAPORAN PRAKTIKUM STRUKTUR DATA**

**MODUL KE-07**

**JUDUL PRAKTIKUM**



**Disusun Oleh:**

**Nama** : Restu Wibisono  
**NPM** : 2340506061  
**Kelas** : 03 (Tiga)

**Program Studi S1 Teknologi Informasi**

**Fakultas Teknik, Universitas Tidar**

**Genap 2023/2024**

## I. Tujuan Praktikum

Adapun tujuan praktikum ini sebagai berikut :

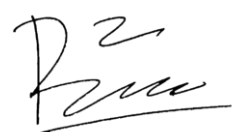
1. Mahasiswa mampu menerapkan konsep tree pada bahasa pemrograman python

## II. Dasar Teori

Struktur data tree adalah sebuah struktur hierarkis yang digunakan untuk mewakili dan mengatur data dengan cara yang mudah untuk dinavigasi dan dicari. Ini merupakan kumpulan dari node yang saling terhubung melalui tepi dan memiliki hubungan hierarkis di antara node-node tersebut. Terminologi dalam Struktur Data Tree:

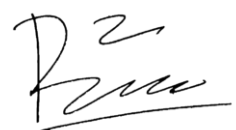
- Node Induk: Node yang menjadi pendahulu suatu node disebut node induk dari node tersebut. {B} adalah node induk dari {D, E}.
- Node Anak: Node yang merupakan pengganti langsung dari suatu node disebut node anak dari node tersebut. Contoh: {D, E} adalah node anak dari {B}.
- Node Akar: Node paling atas dari sebuah tree atau node yang tidak memiliki node induk disebut node akar. {A} adalah node akar dari tree tersebut. Sebuah tree yang tidak kosong harus mengandung tepat satu node akar dan tepat satu jalur dari akar ke semua node lain dalam tree.
- Node Daun atau Node Eksternal: Node yang tidak memiliki node anak disebut node daun. {K, L, M, N, O, P, G} adalah node daun dari tree tersebut.
- Silsilah dari Sebuah Node: Node-node pendahulu di jalur dari akar ke node tersebut disebut silsilah dari node tersebut. {A, B} adalah node silsilah dari node {E}.
- Keturunan: Sebuah node x adalah keturunan dari node lain y jika dan hanya jika y adalah leluhur dari x.
- Saudara: Anak-anak dari node induk yang sama disebut saudara. {D, E} disebut saudara.

Tanda Tangan



- Tingkat sebuah node: Jumlah tepi pada jalur dari node akar ke node tersebut. Node akar memiliki tingkat 0.
- Node Internal: Sebuah node dengan setidaknya satu node anak disebut Node Internal.
- Tetangga dari Sebuah Node: Node induk atau anak dari node tersebut disebut tetangga dari node tersebut.
- Subtree: Setiap node dari tree bersama dengan keturunannya.

Tanda Tangan

A handwritten signature in black ink, consisting of a stylized 'D' followed by a series of loops and a horizontal stroke at the bottom.

### III. Hasil dan Pembahasan

#### 1. Implementasi Sederhana dari Binary Tree

```
class Node:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.val = key

if __name__ == '__main__':
    root = Node(1)
    root.left = Node(2)
    root.right = Node(3)
    root.left.left = Node(4)
    root.left.right = Node(5)
```

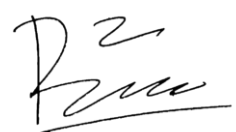
Python

1  
2  
3  
4  
5

(Gambar 3.1.1)

1. Membuat kelas untuk Node (simpul) dalam tree biner. Yang akan berfungsi menjadi cetakan untuk membuat objek simpul dalam tree.
2. metode konstruktor dengan `def __init__(self, key):` untuk kelas 'Node'. Saat objek Node dibuat, ini akan dipanggil yang berfungsi menginisialisasi objek. Dengan menerima parameter 'key', yang merupakan nilai yang akan disimpan di dalam simpul.
3. 'self.left' berfungsi untuk menyimpan referensi ke simpul anak kiri. Pembuatan 'Node', atribut ini diatur ke 'None', yang menunjukkan simpul tersebut belum memiliki anak sampul.
4. Sama seperti sebelumnya, 'self.right' berfungsi untuk menyimpan referensi ke simpul anak kanan. Pada pembuatan objek 'Node', atribut ini diatur ke 'None', untuk menunjukkan simpul tersebut awalnya tidak memiliki anak kanan.
5. `self.val = key` berfungsi menyimpan nilai dari simpul itu sendiri. Nilai ini diambil dari parameter 'key' yang diterima oleh konstruktor.
6. Idiom `if __name__ == '__main__':` Python yang digunakan untuk beberapa kode, bukan jika diimpor sebagai modul.

Tanda Tangan



7. Pada `root = Node(1"-5")` berfungsi membuat objek 'Node' dengan nilai kunci 1 dan menyimpannya ke dalam variabel 'root', yang akan menjadi akar dari tree biner. Penyisipan Elemen

```
class Node:
    def __init__(self, data):
        self.left = None
        self.right = None
        self.val = data

    def insert(self, data):
        if self.val:
            if data < self.val:
                if self.left is None:
                    self.left = Node(data)
                else:
                    self.left.insert(data)
            elif data > self.val:
                if self.right is None:
                    self.right = Node(data)
                else:
                    self.right.insert(data)
            else:
                self.val = data

    def PrintTree(self):
        if self.left:
            self.left.PrintTree()
        print(self.val),
        if self.right:
            self.right.PrintTree()

root = Node(12)
root.insert(6)
root.insert(14)
root.insert(3)
root.PrintTree()
```

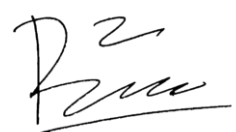
Python

3  
6  
12  
14

(Gambar 3.1.2)

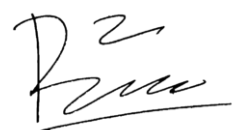
1. Untuk mendefinisikan kelas Node digunakan 'class Node' merepresentasikan simpul dalam BST.
2. Konstruktor menggunakan `def __init__(self, data):`. Ketika Node dibuat, metode akan dipanggil untuk menginisialisasi. Ini akan menerima parameter 'data', yang merupakan nilai yang akan disimpan dalam simpul.
3. 'self.left = None' dan 'self.right = None': adalah atribut 'left' dan 'right' yang digunakan untuk menyimpan referensi dari simpul saat ini. Jika simpul baru di buat, kedua atribut di atur menjadi 'None' .

Tanda Tangan



4. Atribut 'val' berfungsi untuk menyimpan nilai dari simpul itu sendiri.
  5. Metode 'insert' berfungsi untuk menyisipkan elemen baru. Ini menerima parameter 'data', yang mana merupakan nilai dari elemen yang akan di sisipkan.
  6. 'if data < self.val:' dan 'if data > self.val:': berfungsi untuk membandingkan nilai 'data' dengan nilai 'self.val'.
  7. 'if self.left is None:' dan 'if self.right is None:': berfungsi untuk memeriksa anak kiri atau anak kanan dari simpul saat ini kosong. Jika kosong maka elemen baru akan.
  8. 'self.left.insert(data)' dan 'self.right.insert(data)': Jika anak kiri atau anak kanan tidak kosong, metode 'insert' akan dipanggil rekursif.
  9. 'def PrintTree(self):' Metode 'PrintTree' digunakan untuk mencetak nilai-nilai dalam BST secara inorder traversal.
  10. 'print(self.val)' berfungsi untuk mencetak nilai dari simpul saat ini.
2. Lintasan Elemen In-order Traversal

Tanda Tangan

A handwritten signature in black ink, consisting of a stylized 'P' followed by a series of loops and a horizontal stroke at the bottom.

```

class Node:
    def __init__(self, data):
        self.left = None
        self.right = None
        self.val = data

    def insert(self, data):
        if self.val:
            if data < self.val:
                if self.left is None:
                    self.left = Node(data)
                else:
                    self.left.insert(data)
            elif data > self.val:
                if self.right is None:
                    self.right = Node(data)
                else:
                    self.right.insert(data)
            else:
                self.val = data

    def PrintTree(self):
        if self.left:
            self.left.PrintTree()
        print(self.val),
        if self.right:
            self.right.PrintTree()

    def inorderTraversal(self, root):
        res = []
        if root:
            res = self.inorderTraversal(root.left)
            res.append(root.val)
            res = res + self.inorderTraversal(root.right)
        return res

root = Node(27)
root.insert(14)
root.insert(35)
root.insert(10)
root.insert(19)
root.insert(31)
root.insert(42)
print(root.inorderTraversal(root))

```

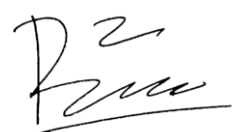
Python

[10, 14, 19, 27, 31, 35, 42]

(Gambar 3.2.1)

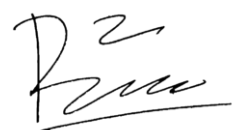
1. Untuk mendefinisikan kelas Node digunakan 'class Node:' merepresentasikan simpul dalam tree biner.
2. Konstruktor menggunakan def \_\_init\_\_(self, data):. Ketika Node dibuat, metode akan dipanggil untuk menginisialisasi. Ini akan menerima parameter 'data', yang merupakan nilai yang akan disimpan dalam simpul.
3. 'self.left = None' dan 'self.right = None': adalah atribut 'left' dan 'right' yang digunakan untuk menyimpan referensi dari simpul saat ini. Jika simpul baru di buat, kedua atribut di atur menjadi 'None' .

Tanda Tangan



4. Atribut 'val' berfungsi untuk menyimpan nilai dari simpul itu sendiri.
  5. Metode 'insert' berfungsi untuk menyisipkan elemen baru. Ini menerima parameter 'data', yang mana merupakan nilai dari elemen yang akan di sisipkan.
  6. Metode 'PrintTree' digunakan untuk mencetak nilai dalam tree biner secara inorder.
  7. Pada 'inorderTraversal' Berfungsi untuk melakukan penelusuran dari tree biner yang akan mengembalikan daftar nilai-nilai simpul urutan inorder.
  8. Variabel 'res' digunakan untuk menyimpan hasil penelusuran inorder.
  9. Selanjutnya memeriksa apakah 'root' bukan 'None'. Jika tidak, simpul akan melakukan penelusuran.
  10. Lalu melakukan rekursi yang berfungsi menelusuri anak kiri dari 'root' dan menyimpan hasilnya dalam 'res'.
  11. Menambahkan nilai 'root' ke dalam 'res' yang dilakukan setelah menelusuri anak kiri, simpul yang diproses terakhir adalah simpul yang paling kiri.
  12. 'return res' akan mengembalikan daftar yang berisi nilai dalam inorder.
  13. 'root = Node(27)' Membuat objek 'root' dengan nilai 27.
  14. Menyisipkan beberapa nilai ke dalam tree biner menggunakan metode 'insert'.
  15. Mencetak hasil penelusuran inorder dengan 'print(root.inorderTraversal(root))' dari tree biner dengan menggunakan metode 'inorderTraversal'.
3. Lintasan Elemen Pre-order Traversal

Tanda Tangan





```

class Node:
    def __init__(self, data):
        self.left = None
        self.right = None
        self.val = data

    def insert(self, data):
        if self.val:
            if data < self.val:
                if self.left is None:
                    self.left = Node(data)
                else:
                    self.left.insert(data)
            elif data > self.val:
                if self.right is None:
                    self.right = Node(data)
                else:
                    self.right.insert(data)
            else:
                self.val = data

    def PrintTree(self):
        if self.left:
            self.left.PrintTree()
        print(self.val),
        if self.right:
            self.right.PrintTree()

    def preorderTraversal(self, root):
        res = []
        if root:
            res.append(root.val)
            res = res + self.preorderTraversal(root.left)
            res = res + self.preorderTraversal(root.right)
        return res

root = Node(27)
root.insert(14)
root.insert(35)
root.insert(10)
root.insert(19)
root.insert(31)
root.insert(42)
print(root.preorderTraversal(root))

```

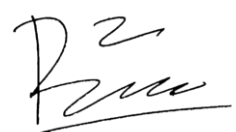
Python

[27, 14, 10, 19, 35, 31, 42]

(Gambar 3.3.1)

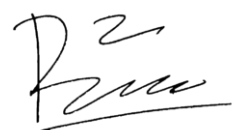
1. 'class Node:': Mendefinisikan kelas Node untuk merepresentasikan simpul dalam tree biner.
2. metode konstruktor dengan def \_\_init\_\_(self, key): untuk kelas 'Node'. Saat objek Node dibuat, ini akan dipanggil yang berfungsi menginisialisasi objek. Dengan menerima parameter 'key', yang merupakan nilai yang akan disimpan di dalam simpul.
3. 'self.left' berfungsi untuk menyimpan referensi ke simpul anak kiri. Pembuatan 'Node', atribut ini diatur ke 'None', yang menunjukkan simpul tersebut belum memiliki anak sampul.

Tanda Tangan



4. self.val = key berfungsi menyimpan nilai dari simpul itu sendiri. Nilai ini diambil dari parameter 'key' yang diterima oleh konstruktor.
  5. Metode 'insert' berfungsi untuk menyisipkan elemen baru. Ini menerima parameter 'data', yang mana merupakan nilai dari elemen yang akan di sisipkan.
  6. Metode 'PrintTree' digunakan untuk mencetak nilai dalam tree biner secara inorder.
  7. Pada 'inorderTraversal' Berfungsi untuk melakukan penelusuran dari tree biner yang akan mengembalikan daftar nilai-nilai simpul urutan inorder.
  8. Variabel 'res' digunakan untuk menyimpan hasil penelusuran inorder.
  9. Selanjutnya memeriksa apakah 'root' bukan 'None'. Jika tidak, simpul akan melakukan penelusuran.
  10. Lalu melakukan rekursi yang berfungsi menelusuri anak kiri dari 'root' dan menyimpan hasilnya dalam 'res'.
  11. 'return res' akan mengembalikan daftar yang berisi nilai dalam inorder.
  12. 'root = Node(27)' Membuat objek 'root' dengan nilai 27.
  13. Menyisipkan beberapa nilai ke dalam tree biner menggunakan metode 'insert'.
  14. Mencetak hasil penelusuran inorder dengan 'print(root.inorderTraversal(root))' dari tree biner dengan menggunakan metode 'inorderTraversal'.
4. Lintasan Elemen Post-order Traversal

Tanda Tangan



```
class Node:
    def __init__(self, data):
        self.left = None
        self.right = None
        self.val = data

    def insert(self, data):
        if self.val:
            if data < self.val:
                if self.left is None:
                    self.left = Node(data)
                else:
                    self.left.insert(data)
            elif data > self.val:
                if self.right is None:
                    self.right = Node(data)
                else:
                    self.right.insert(data)
            else:
                self.val = data

    def PrintTree(self):
        if self.left:
            self.left.PrintTree()
        print(self.val),
        if self.right:
            self.right.PrintTree()

    def postorderTraversal(self, root):
        res = []
        if root:
            res = self.postorderTraversal(root.left)
            res = res + self.postorderTraversal(root.right)
            res.append(root.val)
        return res

root = Node(27)
root.insert(14)
root.insert(35)
root.insert(10)
root.insert(19)
root.insert(31)
root.insert(42)
print(root.postorderTraversal(root))
```

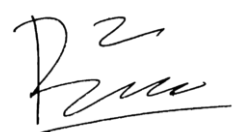
Python

[10, 19, 14, 31, 42, 35, 27]

(Gambar 3.4.1)

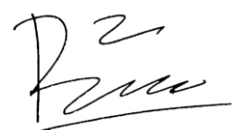
1. Untuk mendefinisikan kelas Node digunakan 'class Node:' merepresentasikan simpul dalam tree biner.
2. Konstruktor menggunakan def \_\_init\_\_(self, data):. Ketika Node dibuat, metode akan dipanggil untuk menginisialisasi. Ini akan menerima parameter 'data', yang merupakan nilai yang akan disimpan dalam simpul.
3. 'self.left = None' dan 'self.right = None': adalah atribut 'left' dan 'right' yang digunakan untuk menyimpan referensi dari simpul saat ini. Jika simpul baru di buat, kedua atribut di atur menjadi 'None' .

Tanda Tangan



4. Atribut 'val' berfungsi untuk menyimpan nilai dari simpul itu sendiri.
  5. Metode 'insert' berfungsi untuk menyisipkan elemen baru. Ini menerima parameter 'data', yang mana merupakan nilai dari elemen yang akan di sisipkan.
  6. Metode 'PrintTree' digunakan untuk mencetak nilai dalam tree biner secara inorder.
  7. Pada 'inorderTraversal' Berfungsi untuk melakukan penelusuran dari tree biner yang akan mengembalikan daftar nilai-nilai simpul urutan inorder.
  8. Variabel 'res' digunakan untuk menyimpan hasil penelusuran inorder.
  9. Selanjutnya memeriksa apakah 'root' bukan 'None'. Jika tidak, simpul akan melakukan penelusuran.
  10. Lalu melakukan rekursi yang berfungsi menelusuri anak kiri dari 'root' dan menyimpan hasilnya dalam 'res'.
  11. Menambahkan nilai 'root' ke dalam 'res' yang dilakukan setelah menelusuri anak kiri, simpul yang diproses terakhir adalah simpul yang paling kiri.
  12. 'return res' akan mengembalikan daftar yang berisi nilai dalam inorder.
  13. 'root = Node(27)' Membuat objek 'root' dengan nilai 27.
  14. Menyisipkan beberapa nilai ke dalam tree biner menggunakan metode 'insert'.
  15. Mencetak hasil penelusuran inorder dengan 'print(root.inorderTraversal(root))' dari tree biner dengan menggunakan metode 'inorderTraversal'.
5. Hapus Elemen

Tanda Tangan



```

class Node:
    def __init__(self, data):
        self.left = None
        self.right = None
        self.val = data

    def inorder(self, temp):
        if(not temp):
            return
        self.inorder(temp.left)
        print(temp.val, end=" ")
        self.inorder(temp.right)

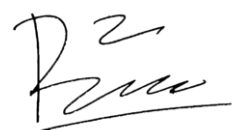
    def deleteDeepest(self, root, d_node):
        q = []
        q.append(root)
        while(len(q)):
            temp = q.pop(0)
            if temp is d_node:
                temp = None
                return
            if temp.right:
                if temp.right is d_node:
                    temp.right = None
                    return
                else:
                    q.append(temp.right)
            if temp.left:
                if temp.left is d_node:
                    temp.left = None
                    return
                else:
                    q.append(temp.left)

    def deletion(self, root, key):
        if root == None:
            return None
        if root.left == None and root.right == None:
            if root.val == key:
                return None
            else:
                return root

```

(Gambar 3.5.1)

Tanda Tangan



```
key_node = None
q = []
q.append(root)
temp = None
while(len(q)):
    temp = q.pop(0)
    if temp.val == key:
        key_node = temp
    if temp.left:
        q.append(temp.left)
    if temp.right:
        q.append(temp.right)
if key_node:
    x = temp.val
    self.deleteDeepest(root, temp)
    key_node.val = x
    self.deleteDeepest(root, temp)
return root

if __name__ == '__main__':
    root = Node(10)
    root.left = Node(11)
    root.left.left = Node(7)
    root.left.right = Node(12)
    root.right = Node(9)
    root.right.left = Node(15)
    root.right.right = Node(8)
    print("The tree before the deletion:")
    root.inorder(root)
    key = 11
    root = root.deletion(root, key)
    print()
    print("The tree after the deletion:")
    root.inorder(root)
```

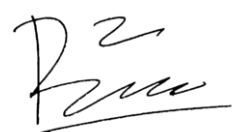
Python

The tree before the deletion:  
7 11 12 10 15 9 8  
The tree after the deletion:  
7 8 12 10 15 9

(Gambar 3.5.2)

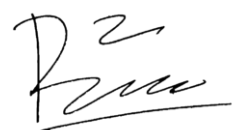
1. Untuk mendefinisikan kelas Node digunakan 'class Node:' merepresentasikan simpul dalam tree biner.
2. Konstruktor menggunakan def \_\_init\_\_(self, data):. Ketika Node dibuat, metode akan dipanggil untuk menginisialisasi. Ini akan menerima parameter 'data', yang merupakan nilai yang akan disimpan dalam simpul.
3. 'self.left = None' dan 'self.right = None': adalah atribut 'left' dan 'right' yang digunakan untuk menyimpan referensi dari simpul saat ini. Jika simpul baru di buat, kedua atribut di atur menjadi 'None' .
4. Atribut 'val' berfungsi untuk menyimpan nilai dari simpul itu sendiri.

Tanda Tangan



5. Metode 'insert' berfungsi untuk menyisipkan elemen baru. Ini menerima parameter 'data', yang mana merupakan nilai dari elemen yang akan di sisipkan.
6. Dengan 'deleteDeepest' akan menghapus simpul terdalam (paling kanan) dari tree biner yang akan membantu dalam operasi penghapusan simpul.
7. Selanjutnya 'deletion' berfungsi untuk menghapus simpul dengan nilai 'key' dari tree yang memiliki akar 'root' lalu akan memindai tree biner dengan pendekatan BFS menggunakan queue.
8. Untuk memeriksa apakah tree kosong digunakan 'if root == None:'. Jika ya, langsung mengembalikan 'None'.
9. Kemudian program akan memeriksa apakah 'root' merupakan simpul daun. Jika ya, dan nilai 'root' sama dengan 'key', simpul akan dihapus dan mengembalikan 'None'. Jika nilai 'root' berbeda dengan 'key', maka simpul tersebut tetap dipertahankan.
10. 'q = []' Inisialisasi antrian.
11. Memasukkan akar tree 'q.append(root)' ke dalam antrian.
12. Inisialisasi variabel 'temp'.
13. 'while(len(q)):' Melakukan loop selama antrian tidak kosong.
14. Memeriksa apakah nilai simpul yang diambil sama dengan 'key'. jika sama, akan menyimpan simpul tersebut ke dalam 'key\_node'.
15. 'if temp.left:' atau 'if temp.right:' Jika simpul memiliki anak kiri atau anak kanan, masukkan anak kiri atau kanan ke dalam antrian.
16. Untuk menyimpan nilai dari simpul terakhir dalam 'x'.
17. 'self.deleteDeepest(root, temp)' Menghapus simpul terakhir dari tree.
18. Menetapkan nilai 'x' ke simpul yang akan dihapus.
19. Menghapus simpul yang dipindahkan ke simpul yang akan dihapus dengan 'self.deleteDeepest(root, temp)'.

Tanda Tangan



20. 'return root' berfungsi mengembalikan akar tree yang diperbarui setelah operasi penghapusan.

## 6. Pencarian Pada Binary Search Tree (BST)

```
class Node:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.val = key

    def insert(self, key):
        if self is None:
            return Node(key)

        if key < self.val:
            if self.left is None:
                self.left = Node(key)
            else:
                self.left.insert(key)
        elif key > self.val:
            if self.right is None:
                self.right = Node(key)
            else:
                self.right.insert(key)

    def search(self, key):
        if self is None or self.val == key:
            return self

        if key < self.val:
            if self.left is None:
                return None
            else:
                return self.left.search(key)
        else:
            if self.right is None:
                return None
            else:
                return self.right.search(key)
```

(Gambar 3.6.1)

```
if __name__ == '__main__':
    root = Node(50)
    root.insert(30)
    root.insert(20)
    root.insert(40)
    root.insert(70)
    root.insert(60)
    root.insert(80)

    key = 6

    if root.search(key) is None:
        print(key, "not found")
    else:
        print(key, "found")

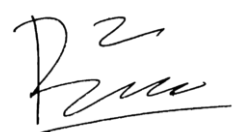
    key = 60

    if root.search(key) is None:
        print(key, "not found")
    else:
        print(key, "found")
```

Python

6 not found  
60 found

Tanda Tangan

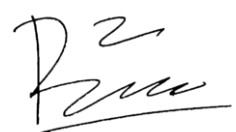




(Gambar 3.6.2)

1. Untuk mendefinisikan kelas Node digunakan 'class Node:' merepresentasikan simpul dalam tree biner.
2. Konstruktor menggunakan `def __init__(self, data):`. Ketika Node dibuat, metode akan dipanggil untuk menginisialisasi. Ini akan menerima parameter 'data', yang merupakan nilai yang akan disimpan dalam simpul.
3. 'self.left = None' dan 'self.right = None': adalah atribut 'left' dan 'right' yang digunakan untuk menyimpan referensi dari simpul saat ini. Jika simpul baru di buat, kedua atribut di atur menjadi 'None' .
4. Atribut 'val' berfungsi untuk menyimpan nilai dari simpul itu sendiri.
5. Metode 'insert' berfungsi untuk menyisipkan elemen baru.
6. Untuk mencari digunakan fungsi 'def search(self, key):': Metode 'search' dengan nilai tertentu dalam tree biner.
7. Memeriksa apakah simpul saat ini 'None' atau memiliki nilai yang sama dengan 'key' jika sama, mengembalikan simpul tersebut.
8. 'if key < self.val:' dan 'if key > self.val:' akan membandingkan nilai 'key' dengan nilai 'self.val' untuk menentukan arah pencarian.
9. Pembuatan tree biner dengan nilai-nilai tertentu menggunakan metode 'insert'.
10. Pencarian elemen dengan nilai dalam tree biner menggunakan metode 'search' saat nilai ditemukan, mencetak "found", jika tidak, mencetak "not found".

Tanda Tangan



#### IV. Latihan

```
class Node:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.val = key

def insert(root, key):
    if root is None:
        return Node(key)
    else:
        if key < root.val:
            root.left = insert(root.left, key)
        else:
            root.right = insert(root.right, key)
    return root

def inorder_traversal(root):
    if root:
        inorder_traversal(root.left)
        print(root.val, end=" ")
        inorder_traversal(root.right)

def preorder_traversal(root):
    if root:
        print(root.val, end=" ")
        preorder_traversal(root.left)
        preorder_traversal(root.right)

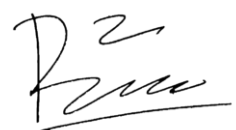
def postorder_traversal(root):
    if root:
        postorder_traversal(root.left)
        postorder_traversal(root.right)
        print(root.val, end=" ")

def minValueNode(node):
    current = node
    while(current.left is not None):
        current = current.left
    return current

def deleteNode(root, key):
    if root is None:
        return root
    if key < root.val:
        root.left = deleteNode(root.left, key)
    elif(key > root.val):
        root.right = deleteNode(root.right, key)
```

(Gambar 4.1)

Tanda Tangan



```

else:
    if root.left is None:
        temp = root.right
        root = None
        return temp
    elif root.right is None:
        temp = root.left
        root = None
        return temp
    temp = minValueNode(root.right)
    root.val = temp.val
    root.right = deleteNode(root.right, temp.val)
return root

# Contoh penggunaan
root = None
root = insert(root, 50)
root = insert(root, 30)
root = insert(root, 20)
root = insert(root, 40)
root = insert(root, 70)
root = insert(root, 60)
root = insert(root, 80)

print("Inorder Traversal:")
inorder_traversal(root)
print("\nPreorder Traversal:")
preorder_traversal(root)
print("\nPostorder Traversal:")
postorder_traversal(root)

key = 20
root = deleteNode(root, key)
print("\nInorder Traversal setelah menghapus", key, ":")
inorder_traversal(root)

```

Python

```

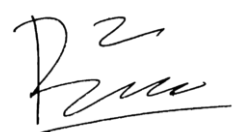
Inorder Traversal:
20 30 40 50 60 70 80
Preorder Traversal:
50 30 20 40 70 60 80
Postorder Traversal:
20 40 30 60 80 70 50
Inorder Traversal setelah menghapus 20 :
30 40 50 60 70 80

```

(Gambar 4.2)

1. Untuk mendefinisikan kelas Node digunakan 'class Node:' merepresentasikan simpul dalam tree biner.
2. Metode konstruktor 'def \_\_init\_\_(self, key):' untuk kelas Node. Menginisialisasi atribut left, right, dan val.
3. 'def insert(root, key):' berfungsi untuk menyisipkan elemen ke dalam tree biner jika tree kosong, maka fungsi akan membuat simpul baru sebagai akar. Saat sudah terisi, fungsi akan mencari posisi sesuai untuk menyisipkan elemen yang baru saja di tambahkan.
4. 'def inorder\_traversal(root):', 'def preorder\_traversal(root):', 'def postorder\_traversal(root):' Fungsi-fungsi yang melakukan

Tanda Tangan



penelusuran di dalam urutan inorder, preorder, serta postorder, yang masing-masing dengan menggunakan rekursi.

5. Selanjutnya menggunakan fungsi 'def minValueNode(node):' untuk menemukan simpul yang memiliki nilai terkecil dalam tree biner.
6. Menghapus simpul dengan nilai tertentu dari tree biner digunakan 'def deleteNode(root, key):'.

Tanda Tangan

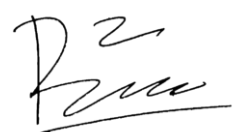
A handwritten signature in black ink, consisting of a stylized 'D' followed by a series of loops and a long horizontal stroke at the bottom.

## **V. Kesimpulan**

Kursus pemrograman Python di atas memberikan pemahaman yang kuat tentang cara mengimplementasikan dan mengoperasikan struktur data tree biner. Melalui penggunaan kelas dan metode, dapat membuat, menyisipkan, mencari, dan menghapus node di tree secara efisien. Metode rekursif di gunakan untuk melakukan tree traversal terurut, preorder atau postorder. Dengan cara ini, dapat dengan mudah mengakses dan memanipulasi data tree sesuai kebutuhan. Tes fungsional yang tersedia di laboratorium memastikan bahwa operasi pabrik berfungsi dengan benar dan memberikan hasil yang diharapkan.

Selain itu, praktik ini memberikan penerapan nyata algoritma pencarian dalam konteks tree biner. Dengan menggunakan teknik rekursi dan perbandingan nilai kunci, dapat dengan cepat menemukan entri tertentu di tree. Penerapan algoritma ini penting dalam berbagai aplikasi, seperti pencarian, pengurutan, dan analisis struktur data. Oleh karena itu, praktikum ini tidak hanya mengajarkan dasar-dasar pemrograman Python tetapi juga memberikan gambaran tentang pemrosesan data dan algoritma yang digunakan dalam konteks struktur data yang serumit tree biner.

Tanda Tangan

A handwritten signature in black ink, appearing to be 'P. Z. R.', is written over a light blue grid background.