LAPORAN PRAKTIKUM

STRUKTUR DATA

MODUL KE-07

TREE DALAM PYTHON

Disusun Oleh:

Nama : Oktario Mufti Yudha

NPM : 2320506044

Kelas : 04 (Empat)

Program Studi S1 Teknologi Informasi

Fakultas Teknik, Universitas Tidar

Genap 2023/2024

- **Tujuan Praktikum**

Adapun tujuan praktikum ini sebagai berikut :

a. Mahasiswa akan belajar tentang konsep dasar Binary Search Tree, termasuk aturan penyisipan dan penghapusan, serta algoritma traversal seperti inorder, preorder, dan postorder.
b. Mahasiswa akan mempelajari cara mengimplementasikan BST menggunakan bahasa pemrograman tertentu, seperti Python, Java, atau C++. Mereka akan belajar bagaimana membuat kelas atau struktur data untuk merepresentasikan simpul dan pohon secara keseluruhan.

- **Dasar Teori**

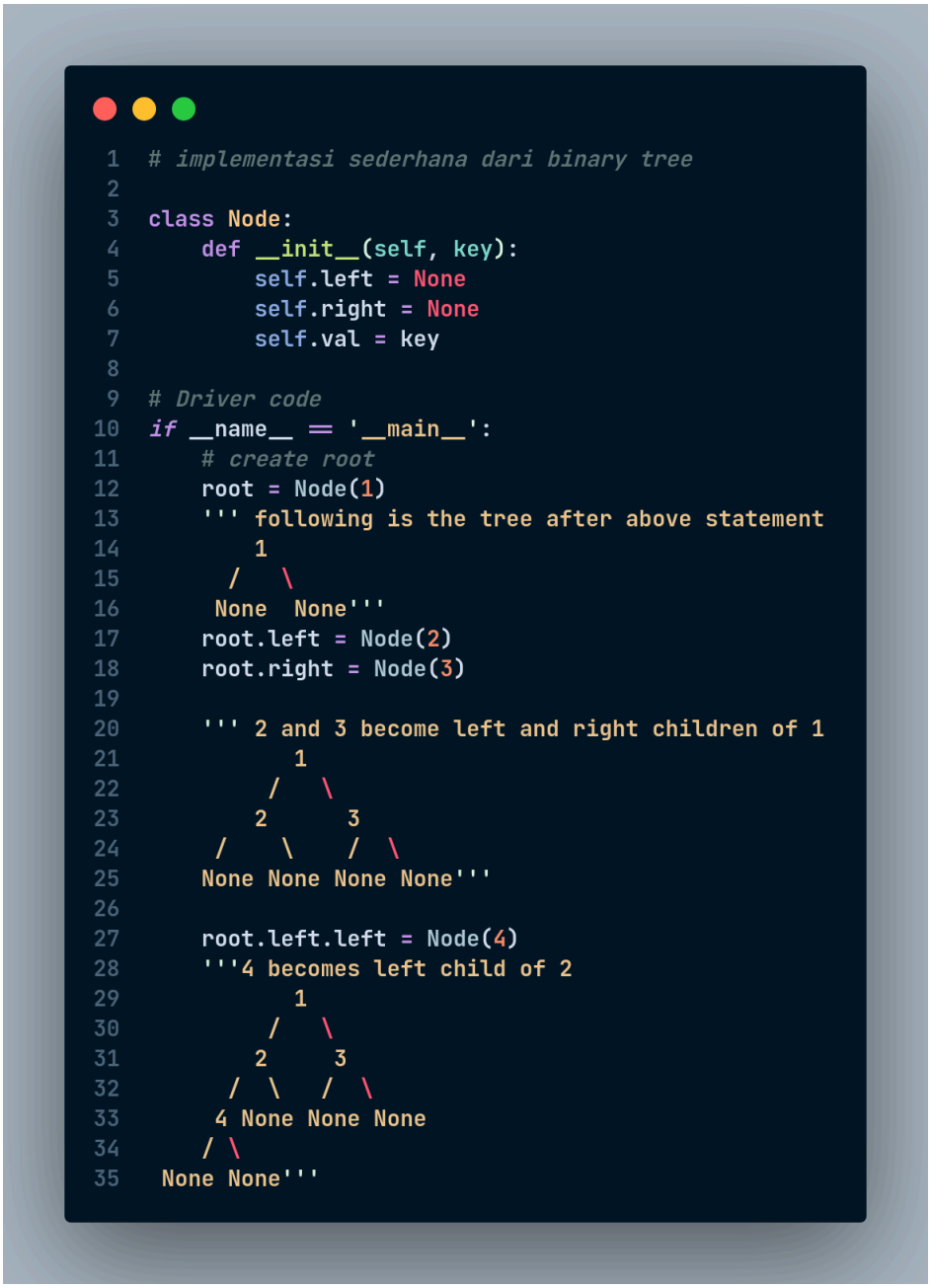
Struktur data pohon adalah sebuah struktur hierarkis yang digunakan untuk mewakili dan mengatur data dengan cara yang mudah untuk dinavigasi dan dicari. Ini merupakan kumpulan dari node yang saling terhubung melalui tepi dan memiliki hubungan hierarkis di antara node-node tersebut.

Terminologi dalam Struktur Data Pohon:

a. Node Induk: Node yang menjadi pendahulu suatu node disebut node induk dari node tersebut. {B} adalah node induk dari {D, E}.
b. Node Anak: Node yang merupakan pengganti langsung dari suatu node disebut node anak dari node tersebut. Contoh: {D, E} adalah node anak dari {B}.
c. Node Akar: Node paling atas dari sebuah pohon atau node yang tidak memiliki node induk disebut node akar. {A} adalah node akar dari pohon tersebut. Sebuah pohon yang tidak kosong harus mengandung tepat satu node akar dan tepat satu jalur dari akar ke semua node lain dalam pohon.
d. 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 pohon tersebut.
e. 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}.
f. Keturunan: Sebuah node x adalah keturunan dari node lain y jika dan hanya

- **Hasil dan Pembahasan**

  Implementasi sederhana dari binary tree



```
# implementasi sederhana dari binary tree

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

# Driver code
if __name__ == '__main__':
    # create root
    root = Node(1)
    ''' following is the tree after above statement
        1
      /   \
     None  None'''
    root.left = Node(2)
    root.right = Node(3)

    ''' 2 and 3 become left and right children of 1
           1
         /   \
        2      3
     /    \    /  \
    None None None None'''

    root.left.left = Node(4)
    '''4 becomes left child of 2
           1
         /   \
        2     3
      /  \   /  \
     4 None None None
    / \
 None None'''
```

(Gambar 3.1)

- Class Node: Ini adalah definisi dari sebuah kelas yang disebut Node. Kelas ini memiliki satu metode khusus yang disebut __init__, yang digunakan untuk menginisialisasi objek Node baru. Setiap objek Node memiliki tiga atribut: left, right, dan val. left dan right adalah referensi ke node anak kiri dan kanan dari node saat

ini, sementara val adalah nilai (biasanya disebut sebagai kunci) yang disimpan dalam node tersebut.

- Method __init__: Metode khusus __init__ digunakan untuk menginisialisasi objek Node. Ini mengambil satu argumen yang disebut key, yang merupakan nilai yang akan disimpan dalam node saat dibuat. Saat sebuah node dibuat, atribut left dan right diatur menjadi None, dan atribut val diatur menjadi nilai key yang diberikan.
- Driver code: Bagian ini merupakan bagian kode yang akan dieksekusi saat file Python dijalankan secara langsung, bukan diimpor sebagai modul. Ini digunakan untuk menguji kelas yang telah didefinisikan sebelumnya.
- Membuat root: Pada bagian ini, sebuah objek Node baru dibuat dengan nilai kunci 1 dan disimpan dalam variabel root. Ini adalah akar dari pohon yang akan dibuat.
- Mengatur anak kiri dan kanan dari root: Setelah membuat root, dua node baru dibuat dengan nilai kunci 2 dan 3, dan kemudian disimpan sebagai anak kiri dan kanan dari root, secara berturut-turut.
- Mengatur anak kiri dari node 2: Kemudian, sebuah node baru dengan nilai kunci 4 dibuat dan disimpan sebagai anak kiri dari node 2.

Penyisipan elemen

```python
# Penyisipan elemen
class Node:
    def __init__(self, data):
        self.left = None
        self.right = None
        self.data = data

    def insert(self, data):
# Compare the new value with the parent node
        if self.data:
            if data < self.data:
                if self.left is None:
                    self.left = Node(data)
                else:
                    self.left.insert(data)
            elif data > self.data:
                if self.right is None:
                    self.right = Node(data)
                else:
                    self.right.insert(data)
        else:
            self.data = data

# Print the tree
    def PrintTree(self):
        if self.left:
            self.left.PrintTree()
        print( self.data),
        if self.right:
            self.right.PrintTree()

# Use the insert method to add nodes
root = Node(12)
root.insert(6)
root.insert(14)
root.insert(3)
root.PrintTree()
```

(Gambar 3.2)

- Class Node: Ini adalah definisi dari sebuah kelas yang disebut Node. Setiap objek Node memiliki tiga atribut: left, right, dan data. left dan right adalah referensi ke node anak kiri dan kanan dari node saat ini, sedangkan data adalah nilai yang disimpan dalam node tersebut.
- Method __init__: Metode khusus __init__ digunakan untuk menginisialisasi objek Node. Ketika sebuah objek Node dibuat, atribut left dan right diatur menjadi None, dan atribut data diatur

menjadi nilai data yang diberikan.

- Method insert: Ini adalah metode yang digunakan untuk menyisipkan elemen baru ke dalam pohon. Pertama-tama, itu membandingkan nilai yang akan disisipkan dengan nilai data dari node saat ini. Jika nilai baru lebih kecil dari nilai data saat ini, itu mengecek anak kiri dari node saat ini. Jika anak kiri masih kosong, maka nilai baru akan disisipkan di sana; jika tidak, metode insert dipanggil secara rekursif pada anak kiri. Hal yang sama berlaku jika nilai baru lebih besar dari nilai data saat ini, tetapi kali ini untuk anak kanan.
- Method PrintTree: Metode ini digunakan untuk mencetak pohon secara in-order. Ini dilakukan dengan cara pertama mencetak anak kiri, kemudian mencetak nilai data node saat ini, dan terakhir mencetak anak kanan. Ini juga dilakukan secara rekursif.
- Membuat objek root: Pada bagian ini, sebuah objek Node baru dibuat dengan nilai data 12 dan disimpan dalam variabel root.
- Menyisipkan elemen: Tiga elemen baru disisipkan ke dalam pohon, yaitu 6, 14, dan 3, dengan menggunakan metode insert.
- Mencetak pohon: Metode PrintTree dipanggil pada root untuk mencetak pohon secara in-order.

In Order Traversal

```python
# in-oreder traversal
class Node:
    def __init__(self, data):
        self.left = None
        self.right = None
        self.data = data

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

# print the tree
    def PrintTree(self):
        if self.left:
            self.left.PrintTree()
        print(self.data),
        if self.right:
            self.right.PrintTree()

# in-order traversal
# Left -> Root -> Right
    def inorderTraversal(self, root):
        res = []
        if root:
            res = self.inorderTraversal(root.left)
            res.append(root.data)
            res = res + self.inorderTraversal(root.right)
        return res

# Use the insert method to add nodes
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))
```

(Gambar 3.3)

- Class Node: Ini adalah definisi dari sebuah kelas yang disebut Node. Setiap objek Node memiliki tiga atribut: left, right, dan data. left dan right adalah referensi ke node anak kiri dan kanan dari node saat ini, sedangkan data adalah nilai yang disimpan dalam node tersebut.
- Method __init__: Metode khusus __init__ digunakan untuk

menginisialisasi objek Node. Ketika sebuah objek Node dibuat, atribut left dan right diatur menjadi None, dan atribut data diatur menjadi nilai data yang diberikan.

- Method insert: Ini adalah metode yang digunakan untuk menyisipkan elemen baru ke dalam pohon. Metodenya sama dengan yang telah dijelaskan sebelumnya.
- Method PrintTree: Metode ini digunakan untuk mencetak pohon secara in-order. Ini dilakukan dengan cara pertama mencetak anak kiri, kemudian mencetak nilai data node saat ini, dan terakhir mencetak anak kanan. Ini juga dilakukan secara rekursif.
- Method inorderTraversal: Metode ini digunakan untuk melakukan penelusuran dalam-order pada pohon. Ini mengikuti urutan "kiri-akar-kanan". Secara rekursif, metode ini menelusuri pohon dari anak kiri, kemudian menambahkan nilai data node saat ini ke daftar res, dan terakhir menelusuri anak kanan. Ini dilakukan hingga seluruh pohon ditelusuri.
- Membuat objek root: Sebuah objek Node baru dibuat dengan nilai data 27 dan disimpan dalam variabel root.
- Menambahkan elemen: Beberapa elemen baru ditambahkan ke dalam pohon menggunakan metode insert.
- Mencetak hasil dari penelusuran dalam-order: Metode inorderTraversal dipanggil pada root untuk melakukan penelusuran dalam-order pada pohon, dan hasilnya dicetak.

Pre Order Traversal

```python
# pre-order traversal
class Node:
    def __init__(self, data):
        self.left = None
        self.right = None
        self.data = data

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

# print the tree
    def PrintTree(self):
        if self.left:
            self.left.PrintTree()
        print(self.data),
        if self.right:
            self.right.PrintTree()

# pre-order traversal
# Root -> Left ->Right
    def PreorderTraversal(self, root):
        res = []
        if root:
            res.append(root.data)
            res = res + self.PreorderTraversal(root.left)
            res = res + self.PreorderTraversal(root.right)
        return res

# Use the insert method to add nodes
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))
```

(Gambar 3.4)

- Class Node: Ini adalah definisi dari sebuah kelas yang disebut Node. Setiap objek Node memiliki tiga atribut: left, right, dan data. left dan right adalah referensi ke node anak kiri dan kanan dari node saat ini, sedangkan data adalah nilai yang disimpan dalam node tersebut.
- Method __init__: Metode khusus __init__ digunakan untuk

menginisialisasi objek Node. Ketika sebuah objek Node dibuat, atribut left dan right diatur menjadi None, dan atribut data diatur menjadi nilai data yang diberikan.

- Method insert: Ini adalah metode yang digunakan untuk menyisipkan elemen baru ke dalam pohon. Metodenya sama dengan yang telah dijelaskan sebelumnya.
- Method PrintTree: Metode ini digunakan untuk mencetak pohon secara in-order. Ini dilakukan dengan cara pertama mencetak anak kiri, kemudian mencetak nilai data node saat ini, dan terakhir mencetak anak kanan. Ini juga dilakukan secara rekursif.
- Method PreorderTraversal: Metode ini digunakan untuk melakukan penelusuran prapesan pada pohon. Ini mengikuti urutan "akar-kiri-kanan". Secara rekursif, metode ini pertama-tama menambahkan nilai data node saat ini ke daftar res, kemudian menelusuri anak kiri, dan terakhir menelusuri anak kanan. Ini dilakukan hingga seluruh pohon ditelusuri.
- Membuat objek root: Sebuah objek Node baru dibuat dengan nilai data 27 dan disimpan dalam variabel root.
- Menambahkan elemen: Beberapa elemen baru ditambahkan ke dalam pohon menggunakan metode insert.
- Mencetak hasil dari penelusuran prapesan: Metode PreorderTraversal dipanggil pada root untuk melakukan penelusuran prapesan pada pohon, dan hasilnya dicetak.

Post Order Traversal

```python
# post-order traversal
class Node:
    def __init__(self, data):
        self.left = None
        self.right = None
        self.data = data

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

# print the tree
    def PrintTree(self):
        if self.left:
            self.left.PrintTree()
        print(self.data),
        if self.right:
            self.right.PrintTree()

# post-order traversal
# Left ->Right -> Root
    def PostorderTraversal(self, root):
        res = []
        if root:
            res = self.PostorderTraversal(root.left)
            res = res + self.PostorderTraversal(root.right)
            res.append(root.data)
        return res

# Use the insert method to add nodes
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))
```

(Gambar 3.5)

- Class Node: Ini adalah definisi dari sebuah kelas yang disebut Node. Setiap objek Node memiliki tiga atribut: left, right, dan data. left dan right adalah referensi ke node anak kiri dan kanan dari node saat ini, sedangkan data adalah nilai yang disimpan dalam node tersebut.
- Method __init__: Metode khusus __init__ digunakan untuk

menginisialisasi objek Node. Ketika sebuah objek Node dibuat, atribut left dan right diatur menjadi None, dan atribut data diatur menjadi nilai data yang diberikan.

- Method insert: Ini adalah metode yang digunakan untuk menyisipkan elemen baru ke dalam pohon. Metodenya sama dengan yang telah dijelaskan sebelumnya.
- Method PrintTree: Metode ini digunakan untuk mencetak pohon secara in-order. Ini dilakukan dengan cara pertama mencetak anak kiri, kemudian mencetak nilai data node saat ini, dan terakhir mencetak anak kanan. Ini juga dilakukan secara rekursif.
- Method PostorderTraversal: Metode ini digunakan untuk melakukan penelusuran pospesan pada pohon. Ini mengikuti urutan "kiri-kanan-akar". Secara rekursif, metode ini pertama-tama menelusuri anak kiri, kemudian anak kanan, dan akhirnya menambahkan nilai data node saat ini ke daftar res. Ini dilakukan hingga seluruh pohon ditelusuri.
- Membuat objek root: Sebuah objek Node baru dibuat dengan nilai data 27 dan disimpan dalam variabel root.
- Menambahkan elemen: Beberapa elemen baru ditambahkan ke dalam pohon menggunakan metode insert.
- Mencetak hasil dari penelusuran pospesan: Metode PostorderTraversal dipanggil pada root untuk melakukan penelusuran pospesan pada pohon, dan hasilnya dicetak.

Hapus Elemen

```python
# hapus elemen
class Node:
    def __init__(self, data):
        self.left = None
        self.right = None
        self.data = data

# in order traversal of a binary tree
def inorder(temp):
    if not temp:
        return
    inorder(temp.left)
    print(temp.data, end=" ")
    inorder(temp.right)

# function to delete the given deepest node (d_node) in binary tree
def deleteDeepest(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)

# function to delete element in binary tree
def deletion(root, key):
    if root == None:
        return None
    if root.left == None and root.right == None:
        if root.key == key:
            return None
        else:
            return root
    key_node = None
    q = []
    q.append(root)
    temp = None
    while (len(q)):
        temp = q.pop(0)
        if temp.data == key:
            key_node = temp
        if temp.left:
            q.append(temp.left)
        if temp.right:
            q.append(temp.right)
    if key_node:
        x = temp.data
        deleteDeepest(root, temp)
        key_node.data = x
    return root

# Driver code
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("Inorder traversal before deletion: ", end="")
    inorder(root)
    key = 11
    root = deletion(root, key)
    print()
    print("Inorder traversal after deletion: ", end="")
    inorder(root)
```

(Gambar 3.6)

- Class Node: Ini adalah definisi dari sebuah kelas yang disebut Node. Setiap objek Node memiliki tiga atribut: left, right, dan data. left dan right adalah referensi ke node anak kiri dan kanan dari node saat ini, sedangkan data adalah nilai yang disimpan dalam node tersebut.
- Fungsi inorder: Ini adalah fungsi yang digunakan untuk melakukan penelusuran dalam-order pada pohon biner. Ini mencetak nilai data dari setiap node secara in-order (kiri-akar-kanan).
- Fungsi deleteDeepest: Ini adalah fungsi yang digunakan untuk menghapus node terdalam dari pohon biner. Ini menghapus node terdalam dengan mengganti nilainya dengan nilai node terakhir di level terakhir pohon biner.
- Fungsi deletion: Ini adalah fungsi utama untuk menghapus elemen dari pohon biner. Fungsi ini menerima dua argumen: root dari pohon biner dan kunci elemen yang ingin dihapus. Fungsi ini menggunakan pencarian dalam lebar (BFS) untuk menemukan node yang ingin dihapus. Setelah node ditemukan, itu memindahkan nilai dari node terakhir ke node yang ingin dihapus dan kemudian menghapus node terakhir.
- Driver code: Di dalam blok ini, sebuah pohon biner dibangun dengan beberapa node dan elemen-elemen tertentu ditambahkan ke dalamnya. Kemudian, penelusuran dalam-order dilakukan pada pohon biner sebelum dan sesudah penghapusan elemen tertentu. Ini untuk memverifikasi apakah penghapusan dilakukan dengan benar.

Pencarian function to search

```python
# pencarian function to search a given key in a given binary tree
class Node:
    # constructor to create a new binary node
    def __init__(self, key):
        self.left = None
        self.right = None
        self.key = key

#  A utility function to insert
# a new node with the given key in BST
def insert(node, key):
    # if the tree is empty, return a new node
    if node is None:
        return Node(key)

    # otherwise, recur down the tree
    if key < node.key:
        node.left = insert(node.left, key)
    else:
        node.right = insert(node.right, key)

    # return the (unchanged) node pointer
    return node

# utility function to search a given key in BST
def search(root, key):
    # base case: root is null or key is present at root
    if root is None or root.key == key:
        return root

    # key is greater than root's key
    if root.key < key:
        return search(root.right, key)

    # key is smaller than root's key
    return search(root.left, key)

# Driver code
if __name__ == '__main__':
    root = None
    root = insert(root, 50)
    insert(root, 30)
    insert(root, 20)
    insert(root, 40)
    insert(root, 70)
    insert(root, 60)
    insert(root, 80)

    # key to be found
    key = 6

    # search key in a BST
    if search(root, key) is None:
        print(key, "not found")
    else:
        print(key, "found")

    key = 60

    # search in a BST
    if search(root, key) is None:
        print(key, "not found")
    else:
        print(key, "found")

```

(Gambar 3.7)

- Class Node: Ini adalah definisi dari sebuah kelas yang disebut Node. Setiap xobjek Node memiliki tiga atribut: left, right, dan key. left dan right adalah referensi ke node anak kiri dan kanan dari

node saat ini, sedangkan key adalah nilai yang disimpan dalam node tersebut.

- Fungsi insert: Ini adalah fungsi utilitas untuk menyisipkan elemen baru ke dalam pohon. Fungsi ini memeriksa apakah pohon kosong. Jika pohon kosong, itu membuat node baru dan mengembalikannya. Jika tidak, itu melakukan rekursi ke kiri atau ke kanan tergantung pada nilai kunci yang disediakan.
- Fungsi search: Ini adalah fungsi utilitas untuk mencari elemen tertentu dalam pohon. Fungsi ini bekerja secara rekursif, memeriksa apakah root kosong atau apakah kunci yang dicari sama dengan kunci root. Jika tidak, itu memanggil dirinya sendiri rekursif ke kiri atau kanan tergantung pada nilai kunci yang disediakan.
- Driver code: Di dalam blok ini, sebuah pohon biner pencarian (BST) dibangun dengan beberapa node dan elemen-elemen tertentu ditambahkan ke dalamnya. Kemudian, pencarian dilakukan untuk mencari elemen tertentu dalam pohon. Hasil pencarian dicetak untuk menunjukkan apakah elemen tersebut ditemukan atau tidak.

- **Latihan**

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

# Fungsi untuk menyisipkan node baru dengan kunci tertentu
def insert(root, key):
    if root is None:
        return Node(key)
    else:
        if root.val < key:
            root.right = insert(root.right, key)
        else:
            root.left = insert(root.left, key)
    return root

# Fungsi untuk melakukan traversal in-order tree (kiri, akar, kanan)
def inorder(root):
    if root:
        inorder(root.left)
        print(root.val),
        inorder(root.right)

# Fungsi untuk mencari node dengan nilai minimum
def minValueNode(node):
    current = node
    while(current.left is not None):
        current = current.left
    return current

# Fungsi untuk menghapus elemen dari tree
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)
    else:
        if root.left is None:
            return root.right
        elif root.right is None:
            return root.left
        root.val = minValueNode(root.right).val
        root.right = deleteNode(root.right, root.val)
    return root

# Driver code
r = Node(50)
r = insert(r, 30)
r = insert(r, 20)
r = insert(r, 40)
r = insert(r, 70)
r = insert(r, 60)
r = insert(r, 80)

print("Traversal Inorder dari BST yang dibentuk adalah")
inorder(r)

print("\nHapus 20")
r = deleteNode(r, 20)
print("Traversal Inorder dari BST yang dimodifikasi adalah")
inorder(r)

print("\nHapus 30")
r = deleteNode(r, 30)
print("Traversal Inorder dari BST yang dimodifikasi adalah")
inorder(r)

print("\nHapus 50")
r = deleteNode(r, 50)
print("Traversal Inorder dari BST yang dimodifikasi adalah")
inorder(r)
```

(Gambar 4.1)

- Class Node: Mendefinisikan sebuah kelas Python yang disebut Node. Setiap objek Node memiliki atribut left, right, dan val. left dan right adalah referensi ke node anak kiri dan kanan dari node saat ini, sedangkan val adalah nilai yang disimpan dalam node tersebut.
- Method __init__: Ini adalah metode khusus untuk inisialisasi objek Node yang baru dibuat. Metode ini menerima parameter key dan mengatur atribut val node dengan nilai key. Atribut left dan right diatur sebagai None, menandakan bahwa node tersebut belum memiliki anak.
- Fungsi insert: Fungsi ini digunakan untuk menyisipkan node baru dengan kunci tertentu ke dalam pohon. Fungsi ini memeriksa apakah pohon (disebut juga root) kosong. Jika iya, maka fungsi akan membuat node baru dengan kunci yang diberikan. Jika tidak, maka fungsi akan mencari posisi yang tepat untuk menyisipkan node baru berdasarkan nilai kuncinya (key). Fungsi ini beroperasi secara rekursif.
- Fungsi inorder: Fungsi ini melakukan traversal in-order pada pohon. Traversal in-order mengunjungi node-node dalam urutan kiri, akar, kanan. Fungsi ini mencetak nilai val dari setiap node dalam urutan traversal in-order.
- Fungsi minValueNode: Fungsi ini digunakan untuk mencari node dengan nilai minimum dalam sebuah pohon. Fungsi ini melakukan iterasi ke kiri dari node yang diberikan sampai tidak ada anak kiri lagi, kemudian mengembalikan node tersebut.
- Fungsi deleteNode: Fungsi ini digunakan untuk menghapus elemen dari pohon. Fungsi ini menerima root pohon dan kunci elemen yang ingin dihapus. Fungsi ini beroperasi secara rekursif. Jika kunci yang ingin dihapus kurang dari nilai pada root, maka fungsi akan beroperasi pada anak kiri, dan sebaliknya untuk anak kanan. Jika node yang ditemukan sama dengan kunci, fungsi akan menghapusnya berdasarkan aturan khusus untuk pohon biner pencarian.
- Driver code: Di bagian ini, beberapa node ditambahkan ke dalam pohon menggunakan fungsi insert. Kemudian, beberapa elemen dihapus dari pohon menggunakan fungsi deleteNode. Hasil traversal

in-order sebelum dan setelah penghapusan dicetak untuk memverifikasi bahwa penghapusan dilakukan dengan benar.

- **Kesimpulan**

  Pada praktikum di atas, kita telah mempelajari implementasi dasar dari pohon biner pencarian (BST) dalam Python. Dengan menggunakan kelas Node sebagai elemen dasar dari pohon, kita dapat membuat struktur data pohon biner yang memungkinkan penyisipan, pencarian, dan penghapusan elemen dengan efisien. Fungsi-fungsi seperti insert, search, dan delete memanfaatkan sifat BST yang terurut untuk menjalankan operasi-operasi ini dengan kompleksitas waktu yang efisien, yaitu sekitar O(log n) untuk kasus rata-rata, dan O(n) dalam kasus terburuk. Selain itu, kita juga mengimplementasikan fungsi-fungsi tambahan seperti traversal in-order dan pencarian nilai minimum, yang berguna untuk mengelola dan memanipulasi data dalam pohon. Dengan pemahaman tentang struktur dan operasi-operasi yang mendasari pohon biner pencarian, kita dapat mengembangkan aplikasi yang menggunakan pohon biner sebagai struktur data yang efisien untuk menyimpan, mengatur, dan memanipulasi data.

  Selain itu, implementasi operasi-operasi dasar pada pohon biner pencarian memperkuat pemahaman tentang konsep dasar struktur data dan rekursi dalam pemrograman. Melalui proses pembuatan dan penggunaan fungsi-fungsi seperti penyisipan, pencarian, dan penghapusan, kita memperdalam pemahaman tentang bagaimana rekursi dapat digunakan untuk mengatasi masalah yang kompleks, seperti manipulasi struktur data yang kompleks. Selain itu, penggunaan pohon biner pencarian sebagai contoh memperkuat konsep abstraksi dan modularitas dalam pengembangan perangkat lunak, di mana kita dapat memisahkan operasi-operasi berbeda ke dalam fungsi-fungsi terpisah yang beroperasi secara mandiri tetapi bekerja bersama-sama untuk mencapai tujuan yang sama. Dengan demikian, materi di atas tidak hanya memperkenalkan konsep pohon biner pencarian, tetapi juga memperkuat pemahaman tentang rekursi, modularitas, dan abstraksi dalam pemrograman.