STRUKTUR DATA – TI.20.B.1
TEKNIK INFORMATIKA – UNIVERSITAS PELITA BANGSA
TUGAS PERTEMUAN – 13

Nama : Febro Herdyanto
NIM : 312010043
Mata Kuliah : Struktur Data
Dosen : Candra Naya,S.Kom.,M.Kom

SOAL:

1. Buatlah fungsi untuk menghapus suatu node pada Tree!
2. Buatlah program lengkap untuk memanipulasi dan mensimulasikan tree dengan berbasis menu!

JAWABAN:

1. Funsgi untuk menghapus suatu node pada Tree

```
void hapus(btree **node, int data) {
        if (*node == NULL) {
                return;
    }
        if(data == (*node)->data) { //data yang ingin dihapus ada di root
                if((*node)->kanan == NULL) {
                        *node = (*node)->kiri;
                }
                 else if((*node)->kiri == NULL) {
                        *node = (*node)->kanan;
                }
                else {
                        btree **successor = getSuccessor(&(*node)->kiri);
                        (*node)->data = (*successor)->data;
                        hapus(successor, (*successor)->data);
                }
    }
    else if(data < (*node)->data) {
                hapus(&(*node)->kiri, data);
    }
    else {
                hapus(&(*node)->kanan, data);
    }
}
```

2. Program memanipulasi data pada tree

```
import random

print('========================')
print('Nama  : Febro Herdyanto')
print('NIM   : 312010043')
print('Kelas : TI.20.B.1')
print('========================')


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


class Pohon:
    def __init__(self):
        self.node = None
        self.height = -1
        self.balance = 0

    def get_height(self):
        if self.node:
            return self.node.height
        else:
            return 0

    def insert(self, key):
        tree = self.node
        new_node = Node(key)

        if tree is None:
            self.node = new_node
            self.node.left = Pohon()
            self.node.right = Pohon()
```

Nama : Febro Herdyanto
NIM : 312010043
Mata Kuliah : Struktur Data
Dosen : Candra Naya,S.Kom.,M.Kom

```
        elif key < tree.key:
            self.node.left.insert(key)

        elif key > tree.key:
            self.node.right.insert(key)

        self.re_balance_tree()

    def re_balance_tree(self):
        self.update_heights(False)
        self.update_balances(False)
        while self.balance < -1 or self.balance > 1:
            if self.balance > 1:
                if self.node.left.balance < 0:
                    self.node.left.rotate_left()
                    self.update_heights()
                    self.update_balances()
                self.rotate_right()
                self.update_heights()
                self.update_balances()

            if self.balance < -1:
                if self.node.right.balance > 0:
                    self.node.right.rotate_right()
                    self.update_heights()
                    self.update_balances()
                self.rotate_left()
                self.update_heights()
                self.update_balances()

    def rotate_right(self):
        root = self.node
        left_child = self.node.left.node
        right_child = left_child.right.node

        self.node = left_child
        left_child.right.node = root
        root.left.node = right_child

    def rotate_left(self):
        root = self.node
        right_child = self.node.right.node
        left_child = right_child.left.node

        self.node = right_child
        right_child.left.node = root
        root.right.node = left_child

    def update_heights(self, recurse=True):
        if not self.node is None:
            if recurse:
                if self.node.left is not None:
                    self.node.left.update_heights()
                if self.node.right is not None:
                    self.node.right.update_heights()

            self.height = max(self.node.left.height,
                              self.node.right.height) + 1
        else:
            self.height = -1

    def update_balances(self, recurse=True):
        if not self.node is None:
            if recurse:
                if self.node.left is not None:
                    self.node.left.update_balances()
                if self.node.right is not None:
                    self.node.right.update_balances()

            self.balance = self.node.left.height - self.node.right.height
        else:
            self.balance = 0
```

Nama : Febro Herdyanto
NIM : 312010043
Mata Kuliah : Struktur Data
Dosen : Candra Naya,S.Kom.,M.Kom

```
    def check_balanced(self):
        if self is None or self.node is None:
            return True

        self.update_heights()
        self.update_balances()
        return ((abs(
            self.balance) < 2) and self.node.left.check_balanced() and
                self.node.right.check_balanced())

    def tree_in_order_traversal(self):
        if self.node is None:
            return []
        nodes_list = []
        l = self.node.left.tree_in_order_traversal()
        for i in l:
            nodes_list.append(i)

        nodes_list.append(self.node.key)

        l = self.node.right.tree_in_order_traversal()
        for i in l:
            nodes_list.append(i)
        return nodes_list

    def logical_successor(self, node):
        '''
        Find the smallese valued node in RIGHT child
        '''
        node = node.right.node
        if node != None:  # jika node tidak None

            while node.left != None:
                print("LS: traversing: " + str(node.key))
                if node.left.node == None:
                    return node
                else:
                    node = node.left.node
        return node

    def print_tree_as_tree_shape(self, node=None, level=0):
        if not node:
            node = self.node

        if node.right.node:
            self.print_tree_as_tree_shape(node.right.node, level + 1)
            print(('\t' * level), (' / '))
        print(('\t' * level), node.key)

        if node.left.node:
            print(('\t' * level), (' \\ '))
            self.print_tree_as_tree_shape(node.left.node, level + 1)

    def delete(self, key=0):
        key = int(key)
        # mencoba menghapus node yang di pilih
        if self.node != None:
            if int(self.node.key) == int(key):
                print("Deleting ... " + str(key))
                if self.node.left.node == None and self.node.right.node == None:
                    self.node = None  # leaves can be killed at will

                elif self.node.left.node == None:
                    self.node = self.node.right.node
                elif self.node.right.node == None:
                    self.node = self.node.left.node
                else:
                    replacement = self.logical_successor(self.node)
                    if replacement != None:  # sanity check
                        print("Found replacement for " + str(key) + " -> " +
str(replacement.key))
                        self.node.key = replacement.key
                        self.node.right.delete(replacement.key)
```

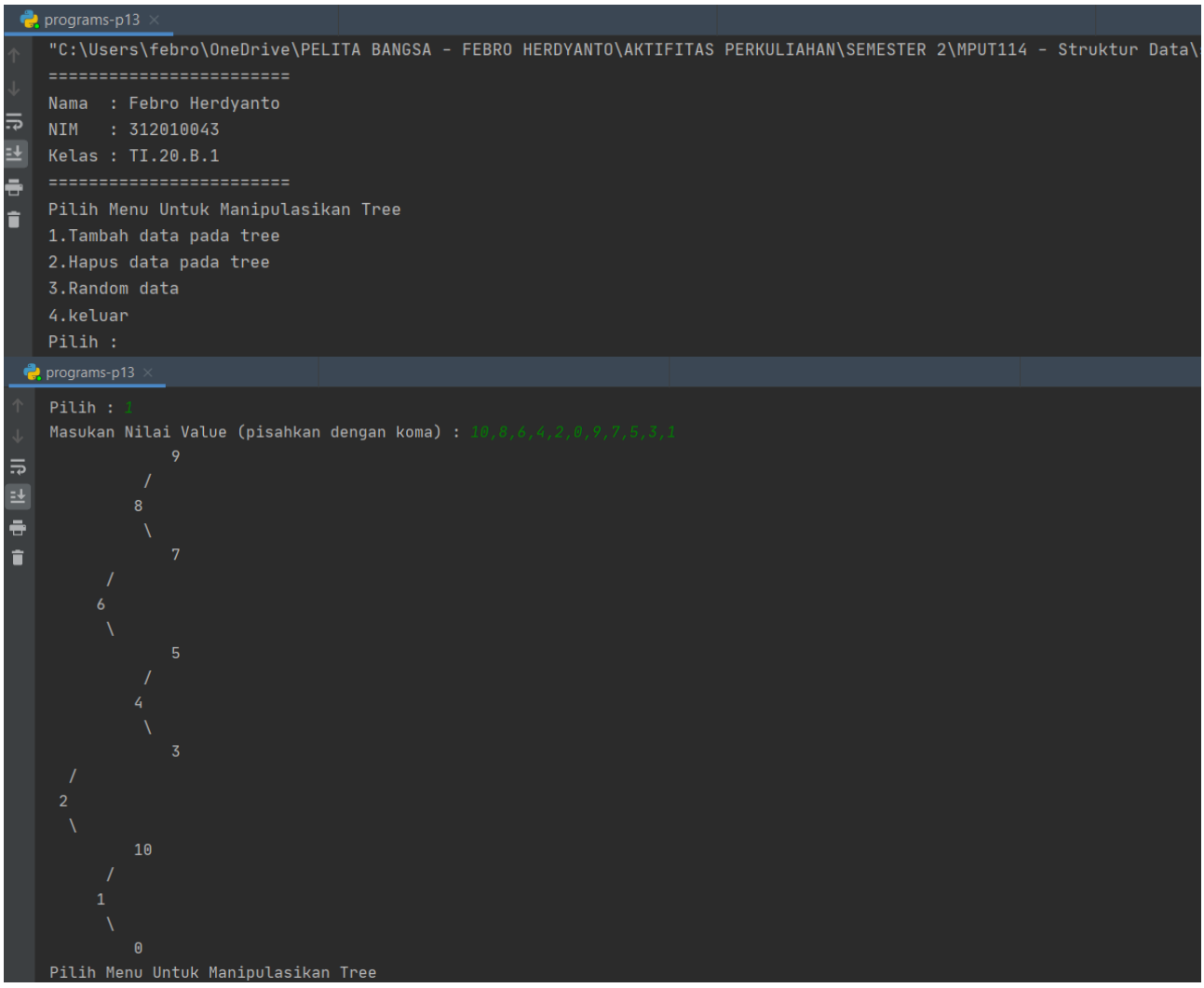
Nama : Febro Herdyanto
NIM : 312010043
Mata Kuliah : Struktur Data
Dosen : Candra Naya,S.Kom.,M.Kom

```
                self.re_balance_tree()
                return
            elif int(key) < int(self.node.key):
                self.node.left.delete(key)
            elif int(key) > int(self.node.key):
                self.node.right.delete(key)

            self.re_balance_tree()
        else:
            return


def create_random_node_list(n=10):
    random_node_list = random.sample(range(1, 100), n)
    print("Input :", random_node_list, "\n")
    return random_node_list


def create_avl_tree(node_list):
    tree = Pohon()
    for node in node_list:
        tree.insert(node)
    return tree


# if __name__ == "__main__":


loop = True
pilihan = 0;
tree = Pohon()
avl = tree

while loop == True:
    print("Pilih Menu Untuk Manipulasikan Tree")
    print("1.Tambah data pada tree")
    print("2.Hapus data pada tree")
    print("3.Random data")
    print("4.keluar")

    pilihan = int(input("Pilih : "))
    if (pilihan == 1):
        v_input = input("Masukan Nilai Value (pisahkan dengan koma) : ")
        vals = v_input.split(',')
        for val in vals:
            avl.insert(val)

        avl.print_tree_as_tree_shape()
    elif (pilihan == 2):

        print(avl.tree_in_order_traversal())
        k = input("Masukan nilai yang akan di hapus : ")
        avl.delete(int(k))
        avl.print_tree_as_tree_shape()

    elif (pilihan == 3):
        avl = create_avl_tree(create_random_node_list(8))
        avl.print_tree_as_tree_shape()
        print('\n')
        print(avl.tree_in_order_traversal())
    elif (pilihan == 4):
        loop = False
```

STRUKTUR DATA – TI.20.B.1
TEKNIK INFORMATIKA – UNIVERSITAS PELITA BANGSA
TUGAS PERTEMUAN – 13

Nama : Febro Herdyanto
NIM : 312010043
Mata Kuliah : Struktur Data
Dosen : Candra Naya,S.Kom.,M.Kom

Screenshot :



STRUKTUR DATA – TI.20.B.1
TEKNIK INFORMATIKA – UNIVERSITAS PELITA BANGSA
TUGAS PERTEMUAN – 13

Nama : Febro Herdyanto
NIM : 312010043
Mata Kuliah : Struktur Data
Dosen : Candra Naya,S.Kom.,M.Kom

programs-p13

```
Pilih : 2
['0', '1', '10', '2', '3', '4', '5', '6', '7', '8', '9']
Masukan nilai yang akan di hapus : 2
Deleting ... 2
LS: traversing: 6
LS: traversing: 4
LS: traversing: 3
Found replacement for 2 -> 3
Deleting ... 3
            9
          /
         8
          \
            7
      /
     6
      \
            5
          /
         4
   /
  3
   \
         10
      /
     1
      \
         0
Pilih Menu Untuk Manipulasikan Tree
```

programs-p13

```
Pilih : 3
Input : [27, 67, 2, 44, 49, 99, 63, 77]

         99
          \
            77
      /
     67
      \
         63
   /
  49
   \
         44
      /
     27
      \
         2


[2, 27, 44, 49, 63, 67, 77, 99]
Pilih Menu Untuk Manipulasikan Tree
```

programs-p13

```
4.keluar
Pilih : 4

Process finished with exit code 0
```