


Nama : Mutiara Novianti Rambe NIM : 064002300029	 Algoritma dan Pemrograman Dasar	Modul 11 Nama Dosen: 1. Abdul Rochman 2. Anung B. Ariwibowo
Hari/Tanggal: Selasa/14 Mei 2024		Nama Aslab: 1. Nathanael W. (064002100020) 2. Adrian Alfajri (064002200009)

MODUL 11 : AVL TREE 2

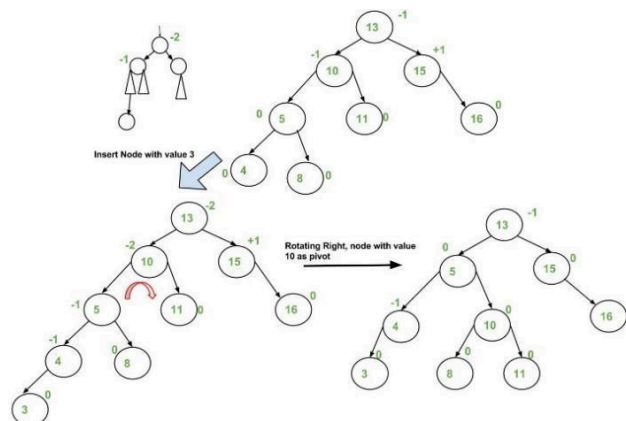
Deskripsi Modul : Memahami dan menerapkan ilmu struktur data dan algoritma untuk menyelesaikan masalah yang disajikan dengan menggunakan program berbasis bahasa Python.

No.	Elemen Kompetensi	Indikator Kinerja	Halaman
1.	Mampu memahami dan mengimplementasikan AVL Tree pada Python	Membuat dan memahami sebuah program yang menerapkan AVL Tree.	

TEORI SINGKAT

AVL Tree adalah Binary Search Tree yang memiliki perbedaan tinggi / level antara sub-tree kanan dan kirinya maksimal 1. AVL Tree digunakan untuk menyeimbangkan Binary Search Tree. Dengan menggunakan AVL Tree waktu pencarian dan bentuk tree yang disederhanakan. AVL Tree dapat direpresentasikan dengan menggunakan array maupun linked list.

Pengurutan node secara manual.



DAFTAR PERTANYAAN

1. Operasi apa saja yang dapat dilakukan pada AVL Tree?
2. Berapakah kompleksitas ketika melakukan operasi insert dan delete pada AVL Tree?

JAWABAN

JAWAB DI SINI!!!

- 1.
- 2.

LAB SETUP

Hal yang harus disiapkan dan dilakukan oleh praktikan untuk menjalankan praktikum modul ini, antara lain:

1. Menyiapkan IDE untuk membangun program python (Spyder, Sublime, VSCode, dll);
2. Python sudah terinstal dan dapat berjalan dengan baik di laptop masing-masing;
3. Menyimpan semua dokumentasi hasil praktikum pada laporan yang sudah disediakan.

ELEMEN KOMPETENSI I

Deskripsi : Mampu membuat program tentang AVL tree sesuai perintah yang ada.

Kompetensi Dasar : Membuat program yang mengimplementasikan AVL tree

LATIHAN 1

1. Buatlah sebuah program yang mengimplementasikan AVL tree yang melakukan operasi insert dan delete node pada tree.
2. Setiap program wajib menampilkan nama dan nim di bagian atas program.
Kondisi ketika code baru dijalankan

```
Aslab Struktur Data Algoritma Pak Anung
==R----16
==  L----5
==  |    L----4
==  R----22
==      L----20
==      R----64
AVL TREE OPERATION :
1.INSERT NODE
2.DELETE NODE
3.CLOSE
```

Kondisi ketika memilih menu 1

```

AVL TREE OPERATION :
1.INSERT NODE
2.DELETE NODE
3.CLOSE

Menu?:1
-----

Element yang ingin di insert: 9
==R----16
==  L----5
==   |    L----4
==   |    R----9
==   R----22
==       L----20
==       R----64
-----
    
```

Kondisi ketika memilih menu 2

```

AVL TREE OPERATION :
1.INSERT NODE
2.DELETE NODE
3.CLOSE

Menu?:2
-----

Element yang ingin dihapus: 22
==R----16
==  L----5
==   |    L----4
==   |    R----9
==   R----64
==       L----20
AVL TREE OPERATION :
1.INSERT NODE
2.DELETE NODE
3.CLOSE

Menu?:3
Close::
==R----16
==  L----5
==   |    L----4
==   |    R----9
==   R----64
==       L----20
    
```

Source Code

```
#Mutiara Novianti Rambe
#064002300029
import sys

class Node:
    def __init__(self, data):
        self.data = data
        self.parent = None
        self.left = None
        self.right = None
        self.bf = 0

class AVLTree:

    def __init__(self):
        self.root = None

    def __printHelper(self, currPtr, indent, last):
        if currPtr != None:
            sys.stdout.write(indent)
            if last:
                sys.stdout.write("R----")
                indent += "  "
            else:
                sys.stdout.write("L----")
                indent += "|  "

            print(currPtr.data)

            self.__printHelper(currPtr.left, indent, False)
            self.__printHelper(currPtr.right, indent, True)

    def __searchTreeHelper(self, node, key):
        if node == None or key == node.data:
            return node

        if key < node.data:
            return self.__searchTreeHelper(node.left, key)
        return self.__searchTreeHelper(node.right, key)

    def __deleteNodeHelper(self, node, key):
        if node == None:
            return node
        elif key < node.data:
            node.left = self.__deleteNodeHelper(node.left, key)
```

```
elif key > node.data:
    node.right = self.__deleteNodeHelper(node.right, key)
else:
    if node.left == None and node.right == None:
        node = None
    elif node.left == None:
        node = node.right
    elif node.right == None:
        node = node.left
    else:
        temp = self.minimum(node.right)
        node.data = temp.data
        node.right = self.__deleteNodeHelper(node.right, temp.data)

if node is not None:
    self.__updateBalance(node)

return node

def __updateBalance(self, node):
    if node == None:
        return

    node.bf = self.__getHeight(node.right) - self.__getHeight(node.left)

    if node.bf > 1 or node.bf < -1:
        self.__rebalance(node)

    if node.parent != None:
        self.__updateBalance(node.parent)

def __rebalance(self, node):
    if node.bf > 1:
        if node.right.bf < 0:
            self.rightRotate(node.right)
            self.leftRotate(node)
        else:
            self.leftRotate(node)
    elif node.bf < -1:
        if node.left.bf > 0:
            self.leftRotate(node.left)
            self.rightRotate(node)
        else:
            self.rightRotate(node)

def __getHeight(self, node):
```

```
if node == None:
    return 0
return max(self.__getHeight(node.left), self.__getHeight(node.right)) + 1

def __preOrderHelper(self, node, result):
    if node != None:
        result.append(node.data)
        self.__preOrderHelper(node.left, result)
        self.__preOrderHelper(node.right, result)

def __postOrderHelper(self, node, result):
    if node is not None:
        self.__postOrderHelper(node.left, result)
        self.__postOrderHelper(node.right, result)
        result.append(node.data)

def preorder(self):
    result = []
    self.__preOrderHelper(self.root, result)
    return result

def postorder(self):
    result = []
    self.__postOrderHelper(self.root, result)
    return result

def insert(self, data):
    if not self.root:
        self.root = Node(data)
    else:
        self.__insertHelper(self.root, data)

def __insertHelper(self, node, data):
    if data < node.data:
        if node.left is None:
            node.left = Node(data)
            node.left.parent = node
            self.__updateBalance(node.left)
        else:
            self.__insertHelper(node.left, data)
    elif data > node.data:
        if node.right is None:
            node.right = Node(data)
            node.right.parent = node
            self.__updateBalance(node.right)
        else:
            self.__insertHelper(node.right, data)
```

```
        self.__insertHelper(node.right, data)

def delete(self, data):
    self.root = self.__deleteNodeHelper(self.root, data)

def minimum(self, node):
    while node.left != None:
        node = node.left
    return node

def rightRotate(self, z):
    y = z.left
    T3 = y.right

    y.right = z
    z.left = T3

    if T3 is not None:
        T3.parent = z

    y.parent = z.parent
    if z.parent is None:
        self.root = y
    elif z == z.parent.left:
        z.parent.left = y
    else:
        z.parent.right = y

    z.parent = y
    z.bf = z.bf + 1 - min(y.bf, 0)
    y.bf = y.bf + 1 + max(z.bf, 0)

def leftRotate(self, z):
    y = z.right
    T2 = y.left

    y.left = z
    z.right = T2

    if T2 is not None:
        T2.parent = z

    y.parent = z.parent
    if z.parent is None:
        self.root = y
    elif z == z.parent.left:
```

```

        z.parent.left = y
    else:
        z.parent.right = y

    z.parent = y
    z.bf = z.bf - 1 - max(y.bf, 0)
    y.bf = y.bf - 1 + min(z.bf, 0)

def main():
    avl = AVLTree()
    initial_digits = [64, 2, 15, 9, 8, 29] # NIM: 064002300029

    for digit in initial_digits:
        avl.insert(digit)
    print("Mutiar Novianti Rambe_064002300029")
    avl._AVLTree__printHelper(avl.root, "", True)

    while True:
        print("\nAVL TREE OPERATION :")
        print("1.INSERT NODE")
        print("2.DELETE NODE")
        print("3.CLOSE")
        choice = input("Menu?: ")

        if choice == '1':
            data = int(input("Element yang ingin di insert: "))
            avl.insert(data)
            avl._AVLTree__printHelper(avl.root, "", True)
        elif choice == '2':
            data = int(input("Element yang ingin di delete: "))
            avl.delete(data)
            avl._AVLTree__printHelper(avl.root, "", True)
        elif choice == '3':
            avl._AVLTree__printHelper(avl.root, "", True)
            break
        else:
            print("Invalid choice. Please choose again.")

if __name__ == "__main__":
    main()

```


Screenshot

```
Mutiara Novianti Rambe_064002300029
R----15
  L----8
  |  L----2
  |  R----9
  R----64
    L----29
```

AVL TREE OPERATION :

- 1.INSERT NODE
- 2.DELETE NODE
- 3.CLOSE

```
Menu?: 1
Element yang ingin di insert: 13
R----15
  L----8
  |  L----2
  |  R----9
  |      R----13
  R----64
    L----29
```

AVL TREE OPERATION :

- 1.INSERT NODE
- 2.DELETE NODE
- 3.CLOSE

```
Menu?: 2
Element yang ingin di delete: 64
R----15
  L----13
  R----29
```

AVL TREE OPERATION :

- 1.INSERT NODE
- 2.DELETE NODE
- 3.CLOSE

```
Menu?: 3
R----15
  L----13
  R----29
```

KESIMPULAN

Buatlah kesimpulan hasil praktikum modul ini!!! (MINIMAL 3 BARIS)

CEKLIST

1. Memahami dan mengimplementasikan AVL tree pada Python (✓)

REFERENSI

<https://www.geeksforgeeks.org/avl-tree-set-1-insertion/>

<https://algorithmtutor.com/Data-Structures/Tree/AVL-Trees/>