МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Алгоритмы и структуры данных»

Тема: Реализация и исследование АВЛ-деревьев

Студент гр. 3344		Тукалкин. В.А.
Преподаватель		Иванов Д.В.

Санкт-Петербург

2024

Цель работы

Создание структуры данных АВЛ-дерева и исследование данной структуры.

Задание

В предыдущих лабораторных работах вы уже проводили исследования и эта не будет исключением. Как и в прошлые разы лабораторную работу можно разделить на две части:

1) решение задач на платформе moodle
2) исследование по заданной теме

В заданиях в качестве подсказки будет изложена основная структура данных (класс узла) и будет необходимо реализовать несколько основных функций: проверка дерева (является ли оно АВЛ деревом), нахождение разницы между связными узлами, вставка узла.

В качестве исследования нужно самостоятельно:

- реализовать функции удаления узлов: любого, максимального и минимального
- сравнить время и количество операций, необходимых для реализованных операций, с теоретическими оценками (очевидно, что проводить исследования необходимо на разных объемах данных)

Также для очной защиты необходимо подготовить визуализацию дерева.

В отчете помимо проведенного исследования необходимо приложить код всей получившей структуры: класс узла и функции.

Выполнение работы

Функции:

1) insert(val, node) – O(logN) – добавляет элемент в АВЛ-дерево, затем делает балансировку, при помощи поворотов (leftRotate(x) и rightRotate(y)).
2) diff(root) – O(logN) – считает минимальную разность родительского и дочернего узлов.
3) getBalance(node) – O(1) – возвращает разность высот между левым и правым поддеревьями.
4) height(node) – O(1) – возвращает высоту дерева.
5) leftRotate(x) – O(1) – левый малый поворот дерева.
6) rightRotate(y) – O(1) – правый малый поворот дерева.
7) visualizeTree(root) – O(logN) – функция визуализации дерева, сделанная при помощи библиотеки Pillow.
8) minValueNode(node) – O(logN) – переходит к минимальному элементу дерева (самому левому).
9) maxValueNode(node) – O(logN) – переходит к максимальному элементу дерева (самому правому).
10) deleteNode(val, node) – O(logN) – удаление узла по значению.
11) deleteMinNode(node) – O(1) – удаление минимального узла, использует функцию deleteNode.
12) deleteMaxNode(node) – O(1) – удаление максимального узла, использует функцию deleteNode.

Класс Node:

1) __init__(self, val, left, right) – O(1) – создаёт узел для дерева.

Тестирование программы

Тесты для проверки корректности работы реализованной структуры данных АВЛ-дерева находятся в файле tests.py. Каждый тест покрывает основные операции. На рисунке 1 показан пример вывода дерева.

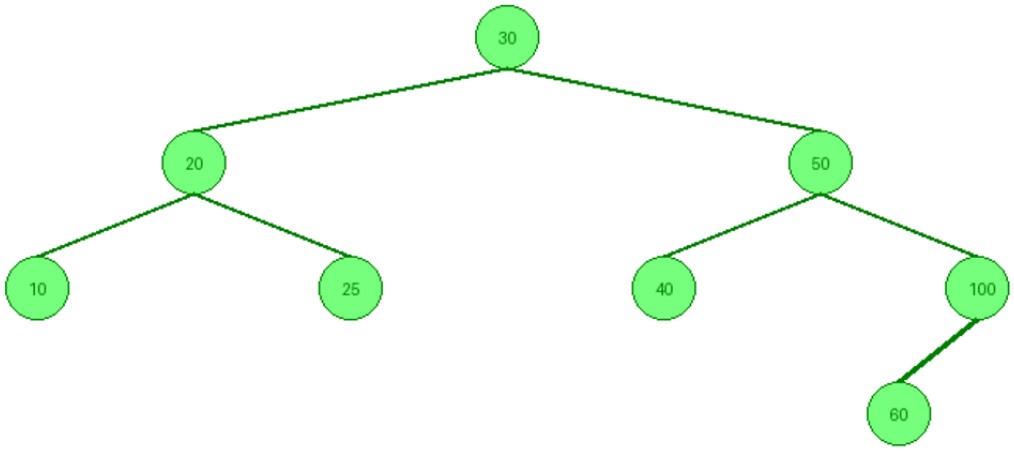

Рис.1 пример вывода дерева

Результаты исследования:

1) длина 10:

Вставка – 0.000273

Удаление – 0.000143

2) длина 1000:

Вставка – 0.004523

Удаление – 0.002486

3) длина 100000:

Вставка – 0.782531

Удаление – 0.398794

Выводы

Была создана структура данных АВЛ-дерева и проведено исследование на разных объёмах данных.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
from modules.insert import insert
from modules.visualizeTree import visualizeTree


if __name__ == '__main__':
    root = None
    values = [10, 20, 30]
    for val in values:
        root = insert(val, root)
    root = insert(40, root)
    visualizeTree(root)
```

Название файла: deleteMaxNode.py

```
from modules.maxValueNode import maxValueNode
from modules.deleteNode import deleteNode


def deleteMaxNode(node):
    if not node:
        return node

    max_node = maxValueNode(node)
    return deleteNode(max_node.val, node)
```

Название файла: deleteMinNode.py

```
from modules.minValueNode import minValueNode
from modules.deleteNode import deleteNode


def deleteMinNode(node):
    if node is None:
        return node

    min_node = minValueNode(node)
    return deleteNode(min_node.val, node)
```

Название файла: deleteNode.py

```
from modules.rightRotate import rightRotate
from modules.leftRotate import leftRotate
from modules.getBalance import getBalance
from modules.height import height
from modules.minValueNode import minValueNode


def deleteNode(val, node):
    if not node:
        return node

    if val < node.val:
        node.left = deleteNode(val, node.left)
    elif val > node.val:
```

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

            temp = minValueNode(node.right)
            node.val = temp.val
            node.right = deleteNode(temp.val, node.right)

        # балансировка
        if node is None:
            return node

        node.height = 1 + max(height(node.left), height(node.right))

        balance = getBalance(node)

        # малый правый поворот
        if balance > 1 and getBalance(node.left) >= 0:
            return rightRotate(node)

        # малый левый поворот
        if balance < -1 and getBalance(node.right) <= 0:
            return leftRotate(node)

        # левый большой поворот
        if balance > 1 and getBalance(node.left) < 0:
            node.left = leftRotate(node.left)
            return rightRotate(node)

        # правый большой поворот
        if balance < -1 and getBalance(node.right) > 0:
            node.right = rightRotate(node.right)
            return leftRotate(node)

        return node
```

Название файла: diff.py

```
def diff(root):
    min_diff = float('inf')

    def dfs(node):
        if not node:
            return
        nonlocal min_diff

        if node.left:
            min_diff = min(min_diff, abs(node.val - node.left.val))
            dfs(node.left)
```

```
            if node.right:
                min_diff = min(min_diff, abs(node.val - node.right.val))
                dfs(node.right)

        dfs(root)
        return min_diff
```

Название файла: getBalance.py

```
from modules.height import height


def getBalance(node):
    if node is None:
        return 0
    return height(node.left) - height(node.right)
```

Название файла: height.py

```
def height(node):
    if node is None:
        return 0
    return node.height
```

Название файла: TimSort.py

```
from modules.Node import Node
from modules.getBalance import getBalance
from modules.height import height
from modules.rightRotate import rightRotate
from modules.leftRotate import leftRotate

def insert(val, node):
    if node is None:
        return Node(val)

    if val < node.val:
        node.left = insert(val, node.left)
    else:
        node.right = insert(val, node.right)

    node.height = 1 + max(height(node.left), height(node.right))

    balance = getBalance(node)

    # малый правый поворот
    if balance > 1 and val < node.left.val:
        return rightRotate(node)

    # малый левый поворот
    if balance < -1 and val > node.right.val:
        return leftRotate(node)

    # левый большой поворот
    if balance > 1 and val > node.left.val:
        node.left = leftRotate(node.left)
        return rightRotate(node)
```

```
    # правый большой поворот
    if balance < -1 and val < node.right.val:
        node.right = rightRotate(node.right)
        return leftRotate(node)

    return node
```

Название файла: leftRotate.py

```
 from modules.height import height


def leftRotate(x):
    y = x.right
    tmp = y.left

    y.left = x
    x.right = tmp

    x.height = 1 + max(height(x.left), height(x.right))
    y.height = 1 + max(height(y.left), height(y.right))
    return y
```

Название файла: maxValueNode.py

```
def maxValueNode(node):
    current = node
    while current.right is not None:
        current = current.right
    return current
```

Название файла: minValueNode.py

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

Название файла: Node.py

```
from typing import Union


class Node:
    def __init__(self, val, left=None, right=None):
        self.val = val
        self.left: Union[Node, None] = left
        self.right: Union[Node, None] = right
        self.height: int = 1
```

Название файла: rightRotate.py

```
from modules.height import height


def rightRotate(y):
    x = y.left
    tmp = x.right

    x.right = y
```

```
        y.left = tmp

        y.height = 1 + max(height(y.left), height(y.right))
        x.height = 1 + max(height(x.left), height(x.right))
        return x
```

Название файла: visualizeTree.py

```
from PIL import Image, ImageDraw


def visualizeTree(root):
    if root is None:
        return

    h = root.height
    w = 2 ** (h - 1)

    image_width = w * 100
    image_height = h * 100
    node_radius = 20
    level_height = 80

    image = Image.new("RGB", (image_width, image_height), "white")
    draw = ImageDraw.Draw(image)

    def draw_node(node, x, y, level):
        if node is None:
            return

        # Draw the node circle
        draw.ellipse((x - node_radius, y - node_radius, x +
node_radius, y + node_radius), outline="green",
                     fill=(118,255,125))
        draw.text((x - 5, y - 5), str(node.val), fill="green")

        if node.left:
            x_left = x - image_width // (2 ** (level + 2))
            y_left = y + level_height
            draw.line((x, y + node_radius, x_left, y_left -
node_radius), fill="green", width=3)
            draw_node(node.left, x_left, y_left, level + 1)

        if node.right:
            x_right = x + image_width // (2 ** (level + 2))
            y_right = y + level_height
            draw.line((x, y + node_radius, x_right, y_right -
node_radius), fill="green", width=3)
            draw_node(node.right, x_right, y_right, level + 1)

    draw_node(root, image_width // 2, 50, 0)
    image.show()
```

Название файла: tests.py

```
from modules.getBalance import getBalance
from modules.height import height
from modules.insert import insert
from modules.diff import diff
```

```
from modules.deleteNode import deleteNode
from modules.maxValueNode import maxValueNode
from modules.minValueNode import minValueNode


def generate(values):
    root = None
    for val in values:
        root = insert(val, root)
    return root


def test_diff():
    root = generate([10, 21, 30])
    assert diff(root) == 9


def test_insert():
    root = generate([10, 20, 30])
    assert insert(40, root).right.right.val == 40


def test_getBalance():
    root = generate([10, 20, 30, 40])
    assert getBalance(root) == -1


def test_height():
    root = generate([10, 20, 30, 40])
    assert height(root) == 3


def test_deleteNode():
    root = generate([10, 20, 30])
    assert deleteNode(20, root).val == 30


def test_minValueNode():
    root = generate([10, 20, 30])
    assert minValueNode(root).val == 10


def test_maxValueNode():
    root = generate([10, 20, 30])
    assert maxValueNode(root).val == 30


test_diff()
test_insert()
test_getBalance()
test_height()
test_deleteNode()
test_minValueNode()
test_maxValueNode()
```