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

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: АВЛ-деревья - поиск и удаление. Демонстрация

Студент гр. 0303 _____ Калмак Д.А.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2021

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Калмак Д.А.

Группа 0303

Тема работы: АВЛ-деревья - поиск и удаление. Демонстрация

Исходные данные:
АВЛ-дерево. "Демонстрация" - визуализация структур данных/алгоритмов. Демонстрация должна быть подробной и понятной (в том числе сопровождаться пояснениями), чтобы программу можно было использовать при объяснении используемой структуры данных и выполняемых с нею действий. Помимо демонстрации (визуализации) следует на каждом шаге выводить текстовые объяснения того, что происходит.

Содержание пояснительной записки:
«Содержание», «Введение», «Узлы АВЛ-дерева», «Структура данных АВЛ-дерево», «Тестирование», «Заключение», «Список использованных источников», «Приложение А. Исходный код программы»

Предполагаемый объем пояснительной записки:
Не менее 17 страниц.

Дата выдачи задания: 16.10.2021
Дата сдачи реферата: 12.12.2021
Дата защиты реферата: 17.12.2021

Студент _____ Калмак Д.А.

Преподаватель _____ Берленко Т.А.

АННОТАЦИЯ

В курсовой работе реализована структура данных АВЛ-дерево. Для узлов АВЛ-дерева реализован собственный класс. АВЛ-дерево обладает методами поиска и удаления. Вместе с этим реализованы методы вставки узлов, измерения высоты дерева и поддерева. Для удобства визуализации реализован метод вывода дерева в консоль. Методы имеют пояснения. Балансировка АВЛ-дерева осуществляется методами малого левого поворота, малого правого поворота, большого левого поворота и большого правого поворота.

СОДЕРЖАНИЕ

	Введение	5
1.	Узлы АВЛ-дерева	6
1.1.	Поля	6
1.2.	Методы	6
2.	Структура данных АВЛ-дерево	7
2.1.	Поля	7
2.2.	Методы	7
3.	Тестирование	9
3.1.	Тестирование	9
3.2.	Тестирование особых случаев	10
	Заключение	11
	Список использованных источников	12
	Приложение А. Исходный код программы	13

ВВЕДЕНИЕ

Цель работы.

Реализовать структуру данных АВЛ-дерево с методами поиска и удаления.

Задачи.

- Реализовать структуру данных АВЛ-дерево.
- Реализовать методы поиска и удаления.
- Визуализировать структуру данных АВЛ-дерево.
- Визуализировать методы.
- Протестировать работу.

1. УЗЛЫ АВЛ-ДЕРЕВА

1.1. Поля

self.key – ключ узла.

self.left – левый ребенок узла.

self.right – правый ребенок узла.

self.height – высота поддерева с корнем в данном узле.

1.2. Методы

__str__(self) – переопределение метода строкового представления объекта. Возвращается ключ узла и его высота поддерева.

2. СТРУКТУРА ДАННЫХ АВЛ-ДЕРЕВО

2.1. Поля

self.root – корень АВЛ-дерева. Изначально равен None.

2.2. Методы

height(self, node) – возвращает высоту поддерева с корнем в узле node.

insert(self, key) – вставка узла с ключом key в АВЛ-дерево. Если корня АВЛ-дерева нет, то корнем дерева становится узел с ключом key, иначе запускается рекурсивная функция вставки узла с ключом key insert_rec(self, key, node), в которую также передается корень АВЛ-дерева.

insert_rec(self, key, node) – вставка осуществляется по правилу, если ключ нового узла меньше текущего на этапе рекурсии, то запускается функция вставки в левое поддерево, иначе в правое поддерево.

Метод вставки сопровождается балансировкой с помощью малого левого поворота, малого правого поворота, большого левого поворота, большого правого поворота. Вставка каждого узла визуализирована в консоли, причем визуализация методов поворота предусмотрена.

delete(self, key) – удаление узла с ключом key. Запускается рекурсивная функция удаления узла с ключом key delete_rec(self, key, node), в которую также передается корень АВЛ-дерева. Рекурсивно ищется узел с ключом key. Если найденный узел имеет только одного ребенка, значения обмениваются ребенком, узел удаляется. Если найденный узел имеет двух детей, то найденному узлу присваивается значение узла с минимальным key в правом поддереве найденного узла. Для этого реализован метод minimum(self, node), который возвращает узел с минимальным ключом, то есть самый левый. Узел с минимальным key в правом поддереве удаляется.

Метод удаления узла, у которого два ребенка, сопровождается балансировкой с помощью малого левого поворота, малого правого поворота,

большого левого поворота, большого правого поворота. Удаление узла визуализировано в консоли. Методы поворота также визуализированы.

find(self, key, node, a) – поиск узла в АВЛ-дереве. Рекурсивно ищется узел с ключом key. Поиск проходит с учетом правила: левый ребенок обладает ключом, значение которого меньше значения его родителя, и наоборот для правого ребенка. Возвращается список a с информацией о поиске.

Метод поиска визуализирует ход поиска узла со значением key. Если узел найден, то в списке содержится информация об узле, иначе список содержит информацию, что узла нет.

small_l_rotate(self, node) – метод малого левого поворота. Используется для балансировки.

big_l_rotate(self, node) – метод большого левого поворота. Используется для балансировки.

small_r_rotate(self, node) – метод малого правого поворота. Используется для балансировки.

big_r_rotate(self, node) – метод большого правого поворота. Используется для балансировки.

print_avl(self, current, indention, child_r, check) – метод для визуализации АВЛ-дерева.

root(self) – метод возвращает корень АВЛ-дерева.

Разработанный код см. в Приложении А.

3. ТЕСТИРОВАНИЕ

3.1. Тестирование

Результаты тестирования представлены в Таблице 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1	Вставить узлы: 7, 11, 2, 13, 1 Вставить узел: 5 Вставить узел: 14 Найти узел: 111 Найти узел: 5 Удалить узел: 5	Insert: 7 V-----7 Insert: 11 V-----7 R-----11 Insert: 2 V-----7 L-----2 R-----11 Insert: 13 V-----7 L-----2 R-----11 R-----13 Insert: 1 V-----7 L-----2 | L-----1 R-----11 R-----13 Insert: 5 V-----7 L-----2 | L-----1 | R-----5 R-----11 R-----13 Insert: 14 Small left rotate V-----7 L-----2 | L-----1 | R-----5 R-----13 L-----11 R-----14	Все верно.

Продолжение таблицы 1

№ п/п	Входные данные	Выходные данные	Комментарии
1		Find: 111 7 Find: 111 13 Find: 111 14 Find: 111 Узел не найден! Find: 5 7 Find: 5 2 Find: 5 key: 5, height: 1 Delete: 5 V-----7 L-----2 | L-----1 R-----13 L-----11 R-----14	Все верно.

3.2. Тестирование особых случаев

Тестирование представлено в файле test.py

ЗАКЛЮЧЕНИЕ

Таким образом, была реализована структура данных АВЛ-дерево класс AVL. АВЛ-дерево состоит из узлов класса Node. Реализованы методы вставки, поиска и удаления узлов. Методы и методы поворотов визуализируются в консоли. Так же визуализируется АВЛ-дерево.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Алгоритмы: теория и практика. Структуры данных. Деревья поиска. Stepik. https://stepik.org/lesson/41563/step/1?unit=20012

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл avl.py

```
class Node:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None
        self.height = 0

    def __str__(self):
        return "key: {}, height: {}\n".format(self.key, self.height)


class AVL:
    def __init__(self):
        self.root = None

    def height(self, node):
        if node is not None:
            return node.height
        else:
            return 0

    def insert(self, key):
        if not self.root:
            print("\nInsert: ", key)
            self.root = Node(key)
            self.print_avl(self.root, "", True, 0)
        else:
            print("\nInsert: ", key)
            self.root = self.insert_rec(key, self.root)
            self.print_avl(self.root, "", True, 0)

    def insert_rec(self, key, node):
        if not node:
            node = Node(key)
        elif key < node.key:
            node.left = self.insert_rec(key, node.left)
        else:
            node.right = self.insert_rec(key, node.right)

        if self.height(node.right) - self.height(node.left) == 2:
            if key < node.right.key:
                node = self.big_l_rotate(node)
            else:
                node = self.small_l_rotate(node)

        if self.height(node.right) - self.height(node.left) == -2:
            if key > node.left.key:
                node = self.big_r_rotate(node)
            else:
                node = self.small_r_rotate(node)
        node.height = max(self.height(node.left),
self.height(node.right)) + 1
```

```
        return node

    def delete(self, key):
        print("\nDelete: ", key)
        self.root = self.delete_rec(key, self.root)
        self.print_avl(self.root, "", True, 0)

    def delete_rec(self, key, node):
        if not node:
            return node
        elif key < node.key:
            node.left = self.delete_rec(key, node.left)
        elif key > node.key:
            node.right = self.delete_rec(key, 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
            else:
                temp = self.minimum(node.right)
                node.key = temp.key
                node.right = self.delete_rec(temp.key, node.right)

        if self.height(node.right) - self.height(node.left) == 2:
            if self.height(node.right.left) -
self.height(node.right.right) > 0:
                node = self.big_l_rotate(node)
            else:
                node = self.small_l_rotate(node)

        if self.height(node.right) - self.height(node.left) == -2:
            if self.height(node.left.left) - self.height(node.left.right)
< 0:
                node = self.big_r_rotate(node)
            else:
                node = self.small_r_rotate(node)
        node.height = max(self.height(node.left),
self.height(node.right)) + 1
        return node

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

    def find(self, key, node, a):
        print("\nFind: ", key)
        if node is None:
            a.append("Узел не найден!")
        else:
            if key < node.key:
                print(node.key)
```

```
                self.find(key, node.left, a)
            elif key > node.key:
                print(node.key)
                self.find(key, node.right, a)
            else:
                a.append(node)
        return a

    def small_l_rotate(self, node):
        print("\nSmall left rotate")
        b = node.right
        c = b.left
        node.right = c
        b.left = node
        b.height = max(self.height(b.left), self.height(b.right)) + 1
        node.height = max(self.height(node.left),
self.height(node.right)) + 1
        return b

    def big_l_rotate(self, node):
        print("\nBig left rotate")
        node.right = self.small_r_rotate(node.right)
        self.print_avl(self.root, "", True, 0)
        return self.small_l_rotate(node)

    def small_r_rotate(self, node):
        print("\nSmall right rotate")
        b = node.left
        c = b.right
        node.left = c
        b.right = node
        b.height = max(self.height(b.left), self.height(b.right)) + 1
        node.height = max(self.height(node.left),
self.height(node.right)) + 1
        return b

    def big_r_rotate(self, node):
        print("\nBig right rotate")
        node.left = self.small_l_rotate(node.left)
        self.print_avl(self.root, "", True, 0)
        return self.small_r_rotate(node)

    def print_avl(self, current, indention, child_r, check):
        if current != None:
            print(indention, end='')
            if not child_r:
                print("L-----", end='')
                indention += "|     "
            else:
                if (current.key == self.root.key and check == 0):
                    print("V-----", end='')
                    check = 1
                else:
                    print("R-----", end='')
                indention += "      "
            print(current.key)
            self.print_avl(current.left, indention, False, check)
```

```
            self.print_avl(current.right, indention, True, check)

    def root(self):
        return self.root
```

Файл main.py

```
from avl import AVL


if __name__ == '__main__':
    tree_avl = AVL()
    nums = [7, 11, 2, 13, 1]
    for num in nums:
        tree_avl.insert(num)
    tree_avl.insert(5)
    tree_avl.insert(14)
    root1 = tree_avl.root
    a = []
    a = tree_avl.find(111, root1, a)
    for i in a:
        print(i)
    a = []
    a = tree_avl.find(5, root1, a)
    for i in a:
        print(i)
    tree_avl.delete(5)
```

Файл test.py

```
from avl import AVL


def top(root, a):
    if root != None:
        a.append(root.key)
        top(root.left, a)
        top(root.right, a)
    return a


def test_bigrightrotate():
    print("\n1")
    tree = AVL()
    nums = [7, 11, 2, 13, 1, 3]
    for num in nums:
        tree.insert(num)
    tree.insert(5)
    tree.delete(13)
    v = top(tree.root, [])
    assert v == [3, 2, 1, 7, 5, 11]


def test_bigleftrotate():
    print("\n2")
    tree = AVL()
    nums = [1, 3, 2]
    for num in nums:
        tree.insert(num)
```

```
    v = top(tree.root, [])
    assert v == [2, 1, 3]


def test_delete_all():
    print("\n3")
    tree = AVL()
    nums = [18, 19, 35, 17, 11, 27, 8]
    for num in nums:
        tree.insert(num)
    tree.delete(8)
    tree.delete(27)
    tree.delete(11)
    tree.delete(17)
    tree.delete(35)
    tree.delete(19)
    tree.delete(18)
    v = top(tree.root, [])
    assert v == []


def test_find_all_and_absent():
    print("\n4")
    tree = AVL()
    nums = [5, 7, 8, 1, 15, 17]
    for num in nums:
        tree.insert(num)
    a = []
    tree.find(15, tree.root, a)
    tree.find(1, tree.root, a)
    tree.find(101, tree.root, a)
    tree.find(7, tree.root, a)
    tree.find(17, tree.root, a)
    tree.find(5, tree.root, a)
    tree.find(8, tree.root, a)
    v = []
    for i in a:
        v.append(str(i))
    assert v == ['key: 15, height: 2\n', 'key: 1, height: 1\n', 'Узел не
найден!', 'key: 7, height: 3\n', 'key: 17, height: 1\n', 'key: 5, height:
2\n', 'key: 8, height: 1\n']


if __name__ == '__main__':
    test_bigrightrotate()
    test_bigleftrotate()
    test_delete_all()
    test_find_all_and_absent()
```