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

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: Исследование структур: АВЛ-дерево и Хеш-таблица (двойное хеширование)

Студент гр. 1303 _____ Кузнецов Н.А.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2022

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Кузнецов Н.А.

Группа 1303

Тема работы : АВЛ-дерево vs Хеш-таблица (двойное хеширование). Исследование

Исходные данные:
"Исследование" - реализация требуемых структур данных/алгоритмов; генерация входных данных (вид входных данных определяется студентом); использование входных данных для измерения количественных характеристик структур данных, алгоритмов, действий; сравнение экспериментальных результатов с теоретическими. Вывод промежуточных данных не является строго обязательным, но должна быть возможность убедиться в корректности алгоритмов.

Предполагаемый объем пояснительной записки:
Не менее 15 страниц.

Дата выдачи задания: 25.10.2022

Дата сдачи реферата: 23.12.2022

Дата защиты реферата: 24.12.2022

Студент _____ Кузнецов Н.А.

Преподаватель _____ Иванов Д.В.

АННОТАЦИЯ

В данной курсовой работе реализованы две структуры данных – АВЛ - дерево и хеш-таблица, разрешающая коллизии открытой адресации, при этом используя двойное хеширование. Исследовано время, за которое в этих структурах производится вставка, поиск и удаление элемента.

Экспериментальные данные сравнены с теоретическими на основе графиков.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы является исследование двух структур данных, это Хеш-таблица с открытой адресацией и АВЛ — дерево. Под исследованием понимается реализация требуемых структур данных/алгоритмов; генерация входных данных (вид входных данных определяется студентом); использование входных данных для измерения количественных характеристик структур данных, алгоритмов, действий; сравнение экспериментальных результатов с теоретическими.

2. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1 Сведения о АВЛ-дереве

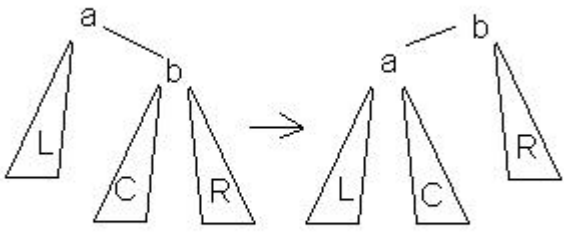
АВЛ-дерево - это сбалансированное двоичное дерево поиска, в котором поддерживается следующее свойство: для каждой его вершины высота её двух поддеревьев различается не более чем на 1. Данный факт дает возможность ускорить поиск, вставку и удаление элементов при работе с данным видом двоичных деревьев поиска.

При изменении дерева (с помощью операций вставки или удаления пары), может быть нарушено основное свойство АВЛ-дерева . Поэтому необходимо после этих действий восстанавливать свойство дерева, это называется балансировкой дерева.

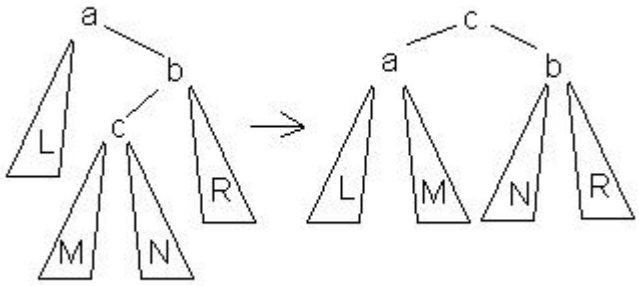
Относительно АВЛ-дерева балансировкой вершины называется операция, которая в случае разницы высот левого и правого поддеревьев = 2, изменяет связи предок-потомок в поддереве данной вершины так, что разница становится <= 1, иначе ничего не меняет. Указанный результат получается вращениями поддерева данной вершины.

Используются 4 типа вращений:

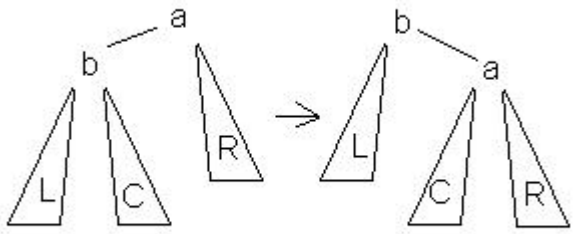
1. Малое левое вращение. Данное вращение используется тогда, когда разница высот a-поддерева и b-поддерева равна 2 и высота C <= высота R



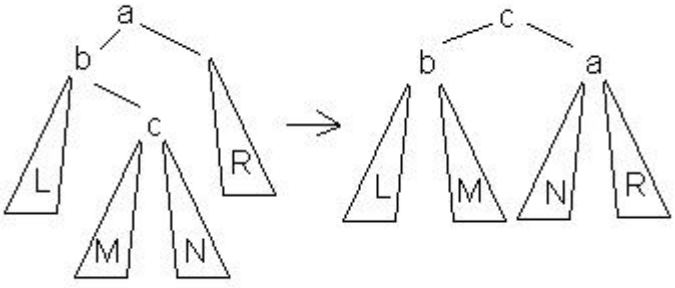
2. Большое левое вращение. Данное вращение используется тогда, когда разница высот a-поддерева и b-поддерева равна 2 и высота c-поддерева > высота R.



3. Малое правое вращение. Данное вращение используется тогда, когда разница высот a-поддерева и b-поддерева равна 2 и высота C <= высота L.



4. Большое правое вращение. Данное вращение используется тогда, когда разница высот a-поддерева и b-поддерева равна 2 и высота c-поддерева > высота L.



2.2 Сведения о Хеш-таблице

Хеш-таблица - структура данных, которая позволяет хранить пары (ключ, значение) и осуществлять доступ к элементу по ключу. Где ключ — хеш-значение, которое используется для индексации данных в таблице, а значение — это данные, которые с этим ключом связаны.

Ключ получается в результате обработки данных в некоторой хэш - функции. Этот процесс называется хешированием.

Когда хеш-значение совпадает для разных ключей возникает конфликт, называемый коллизией хеш-таблицы.

Методы борьбы с коллизиями:

- Метод цепочек.

- Метод открытой адресации: линейное и квадратичное исследование, двойное хеширование.

В данной курсовой работе реализовано *двойное* хеш*ирование.*

При двойном хешировании используются две независимые хеш-функции h1(k) и h2(k)

Операции вставки, удаления и поиска в лучшем случае выполняются за O(1), в худшем — за O(m), что не отличается от обычного линейного разрешения коллизий. Однако в среднем, при грамотном выборе хеш-функций, двойное хеширование будет выдавать лучшие результаты, за счёт того, что вероятность совпадения значений сразу двух независимых хеш-функций ниже, чем одной.

3. РЕАЛИЗАЦИЯ

3.1 Реализация АВЛ-дерева.

Реализован класс *Node*, который будет узловым элементов в дереве. Каждый узел хранит ключ, значение, ссылку на правого и левого ребенка и текущую высоту поддерева, которая считается в ребрах.

Реализован класс *AVLTree,* в котором создается авл-дерево.

Методы:

insert(self, key, value) — вставка ключа и значения в нужное место в дереве.

find(key) - метод поиска узла по ключу.

remove(key) - метод удаления узла по ключу.

findMax() - поиск максимального ключа в дереве

f*indMin()* - поиск минимального ключа в дереве

Также реализованы 4 поворота для балансировки дерева :

small_right_rotate(node) - малый правый поворот

big_right_rotate(node) — большой правый поворот

small_left_rotate(node)— малый левый поворот

big_left_rotate(node) - большой левый поворот

get_height(node) — подсчет высоты для вершины в ребрах

3.2 Реализация хеш-таблицы

class HashTable - реализация хеш-таблицы, имеющая следующие поля: размер таблицы - *size*, количество элементов - *fulness* , массив элементов типа класс *Pair* - *data* и коэфициент заполненности - *fullness*_koef.

Так же класс обладает набором методов.

get_data - получение данных таблицы

print - печать таблицей данных из *data*, с использованием библиотеки *tabulate*

get_size - получение размера таблицы

insert - вставка элемента в таблицу

find - поиск элемента по ключу

delete - удаление элемента по ключу

Некоторые описанные выше *public* методы используют следующие privat методы

expand_table - расширение таблицы в случае ее заполнения

hash_func_1 - первая хеш-функция

hash_func_2 - вторая хеш-функция

double_hash - Функция объеденяющая *hash_func_1* и *hash_func_2* для двойного хеширования

find_place - нахождения места в *data* для заданного ключа *key*

find_elem - нахождения элемента по ключу, если элемент не найден, то возвращается -1

4. СРАВНЕНИЕ ЭКСПЕРИМЕНТАЛЬНЫХ ДАННЫХ С ТЕОРЕТИЧЕСКИМИ

4.1 АВЛ-дерево

Теоретические сведения

Сложность выполнения операций зависит от высоты дерева

Операции	Лучший случай	Средний случай	Худший случай
Вставка	O(1)	O(logn)	O(logn)
Поиск	O(1)	O(logn)	O(logn)
Удаление	O(1)	O(logn)	O(logn)

Таблица 1. Сложность действий, АВЛ-дерево.

Экспериментальные данные

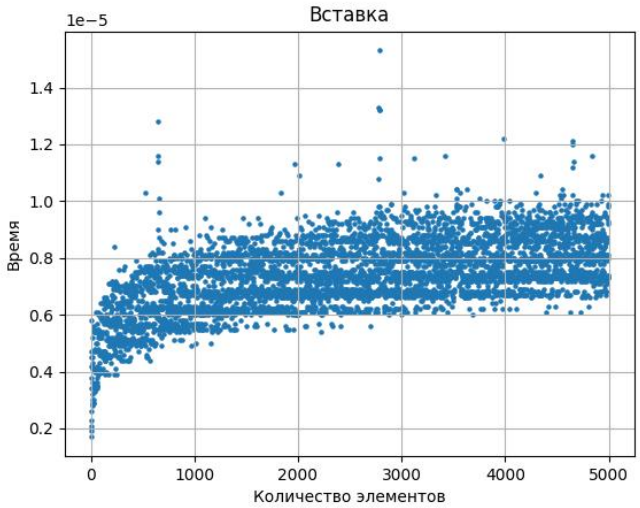


Рисунок 1. - Сложность вставки в АВЛ-дерево.

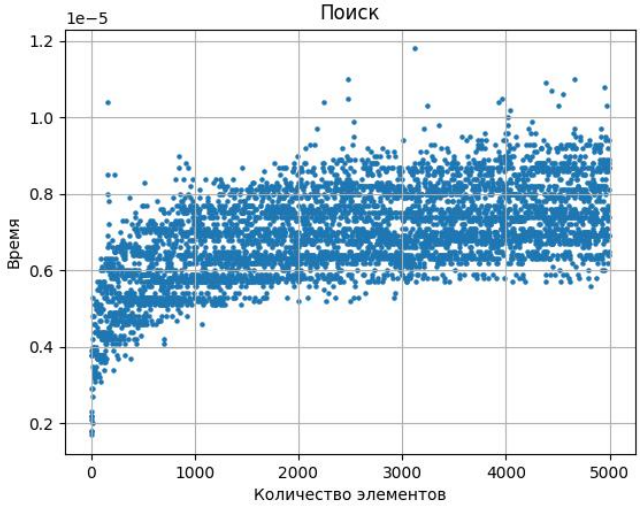


Рисунок 2. - Сложность поиска в АВЛ-дереве.

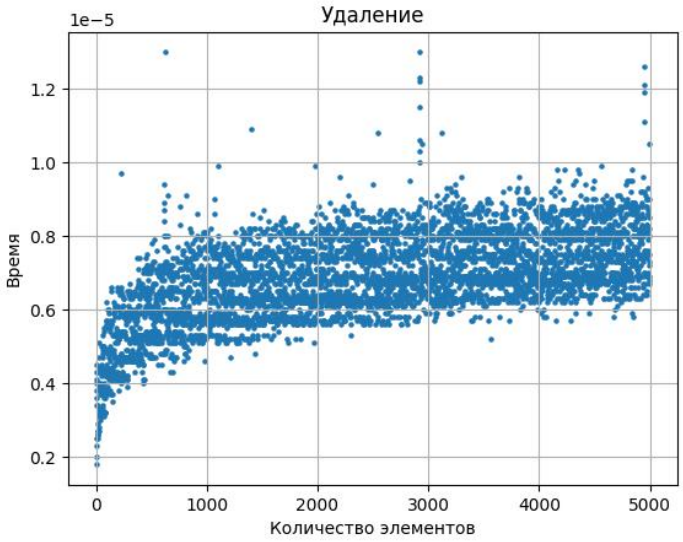


Рисунок 3. - Сложность удаления из АВЛ-дерева

Сложность операций совпадает с теоретическими данными.

4.2 Хеш-таблица

Теоретические сведения

Сложность выполнения операций зависит от количества коллизий (за которые отвечает коэффициент a) или от количества элементов в таблице.

Операции	Лучший случай	Средний случай	Худший случай
Вставка	O(1)	O(a)	O(n)
Поиск	O(1)	O(a)	O(n)
Удаление	O(1)	O(a)	O(n)

Таблица 2. Сложность действий, хеш таблица.

Экспериментальные данные

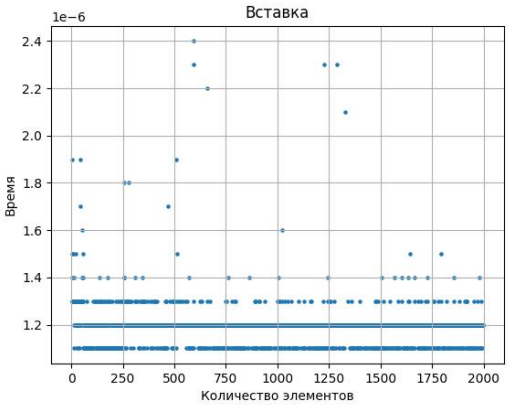


Рисунок 4. - Сложность вставки в хеш-таблицу. Лучший случай.

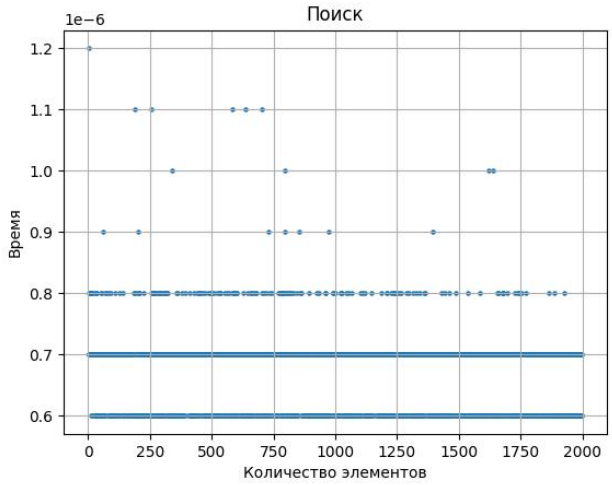


Рисунок 5. - Сложность поиска в хеш-таблице. Лучший случа.

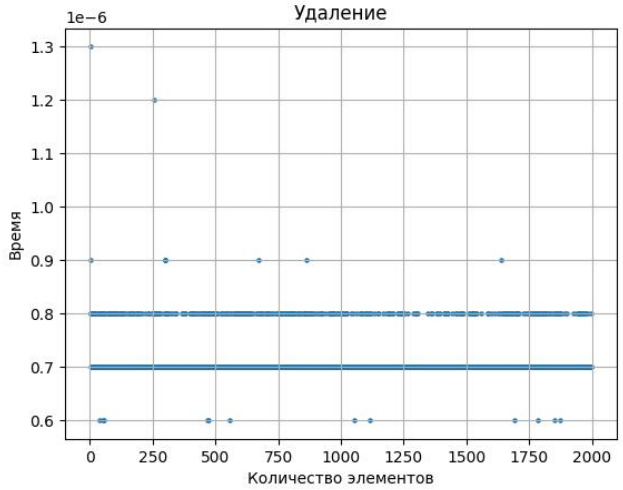


Рисунок 6. - Сложность удаления в хеш-таблице. Лучший случай.

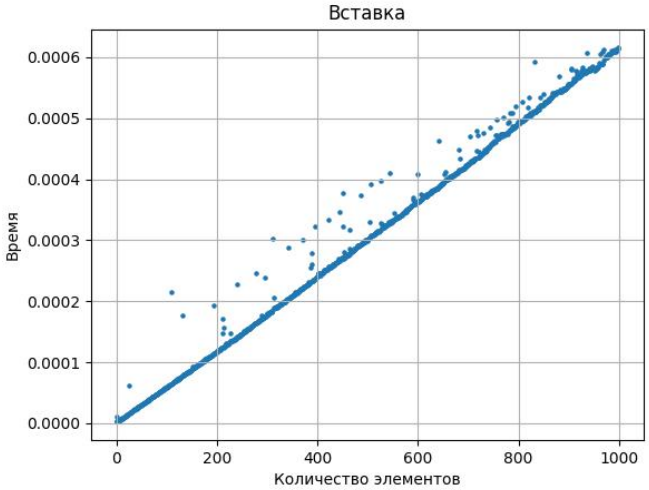


Рисунок 7. - Сложность вставки в хеш-таблицу. Худший случай.

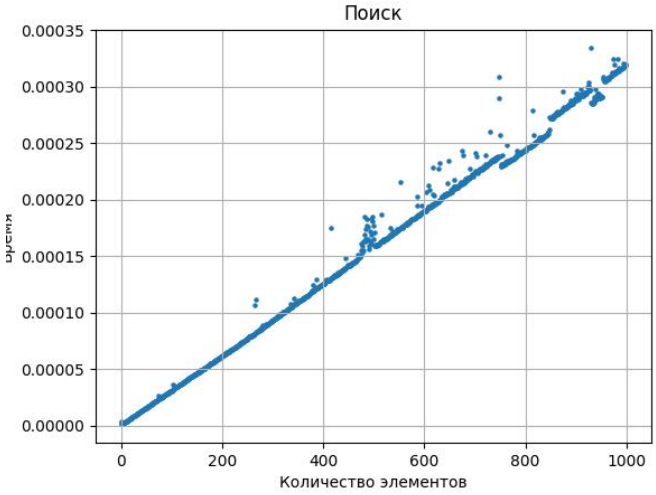


Рисунок 8. - Сложность поиска в хеш-таблице. Худший случай.

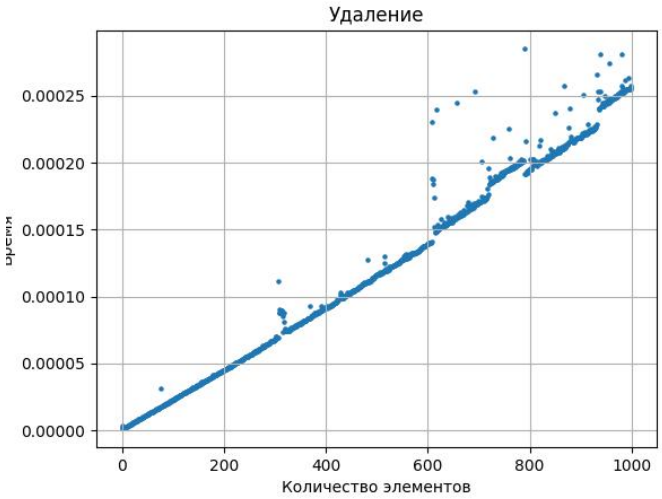


Рисунок 9. - Сложность удаления в хеш-таблице. Худший случай.

Сложность операций совпадает с теоретическими данными.

5. ЗАКЛЮЧЕНИЕ

В ходе курсовой работы были реализованы такие структуры данных, как АВЛ-дерево и хеш-таблица с открытой адресацией на языке Python 3.8.

Были исследованы теоретические положения для данных структур, была протестирована корректность работы этих реализаций, исследовано время работы операций поиска, вставки и удаления для каждой из структур данных.

Было проведено сравнение времени работы данных операций для двух структур данных: исследование подтвердило теоретические оценки на требуемое время для операций.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. "Алгоритмы: теория и практика. https://stepik.org/course/1547/syllabus
2. https://ru.wikipedia.org/wiki/АВЛ-дерево
3. https://ru.wikipedia.org/wiki/Хеш-таблица.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: avl_tree.py

```
from modules.node import Node


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

    def insert(self, key, value):
        new_node = Node(key, value)
        if not self.root:
            self.root = new_node
        else:
            self.root = self.insert_rec(new_node, self.root)

    def insert_rec(self, new_node, node):
        if not node:
            return new_node

        if new_node.key < node.key:
            node.left = self.insert_rec(new_node, node.left)
            if abs(self.get_height(node.left) - self.get_height(node.right))
== 2:
                if new_node.key > node.left.key:
                    node = self.big_right_rotate(node)
                else:  # key < b
                    node = self.small_right_rotate(node)
        elif new_node.key > node.key:
            node.right = self.insert_rec(new_node, node.right)
            if abs(self.get_height(node.right) - self.get_height(node.left))
== 2:
                if new_node.key < node.right.key:
                    node = self.big_left_rotate(node)
                else:  # key > b
                    node = self.small_left_rotate(node)

        elif new_node.key == node.key:
            raise ValueError("Ключ со значением {} уже
присутствует".format(new_node.key))

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

    def remove(self, elem):
        self.root = self.remove_rec(elem, self.root)

    def remove_rec(self, key, node):
        if node is None:
            raise ValueError("Ключа {} нет в дереве".format(key))
```

```
        if key < node.key:
            node.left = self.remove_rec(key, node.left)
            if abs(self.get_height(node.left) - self.get_height(node.right))
== 2:
                if node.right is None:
                    node = self.big_left_rotate(node)
                elif node.right is None:
                    node = self.small_left_rotate(node)
                elif key > node.left.key:
                    node = self.big_right_rotate(node)
                else:
                    node = self.small_right_rotate(node)
        elif key > node.key:
            node.right = self.remove_rec(key, node.right)
            if abs(self.get_height(node.right) - self.get_height(node.left))
== 2:
                if node.right is None:
                    node = self.big_right_rotate(node)
                elif node.left is None:
                    node = self.small_right_rotate(node)
                elif key < node.right.key:
                    node = self.big_left_rotate(node)
                else:
                    node = self.small_left_rotate(node)

        elif node.key == key:  # можно удалить из поддерева большей высоты,
чтобы было меньше балансирования
            if node.left and node.right:
                change_elem = self.findMax_rec(node.left)
                node.key = change_elem.key
                node.left = self.remove_rec(node.key, node.left)
            else:
                if node.right:
                    node = node.right
                    return node
                elif node.left:
                    node = node.left
                    return node
                else:
                    node = None
                    return node
        node.height = max(self.get_height(node.left),
self.get_height(node.right)) + 1
        return node

    def findMax(self):
        if self.root is None:
            return None
        else:
            return self.findMax_rec(self.root)

    def findMax_rec(self, node):
        if node.right:
            return self.findMax_rec(node.right)
        else:
            return node
```

```
    def findMin(self):
        if self.root is None:
            return None
        else:
            return self.findMin_rec(self.root)

    def findMin_rec(self, node):
        if node.left:
            return self.findMin_rec(node.left)
        else:
            return node

    def find(self, key):
        return self.find_rec(key, self.root)

    def find_rec(self, key, node):
        if node is None:
            return None
        if key < node.key:
            return self.find_rec(key, node.left)
        if key > node.key:
            return self.find_rec(key, node.right)
        return node.value

    def get_height(self, node):
        if not node:
            return -1
        else:
            return node.height

    def small_left_rotate(self, node_a):
        node_b = node_a.right
        Lb = node_b.left
        node_a.right = Lb
        node_b.left = node_a

        node_a.height = max(self.get_height(node_a.right),
self.get_height(node_a.left)) + 1
        node_b.height = max(self.get_height(node_b.right),
self.get_height(node_b.left)) + 1

        return node_b

    def small_right_rotate(self, node_a):
        node_b = node_a.left
        Rb = node_b.right
        node_a.left = Rb
        node_b.right = node_a

        node_a.height = max(self.get_height(node_a.right),
self.get_height(node_a.left)) + 1
        node_b.height = max(self.get_height(node_b.right),
self.get_height(node_b.left)) + 1

        return node_b
```

```
    def big_left_rotate(self, node_a):
        node_a.right = self.small_right_rotate(node_a.right)
        return self.small_left_rotate(node_a)

    def big_right_rotate(self, node_a):
        node_a.left = self.small_left_rotate(node_a.left)
        return self.small_right_rotate(node_a)
```

Название файла: hash_table.py

```
from modules.pair import Pair
import tabulate


class HashTable:

    def __init__(self):
        self.__size = 8
        self.__fullness = 0
        self.__fullness_koef = 1.7
        self.__data = []
        for i in range(self.__size):
            self.__data.append(Pair())

    def get_data(self):
        return self.__data

    def __str__(self):
        ret_string = ''
        for i in self.__data:
            ret_string += str(i) + '\n'
        return ret_string

    def print(self):
        head = ['key', 'value']
        arr = []
        for i in range(self.__size):
            arr.append([str(self.__data[i].first),
str(self.__data[i].second)])
        print(tabulate.tabulate(arr, headers=head, tablefmt="outline"))

    def get_size(self):
        return self.__size

    def __expand_table(self):
        for i in range(self.__size, self.__size*2):
            self.__data.append(Pair())
        self.__size *= 2

    def __hash_func_1(self, x):
        # return 0
        return x % self.__size

    def __hash_func_2(self, x):
        # return 1
        return x % (self.__size - 1) + 1
```

```
    def __double_hash(self, x, i):
        return (self.__hash_func_1(x) + i * self.__hash_func_2(x)) %
self.__size

    def __find_place(self, key):
        find_ind = 0
        ind = self.__double_hash(key, find_ind)

        found = False
        while not found:
            if self.__data[ind].is_empty():
                found = True
            else:
                find_ind += 1
                ind = self.__double_hash(key, find_ind)
        return ind

    def insert(self, key, value):
        ind = self.__find_elem(key)
        if ind == -1:
            ind = self.__find_place(key)

        self.__data[ind].first = key
        self.__data[ind].second = value

        self.__fullness += 1
        if self.__size < self.__fullness * self.__fullness_koef:
            self.__expand_table()

    def __find_elem(self, key):
        find_ind = 0
        ind = self.__double_hash(key, find_ind)

        found = False
        while not found:
            if not self.__data[ind].used and self.__data[ind].is_empty():
                ind = -1
                found = True
            elif self.__data[ind].first == key:
                found = True
            else:
                find_ind += 1
                ind = self.__double_hash(key, find_ind)
        return ind

    def find(self, key):
        ind = self.__find_elem(key)
        if ind == -1:
            return 'Not Found'
        return self.__data[ind].second

    def delete(self, key):
        ind = self.__find_elem(key)
        if ind == -1:
            print('Cannot be deleted')
        else:
```

```
            self.__data[ind].first = None
            self.__data[ind].second = None
            self.__data[ind].used = True
```

Название файла: node.py

```
class Node:
    def __init__(self, key, value):
        self.key = key
        self.value = value
        self.left = None
        self.right = None
        self.height = 0  # высота считается в ребрах

    def __str__(self):
        left = self.left.key if self.left else None
        right = self.right.key if self.right else None
        return 'key: {}, left: {}, right: {}'.format(self.key, left, right)
```

Название файла: pair.py

```
class Pair:

    def __init__(self, first=None, second=None):
        self.first = first
        self.second = first
        self.used = False

    def is_empty(self):
        if self.first is None or self.second is None:
            return True
        return False

    def __str__(self):
        if self.is_empty():
            return '=)'
        return f'({self.first} -> {self.second})'
```

Название файла: commands.py

```
from enum import Enum


class Commands(Enum):
    INS = 1
    FIND = 2
     DEL = 3
```

Название файла: test_avl_tree.py

```
from modules.avl_tree import AVLTree
from modules.commands import Commands
import time
import matplotlib.pyplot as plt
import random


class TestAVL:
    def __init__(self, kol, cmd):
        self.__kol = kol
        self.__cmd = cmd

    def start(self):
```

```
        arr = list(range(self.__kol))
        random.shuffle(arr)
        plt.xlabel('Количество элементов')
        plt.ylabel('Время')
        plt.grid()
        y = []
        x = []
        if self.__cmd == Commands.INS:
            plt.title("Вставка")
            avl = AVLTree()
            for i in range(self.__kol):
                t = time.perf_counter()
                avl.insert(arr[i], i)
                y.append(time.perf_counter() - t)
                x.append(i)
            plt.scatter(x, y, s=5)
            plt.show()
            return

        avl = AVLTree()
        for i in range(self.__kol):
            avl.insert(arr[i], i)

        if self.__cmd == Commands.FIND:
            plt.title("Поиск")
            for i in range(self.__kol):
                t = time.perf_counter()
                avl.find(i)
                y.append(time.perf_counter() - t)
                x.append(i)
            plt.scatter(x, y, s=5)
            plt.show()
            return

        if self.__cmd == Commands.DEL:
            plt.title("Удаление")
            for i in range(self.__kol):
                t = time.perf_counter()
                avl.remove(i)
                y.append(time.perf_counter() - t)
                x.append(i)
            plt.scatter(x, y, s=5)
            plt.show()
            return
```

Название файла: test_hash_table.py

```
from modules.hash_table import HashTable
from modules.commands import Commands
import time
import matplotlib.pyplot as plt
import random


class TestHashTable:
    def __init__(self, kol, cmd):
        self.__kol = kol
        self.__cmd = cmd
```

```
    def start(self):
        plt.xlabel('Количество элементов')
        plt.ylabel('Время')
        plt.grid()
        y = []
        x = []
        if self.__cmd == Commands.INS:
            plt.title("Вставка")
            arr = list(range(self.__kol))
            random.shuffle(arr)
            h = HashTable()
            for i in range(self.__kol):
                t = time.perf_counter()
                h.insert(i, i)
                y.append(time.perf_counter() - t)
                x.append(i)
            plt.scatter(x, y, s=5)
            plt.show()
            return

        arr = list(range(self.__kol))
        random.shuffle(arr)
        h = HashTable()
        for i in range(self.__kol):
            h.insert(arr[i], i)

        if self.__cmd == Commands.FIND:
            plt.title("Поиск")
            for i in range(self.__kol):
                t = time.perf_counter()
                h.find(i)
                y.append(time.perf_counter() - t)
                x.append(i)
            plt.scatter(x, y, s=5)
            plt.show()
            return

        if self.__cmd == Commands.DEL:
            plt.title("Удаление")
            for i in range(self.__kol):
                t = time.perf_counter()
                h.delete(i)
                y.append(time.perf_counter() - t)
                x.append(i)
            plt.scatter(x, y, s=5)
            plt.show()
            return
```

Название файла: main.py

```
from test_avl_tree import TestAVL
from test_hash_table import TestHashTable
from modules.commands import Commands
from modules.hash_table import HashTable
from modules.avl_tree import AVLTree


def breadth_first_search(root, dot):
    queue = [root]
```

```
    dot.node(str(root.key))
    while queue:
        tmp_queue = []
        for element in queue:
            if element.left:
                dot.node(str(element.left.key))
                dot.edge(str(element.key), str(element.left.key))
                tmp_queue.append(element.left)
            if element.right:
                dot.node(str(element.right.key))
                dot.edge(str(element.key), str(element.right.key))
                tmp_queue.append(element.right)
        queue = tmp_queue


def main():
    # FOR EXAMPLE
    h = HashTable()
    h.insert(10, "lol")
    h.print()
    test = TestHashTable(1000, Commands.INS)
    test.start()
    test = TestAVL(500, Commands.INS)
    test.start()


if __name__ == "__main__":
    main()
```