МИНОБРНАУКИ РОССИИ
Санкт-Петербургский государственный
электротехнический университет «ЛЭТИ»
им. В.И. Ульянова (Ленина)
Кафедра САПР

ОТЧЁТ
ЛАБОРАТОРНАЯ РАБОТА №1
по дисциплине «Алгоритмы и структуры данных»
Тема: Ассоциативный массив

Студент гр. 9892 _____ Лескин К.А.

Преподаватель _____ Тутуева А.В.

Санкт-Петербург
2022

1 Постановка задачи

Реализовать шаблонный ассоциативный массив (`map`) на основе красно-черного дерева.

Наличие unit-тестов ко всем реализуемым методам является обязательным требованием.

Список методов:

1. `insert(ключ, значение)` — добавление элемента с ключом и значением
2. `remove(ключ)` — удаление элемента по ключу
3. `find(ключ)` — поиск элемента по ключу
4. `clear` — очищение ассоциативного массива
5. `get_keys` — возвращает список ключей
6. `get_values` — возвращает список значений
7. `print` — вывод в консоль

2 Описание реализуемого класса и методов

Задание было выполнено на языке `Python` версии 3.8.

Реализация выполнена в виде трёх классов: `RBTreeColor`, `RBTreeNode`, и `RBTree`.

2.1 RBTreeColor

`RBTreeColor` реализует цвет узла дерева.

Цвет может быть либо красным, либо чёрным.

2.2 RBTreeNode

`RBTreeNode` реализует узел дерева.

Узел дерева имеет ключ, значение, цвет и ссылки на левого, правого потомка и родительский узел. Сравнение узлов происходит через "`python magic methods`"(`__eq__`, `__lt__`, `__gt__`, etc.), где сравниваются хэши ключей.

Так же у узла можно получить:

- "дедушку"(предка второго поколения) через `property` метод `gparent`.

- "дядю"(узел, соседний с родительским) через `property` метод `uncle`.
- "брата"(соседний узел) через `property` метод `bro`.

Узел можно перекрасить методами `color_black()` и `color_red()`.

А так же получить информацию о расположении и цвете: `is_black()`, `is_red()`, `is_left()`, `is_right()`.

2.3 RBTree

`RBTree` реализует само красно-чёрное дерево.

Реализация класса выполнена в "`pythonic way`"с помощью "`python magic methods`". При этом методы требуемые по заданию были сохранены.

Далее идёт список с кратким описанием каждого метода

- `__init__` — Инициализатор.
- `__getitem__` — Получение значения по ключу.
- `__setitem__` — Создание новой пары либо перезапись существующей.
- `__delitem__` — Удаление пары по ключу.
- `__len__` — Получение количества элементов.
- `__str__` — Строковая реперзентация объекта.
- `__iter__` — Получение итератора (по ключам).
- `__bool__` — Булева репрезентация объекта (False, когда пуст, иначе True)
- `height` — Высота дерева
- `get` — Безопасное получение значения по ключу, с возможностью указать значение по умолчанию.
- `items` — Генератор по ключам и значениям одновременно
- `keys` — Получить список ключей.
- `values` — Получить список значений.
- `print_tree` — Вывести объект в виде дерева.

- `get_dot_string` — Получить описание дерева на языке DOT (для визульной репрезентации).
- `insert` — Создание новой пары либо перезапись существующей.
- `remove` — Удаление пары по ключу.
- `find` — Получение значения по ключу.
- `clear` — Очистка дерева.
- `get_keys` — Получить список ключей.
- `get_values` — Получить список значений.
- `print` — Вывести строуовую репрезентацию объекта.
- `_get_max_node` — Получить узел с максимальным значением ключа.
- `_get_min_node` — Получить узел с минимальным значением ключа.
- `_get_height` — Получить высоту поддерева для узла.
- `_fix_insert` — Исправление вставки.
- `_insert_case_1` — Исправление вставки, случай 1.
- `_insert_case_2` — Исправление вставки, случай 2.
- `_insert_case_3` — Исправление вставки, случай 3.
- `_insert_case_4` — Исправление вставки, случай 4.
- `_insert_case_5` — Исправление вставки, случай 5.
- `_replace` — Заменить узел потомком.
- `_delete` — Удалить узел (с ≤ 1 потомком)
- `_fix_delete` — Исправление удаления.
- `_del_case_1` — Исправление удаления, случай 1.
- `_del_case_2` — Исправление удаления, случай 2.
- `_del_case_3` — Исправление удаления, случай 3.

- `_del_case_4` — Исправление удаления, случай 4.
- `_del_case_5` — Исправление удаления, случай 5.
- `_del_case_6` — Исправление удаления, случай 6.
- `_traverse_preorder` — Префиксный обход узлов
- `_traverse_inorder` — Инфиксный обход узлов.
- `_traverse_postorder` — Постфиксный обход узлов.
- `_left_rotate` — Левый поворот.
- `_right_rotate` — Правый поворот.
- `_print_tree` — Рекурсивный метод вывода дерева.
- `_get_node` — Получить узел по ключу.
- `_get_leaf` — Получить узел-родитель по вствляемому ключу.
- `_swap_kv` — Поменять местами данные в узлах.

3 Оценка временной сложности

Оценка временной сложности методов класса RBTree представлена в таблице 1

Таблица 1 – Оценка временной сложности методов класса RBTree

Метод	Оценка временной сложности
`insert`	$O(\log n)$
`remove`	$O(\log n)$
`find`	$O(\log n)$
`clear`	$O(n)$
`get_keys`	$O(n)$
`get_values`	$O(n)$
`print`	$O(n)$

4 Описание unit-тестов

check_tree(tree: RBTree, data: dict)

Проверяет, все ли ключи и значения присутствуют в дереве, верная ли у него длина и высота.

test_init(data)

Проверяет правильность инициализации дерева.

test_insert(data, new_key, existing_key, new_value)

Проверяет правильность вставки нового элемента в дерево.

test_get(data)

Проверяет правильность получения элементов в дереве.

test_get_error()

Проверяет, возникнет ли ожидаемая ошибка при попытке получить значение по несуществующему ключу.

test_delete(data, existing_key)

Проверяет правильность удаления пары из дерева.

test_delete_error(data, missing_key, error)

Проверяет возникнет ли ожидаемая ошибка при попытке удалить значение по несуществующему ключу.

test_clear()

Проверяет правильность очистки дерева.

5 Пример работы

Будем использовать простой декоратор для вывода состояния дерева до и после выполнения примера:

```
examples = []


def example(name):
    def decorator(f):
        def wrapper(t):
            print(f'Example {name}:')
            print(f'Original tree: {t}')

            f(t)

            print('Tree:')
            t.print_tree()
        examples.append(wrapper)
        return wrapper
    return decorator
```

5.1 Поиск

Код примера:

```
@example('find')
def example_find(t):
    for k in t:
        print(f'{k}: {t.find(k)}')
```

Результат выполнения примера (рис. 1):

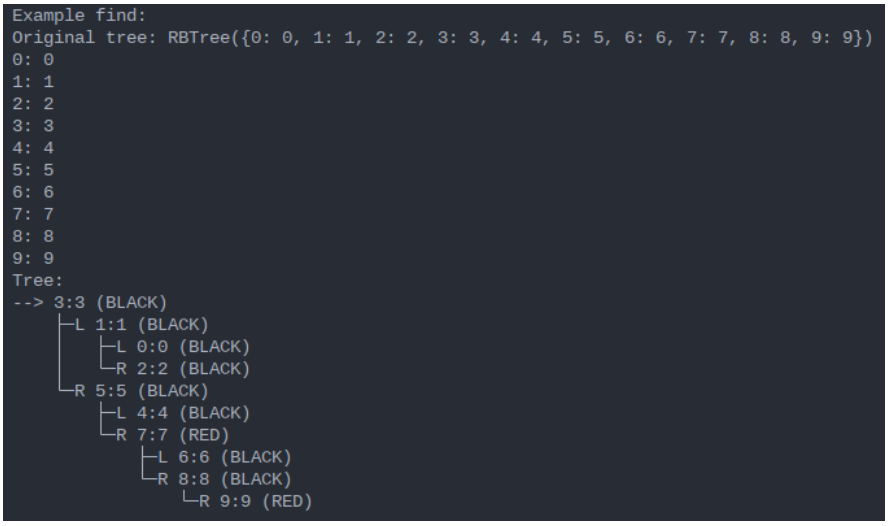

Рис. 1 – Результат выполнения операции "поиск"

5.2 Вставка

Код примера:

```
@example('insert')
def example_insert(t):
    t.insert(1, 'new_value')
    print('Replaced value at key 1 with "new_value":')
    print(t)

    t.insert('new_key', 'another_value')
    print('Inserted value "new_value" with key "new_key":')
    print(t)
```

Результат выполнения примера (рис. 2):

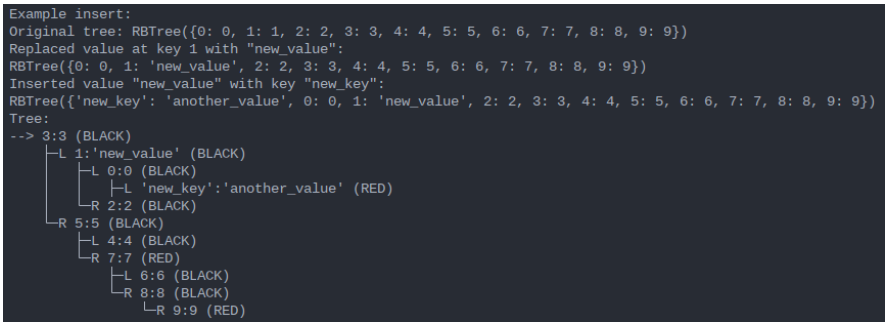

Рис. 2 – Результат выполнения операции "вставка"

5.3 Удаление

Код примера:

```
@example('remove')
def example_remove(t):
    print('Removed key 7:')
    t.remove(7)
    print(t)
```

Результат выполнения примера (рис. 3):

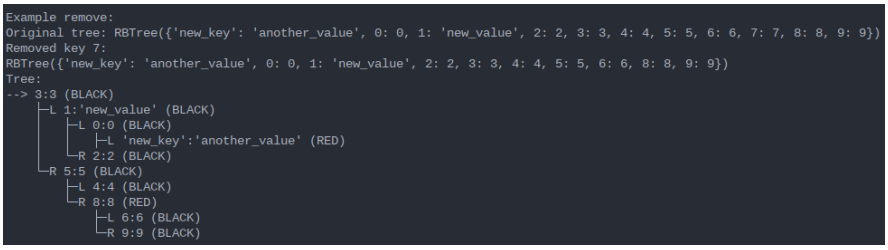

Рис. 3 – Результат выполнения операции "удаление"

5.4 Получение ключей/значений

Код примера:

```
@example('keys and values')
def example_keys_and_values(t):
    print('Keys:')
    print(t.get_keys())

    print('Values:')
    print(t.get_values())
```

Результат выполнения примера (рис. 4):

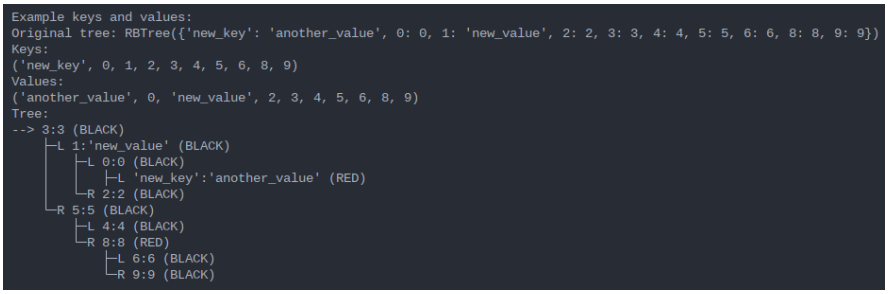

Рис. 4 – Результат выполнения операции "получение ключей/значений"

6 Листинг

Исходный код работы доступен по ссылке:
https://github.com/kira607/1lab-algo-3-2