# МИНОБРНАУКИ РОССИИ

# САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

# ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

# «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

# Кафедра САПР

# Отчет

# по лабораторной работе №1

# по дисциплине «Алгоритмы и структуры данных»

|  |  |  |
| --- | --- | --- |
| Студентка гр. 9301 |  | Служевская А.С. |
| Преподаватель |  | Тутуева А.В. |

Санкт-Петербург

2021

**Постановка задачи. Описание классов и методов**

Задача: реализовать шаблонный массив на основе красно-черного дерева и список методов

1. insert(ключ, значение) — добавление элемента с ключом и значением
2. remove(ключ) — удаление элемента дерева 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

**Оценка временной сложности алгоритмов:**

Таблица 1. Временные затраты

|  |  |
| --- | --- |
| Добавление элемента с ключом и значением / insert(key, value) | O(log(n)) |
| Удаление элемента дерева по ключу / remove(key) | O(log(n)) |
| Поиск элемента по ключу / find(key) | O(log(n)) |
| Очищение ассоциативного массива / clear() | О(n) |
| Возвращает список ключей / get\_keys() | О(n) |
| Возвращает список значений / get\_value() | О(n) |
| Вывод в консоль / print() | О(n) |

**Реализованные Unit-тесты:**

1. insert\_1— добавление одного элемента в дерево
2. insert\_3 —добавление трех элементов в дерево
3. remove — удаление элемента из дерева
4. remove\_error\_double\_remove — удаление уже несуществующего элемента
5. remove\_error\_empty\_tree — удаление из пустого дерева
6. find — поиск элемента
7. find\_error\_out\_of\_range — поиск элемента с несуществующим ключом
8. find\_error\_tree\_empty — поиск в пустом дереве
9. get\_keys — получение массива ключей
10. get\_values — получение массива значений
11. clear — очистка дерева

**Описание реализуемого класса и методов**

1. Node — вспомогательный класс для элементов стэка
2. Stack — вспомогательный класс для реализации дерева. Методы:
   1. get\_size — возвращает размер стэка
   2. isEmpty — проверяет пустоту стэка
   3. clear — очищает стэк
   4. push — добавляет элемент в конец стэка
   5. pop — удаляет элемент из конца стэка
3. RBNode — вспомогательный класс для элементов дерева
4. dft-Iterator — класс для обхода дерева в глубину
   1. next — переходит к следующему элементу
   2. has\_next — проверяет, существует ли следующий элемент
5. RBTree — класс для реализации красно-черного дерева
   1. set\_root — создает корневой элемент дерева
   2. RotateLeft — поворот влево с перестановкой элементов
   3. RotateRight — поворот вправо с перестановкой элементов
   4. coloring — проверяет цвет элемента и перекрашивает при надобности
   5. FixAfterRemoval — «чинит» дерево после того, как из него был удален элемент
   6. is\_empty — проверяет дерево на наличие в нем элементов
   7. insert — вставляет элемент в дерево по ключу
   8. find — ищет элемент по ключу
   9. remove — удаляет элемент по ключу
   10. clear — очищает дерево
   11. get\_size — возвращает размер дерева
   12. get\_keys — возвращает массив с ключами элементов
   13. get\_values — возвращает массив с данными элементов
   14. print — выводит дерево на экран

**Пример работы**

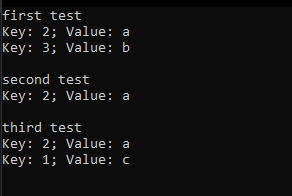
****

Рисунок 1 — Пример работы

**Листинг**

#include <iostream>

#include "RedBlackTree.h"

int main()

{

RBTree<int, char>\* RB = new RBTree<int, char>();

cout << "first test" << endl;

RB->insert(2, 'a');

RB->insert(3, 'b');

RB->print();

cout << "second test" << endl;

RB->remove(3);

RB->print();

cout << "third test" << endl;

RB->insert(1, 'c');

RB->print();

}