**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: Исследование операций вставки и исключения в рандомизированной дерамиде поиска**

Студент гр. 9304 _____ Борисовский В.Ю.

Преподаватель _____ Филатов Ар.Ю.

Санкт-Петербург

2020

**ЗАДАНИЕ**

**НА КУРСОВУЮ РАБОТУ**

Студент Борисовский В.Ю.

Группа 9304

Тема работы: Исследование операций вставки и исключения в рандомизированной дерамиде поиска

Исходные данные:
кратко указываются исходные данные
или основные технические требования

Содержание пояснительной записки:

1. Аннотация
2. Содержание
3. Введение
4. Формальная постановка задачи
5. Описание структур данных и функций, описание алгоритма
6. Тестирование
7. Исследование
8. Заключение
9. Список использованных источников

Предполагаемый объем пояснительной записки:

Не менее 30 страниц.

Дата выдачи задания: 23.11.2020

Дата сдачи реферата: 22.12.2020

Дата защиты реферата: 24.12.2020

| | | |
|---|---|---|
| | _____ | |
| Студент | _____ | Борисовский В.Ю |
| Преподаватель | _____ | Филатов Ар.Ю |

## АННОТАЦИЯ

В данной курсовой работе производится исследование операций вставки и удаление элемнтов в структуре данных рандомизированная дерамида поиска. Исследование производитя с помощью тестов для среднего и худшего случаев работы алгоритма. Результатами исследования являются числовые метрики, на основе которых формируются графики для сравнения с теоретическими оценками.



## SUMMARY

In this course work, studies of insertion and deletion of elements in the data structure of a randomized deramide search are created. The research is performed using tests for the average and worst case of the algorithm. The research results are numerical metrics, on the basis of which comparison graphs are formed using statistical estimates.

## СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

Цель работы: Изучить структуру данных рандомизированная дерамида поиска. Реализация и экспериментальное машинное исследование алгоритмов вставки и удаления в рандомизированную дерамиду поиска.

Задача: Реализовать программу, которая считывает массив элементов рандомизированной дерамиды поиска и изменяет его в соответствии с выбором пользователя, управление осуществляется посредством командной строки.

# 1. ФОРМАЛЬНАЯ ПОСТАНОВКА ЗАДАЧИ

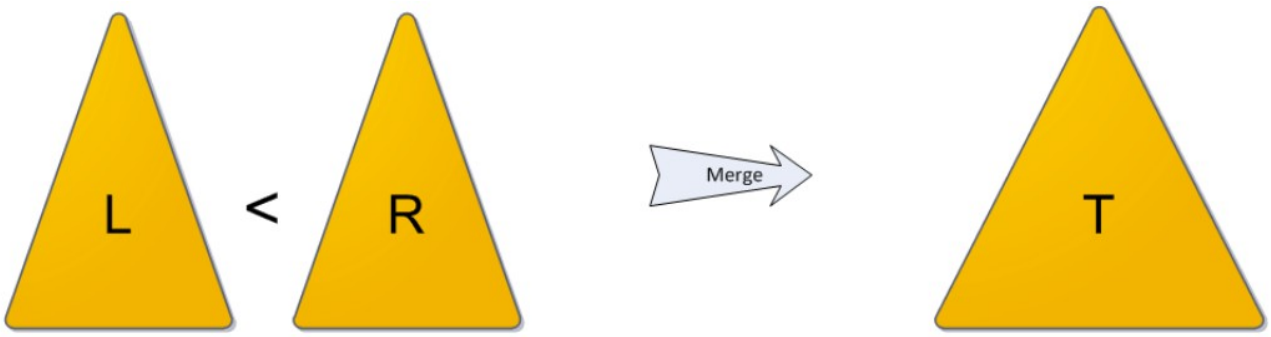
Реализовать структуру данных рандомизированная дерамида поиска и провести исследование работы операций вставки и исключения (в среднем и худшем случае), чтобы проверить теоретическую оценку работы этих операций.

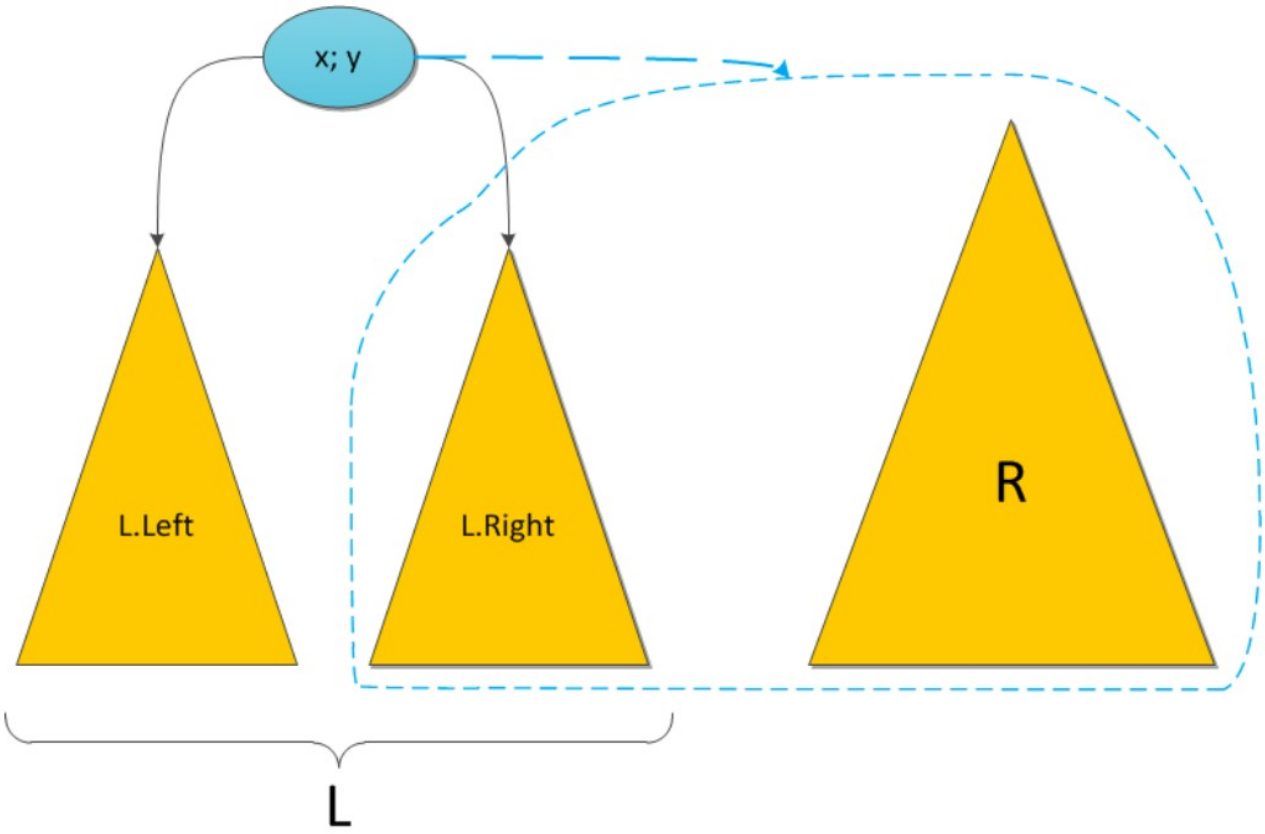
# 2. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

## 2.1. Описание алгоритма

*Алгоритм операции Merge.*

Операция Merge принимает на вход два декартовых дерева **L** и **R**. От нее требуется слить их в одно, тоже корректное, декартово дерево **T**. Следует заметить, что работать операция Merge может не с любыми парами деревьев, а только с теми, у которых все ключи одного дерева ( L ) не превышают ключей второго ( R ). Сравним приоритеты двух корней этих деревьев, очевидно что элемент с большим рпиоритетом станет корнем нового дерева. Таким образом, одно из поддеревьев L/R останется непосредственно левым/правым поддеревом нового дерева, т.к ключи в нем по условию больше/меньше чем в нашем новом корне. А с противположной стороны логично окажется результат рекурсивного применения функции Merge левого/правого поддерева исходного дерева и и соответственно левого/правого поддерева L/R элемента. Поскольку дерамида со случайными приоритетами имеет близкую к логарифмической высоту, а Merge за одну итерацию уменьшет суммарную высоту двух сливаемых деревьев как минимум на единицу, так что общее время работы не превосходит 2H, то сложность примем за O(H), или же O( $\log_2(N)$ ) (т.к высота логарифмическая), где N — число элементов в дереве, а H — высота дерева.

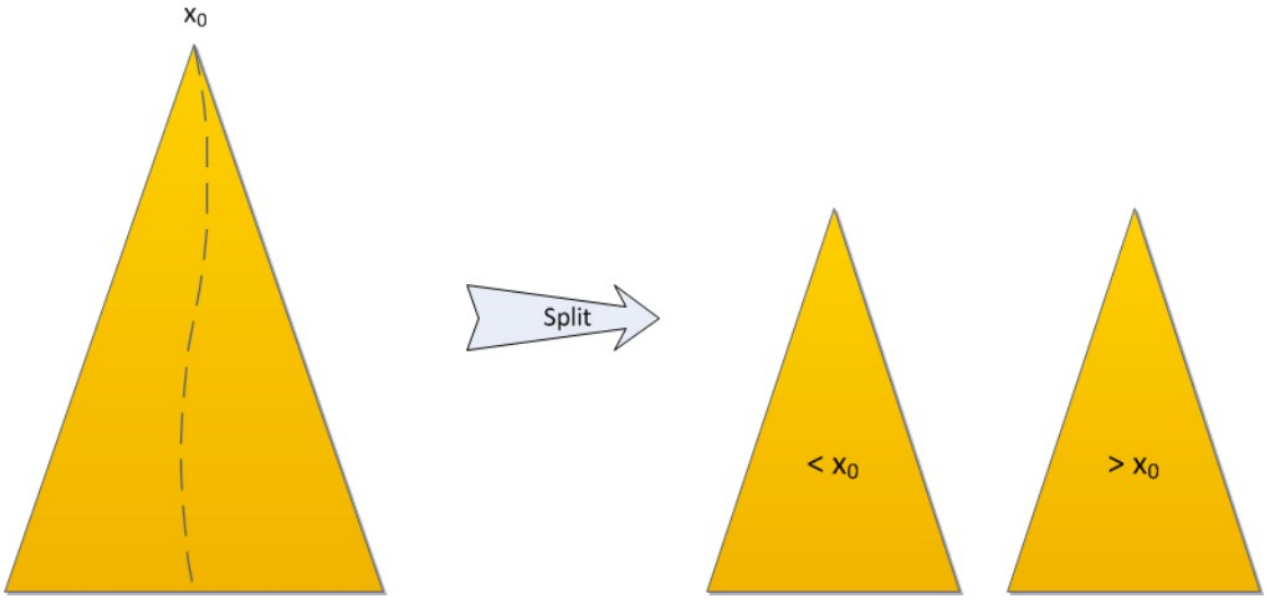


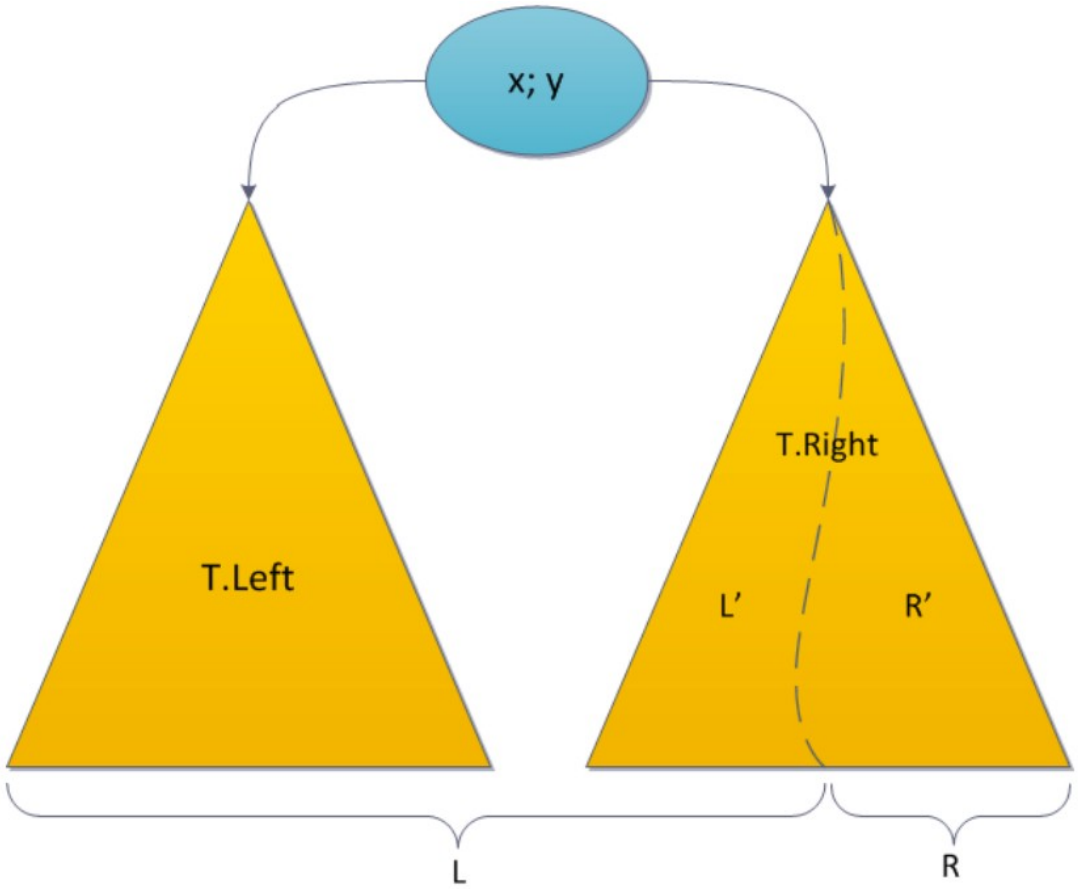



*Алгоритм операции Split*

На вход ей поступает корректное декартово дерево T и некий ключ x0. Задача операции — разделить дерево на два так, чтобы в одном из них ( L ) оказались все элементы исходного дерева с ключами, меньшими x0, а в другом ( R ) — с большими. Никаких особых ограничений на дерево не накладывается.

Корень дерева окажется однозначно либо в R либо в L, в завивисмотси от того, меньше или больше оказался ключ x0, чем ключ корня дерева T. Левое либо правое поддерево дерева T сразу станет L либо R, так как его ключи будут больше/меньше ключа x0 по условию бинарного дерева поиска. А из оставшегося поддерева придется удалить элементы большие/меньшие x0 рекурсивно вызыва функцию Split. Тогда L/R полученное в ходе рекурсивного вызова станет новым поддеревом дерева T, а R/L станет соответственно левым/правим поддеревом уже имеющегося R/L. Сложность по аналогии со Split равняется O( $\log_2(N)$ ), где N — число элементов в дереве.






*Алгоритм операции Insert.*

Сначала спускаемся по дереву (как в обычном бинарном дереве поиска по X), но останавливаемся на первом элементе, в котором значение приоритета оказалось меньше Y. Мы нашли позицию, куда будем вставлять наш элемент. Теперь вызываем Split (X) от найденного элемента (от элемента вместе со всем его поддеревом), и возвращаемые ею L и R записываем в качестве левого и правого сына добавляемого элемента. Т.к при реализации используется операция Split, то сложность будет равняться O( $\log_2(N)$ ), где N — число элементов в дереве.

*Алгоритм операции Erase.*

Спускаемся по дереву (как в обычном бинарном дереве поиска по X), ища удаляемый элемент. Найдя элемент, мы просто вызываем Merge от его левого и правого сыновей, и возвращаемое ею значение ставим на место удаляемого

элемента. Если в ходе рекурсии наткнулись на пустое дерево или приоритет искомого элемента стал больше, чем приоритет рассматриваемых деревьев, то сигнализируем о том, что элемента нет в данной дерамиде. Т.к при реализации используется операция Merge, то сложность будет равняться O( $\log_2(N)$ ), где N — число элементов в дереве.

### 2.2. Описание структур данных и функций

1. *Class bin_tree_node — класс узла дерамиды.*
   *bin_tree_node(int key, int prior)* — конструктор.

2. *Struct Elem_Pair — структура предназначенная для хранения ключа и приоритета дерева.*

3. *Функция Elem_Pair *Elem_Generator(int count) — функция предназначенная для генирации массива случайных элементов в размере count.*

*Функция Elem_Pair *Increasing_Elem_Generator(int count) — функция предназначення для генирации массива возрастающих элементов в размере count.*

*Функция void Split(std::shared_ptr<bin_tree_node> t, int key, std::shared_ptr<bin_tree_node> &left, std::shared_ptr<bin_tree_node> &right)* — функция разделения дерева на два поддерева, в одном поддереве будут все элементы с ключами меньше key, а в другом больше.

*Функция void Insert (std::shared_ptr<bin_tree_node> &t, std::shared_ptr<bin_tree_node> it)* — функция вставки элемента в дерево.

*Функция std::shared_ptr<bin_tree_node> Treaps_Generator(int count)* — функция генерации рандомной дерамиды.

*Функция std::shared_ptr<bin_tree_node> Increasing_Treaps_Generator(int count)* — функция генерации возрастающей дерамиды.

*Функция std::shared_ptr<bin_tree_node> Treaps_Building(Elem_Pair *seq, std::vector<int> vec)* — функция создания дерамиды по заданной входной последовательности.

*Функция void Merge(std::shared_ptr<bin_tree_node> &t, std::shared_ptr<bin_tree_node> left, std::shared_ptr<bin_tree_node> right)* — функция объединения двух деревьев.

*Функция void Erase(std::shared_ptr<bin_tree_node> &t, int key, int prior)* — функция удаления элемента из дерева.

*Функция bool String_Checker(std::string &str, int &index, std::vector<int> &vec)* — проверка входной последовательности на валидность.

*Функция void Display_Treap(std::shared_ptr<bin_tree_node> root, int space = 0, int height = 5)* — функция печати дерева на экран.

4. *Class Analysis — класс исследования.*

*Метод start_insert_middle()* - метод запускающий исследование вставки в среднем случае.

*Метод start_insert_worst()* - метод запускающий исследование вставки в худшем случае.

*Метод start_erase_middle()* - метод запускающий исследование исключения в среднем случае.

*Метод start_erase_worst()* - метод запускающий исследование вставки в худшем случае.

# 3. ТЕСТИРОВАНИЕ

Входные данные: на вход программе подается строка из элементов дерамиды. Затем программа принимает на вход букву «i»  или «d», которая определяет дальнейшие действия программы (i — вставка элемента, d — удаление элемента). А затем элемент, который требуется вставить или удалить. В случае если будет введено «e» произойдет завершение программы. Примеры работы программы продемонстрированы на рисунка 1-4.

Выходные данные: программа выводит дерамиду с удаленным или вставленным элементом. Либо выводит сообщение о том, что удаляемого элемента нет в дереве, и выводит исходное дерево.

Дерево выводится слева направо.

Тестирование производится с помощью скрипта на языке программирования Python. Для запуска скрипта для тестирования прописываем команду python3 test_skript.py. Исходный код представлен в приложении А. Результаты тестирования представлены в таблице Б.1

```
/home/viktor/CLionProjects/algosy/cmake-build-debug/algosy "(10 60)(20 80)(30 10)(40 30)(50 90)(60 40)(70 50)(80 20)"
success

          80(20)

     70(50)

          60(40)

50(90)

          40(30)

               30(10)

     20(80)

          10(60)
if you want to exit press 'e' and if you want to delete an item press 'd' or Insert item press 'i': i
```

Рисунок 1 — пример работы программы при вставке

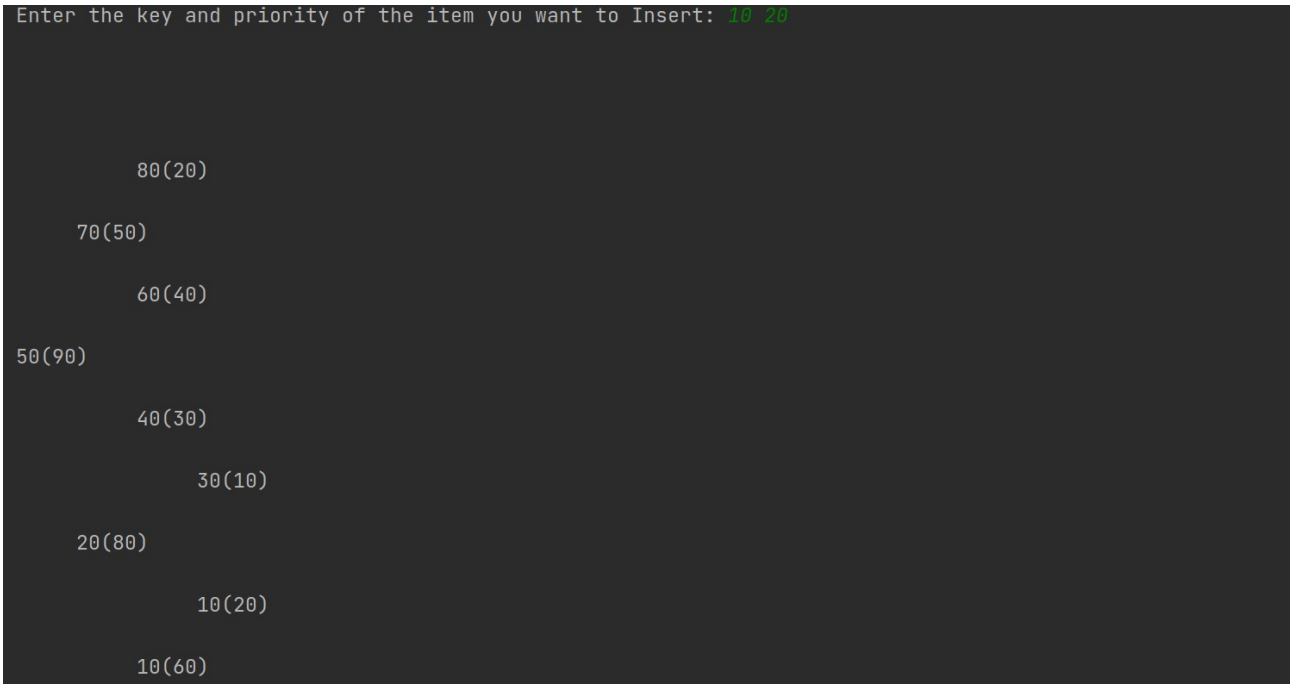



Рисунок 2 — пример работы программы при вставке


Рисунок 3 — пример работы программы при удалении

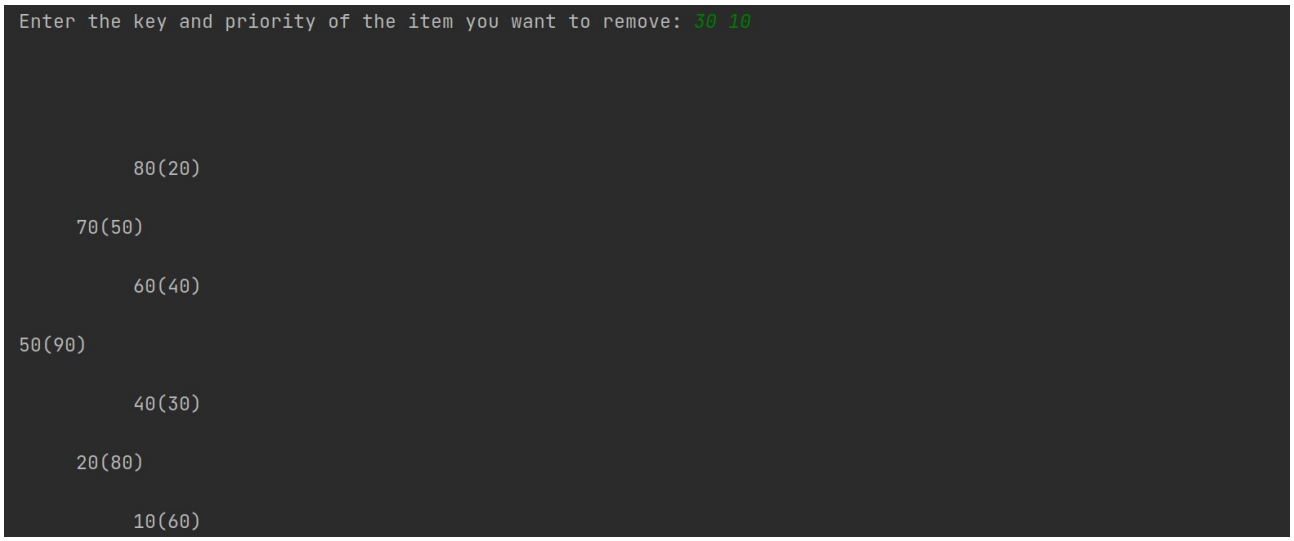



Рисунок 4 — пример работы программы при удалении

Таблица Б.1 — Результаты тестирования

| № п/п | Входные данные | Выходные данные | Результат проверки |
|---|---|---|---|
| 1 | (10 60)(20 80)(30 10)(40 30)(50 90)(60 40)(70 50)(80 20) d 40 30 e | success<br>80(20)<br>70(50)<br>60(40)<br>50(90)<br>40(30)<br>30(10)<br>20(80)<br>10(60)<br>if you want to exit press 'e' and if you want to delete an item press 'd' or Insert item press 'i': Enter the key and priority of the item you want to remove: | success |

| | | | |
|---|---|---|---|
| | | 80(20)<br>70(50)<br>60(40)<br>50(90)<br>30(10)<br>20(80)<br>10(60)<br>if you want to exit press 'e' and if you want to delete an item press 'd' or Insert item press 'i': | |
| 2 | | Wrong expression | Failed |
| 3 | (10 60)(20 80)(30 10)(40 30)(50 90)(60 40)(70 50)(80 20) i 100 30 e | success<br>80(20)<br>70(50)<br>60(40)<br>50(90)<br>40(30)<br>30(10)<br>20(80)<br>10(60)<br>if you want to exit press 'e' and if you want to delete an item press 'd' or Insert item press 'i': Enter the key and priority of the item you want to Insert:<br>100(30) | success |

| | | | |
|---|---|---|---|
| | | 80(20)<br>70(50)<br>60(40)<br>50(90)<br>40(30)<br>30(10)<br>20(80)<br>10(60)<br>if you want to exit press 'e' and if you want to delete an item press 'd' or Insert item press 'i': | |
| 4 | (10 60)(20 80)(30 10)(40 30)(50 90)(60 40)(70 50)(80 20) l 40 30 e | success<br>80(20)<br>70(50)<br>60(40)<br>50(90)<br>40(30)<br>30(10)<br>20(80)<br>10(60)<br>if you want to exit press 'e' and if you want to delete an item press 'd' or Insert item press 'i': You entered an invalid character. | Invalid character |
| 5 | (10 60)(20 80)(30 10)(40 30)(50 90)(60 40)(70 50(80 20) l 40 30 e | wrong string | failed |

# 4. ИССЛЕДОВАНИЕ

## 4.1. Исследование операций на дерамиде

### 4.1.1 Вставка

Чтобы произвести исследование алгоритма вставки заведем статическую переменную, которую будем инкрементировать каждый раз, когда будет рекурсивно вызываться функция вставки. Таким образом подсчитаем количество операций, которое должно будет в теории составлять O( $\log_2(N)$ ).

### 4.1.2 Удаление

Чтобы произвести исследование алгоритма удаления заведем статическую переменную, которую будем инкрементировать каждый раз, когда будет рекурсивно вызываться функция удаления. Таким образом подсчитаем количество операций, которое должно будет в теории составлять O( $\log_2(N)$ **).**

## 4.2. План исследования

Для подтверждения теоретической оценки был создан класс Analysis, который генерирует входные данные двух типов - строго возрастающей последовательности и случайной последовательности. Каждая из видов последовательности используется для операций вставки в дерамиду и удаления из него. Во время исследования работы операций вставки и удаления фиксируется количество вызовов соответствующих функций, в зависимости от высоты дерева.

## 4.3. Результаты исследования

В ходе выполнения исследования были получены данные по числу операций при вставке и удалении элемента в зависимости от числа элементов в дереве. На основе этих данных на языке python были созданы графики иллюстрирующие зависимости операций от числа элементов. Разработанный код в приложении А. Далее ниже будут представлены результаты исследования на рисунках 5 — 8. Синей линией будет изображен график полученный в ходе

исследования, а красной теоретический график (график функции O( $\log_2(N)$ )для среднего случая и O(*N)* для худшего случая).

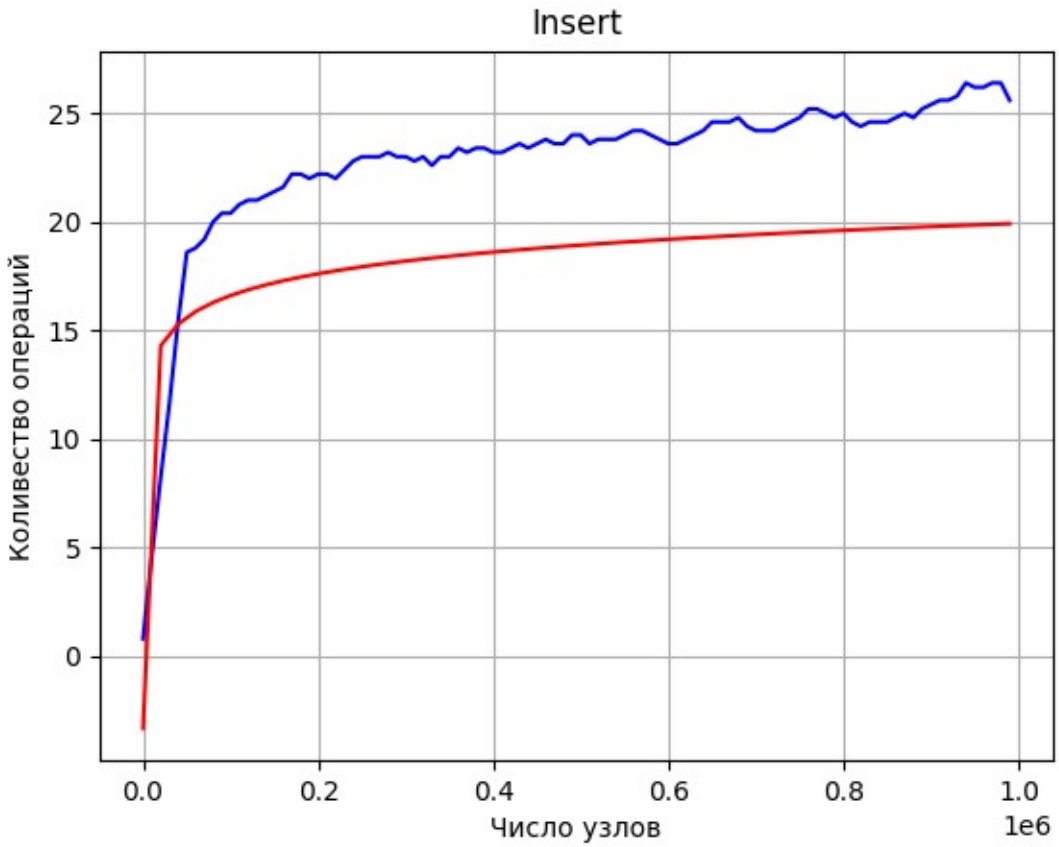



Рисунок 5 — График зависимоти количества операций от числа узлов вставки в среднем случае

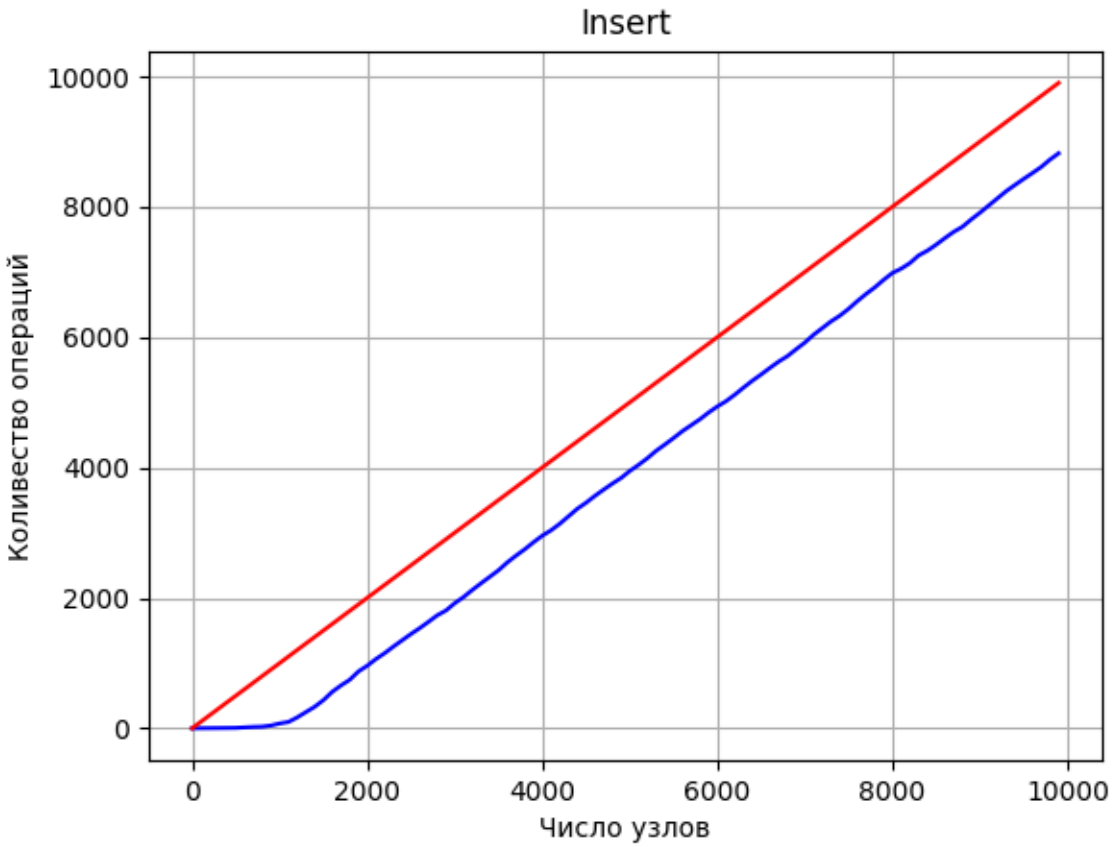



Рисунок 6 — График зависимоти количества операций от числа узлов вставки в худшем случае

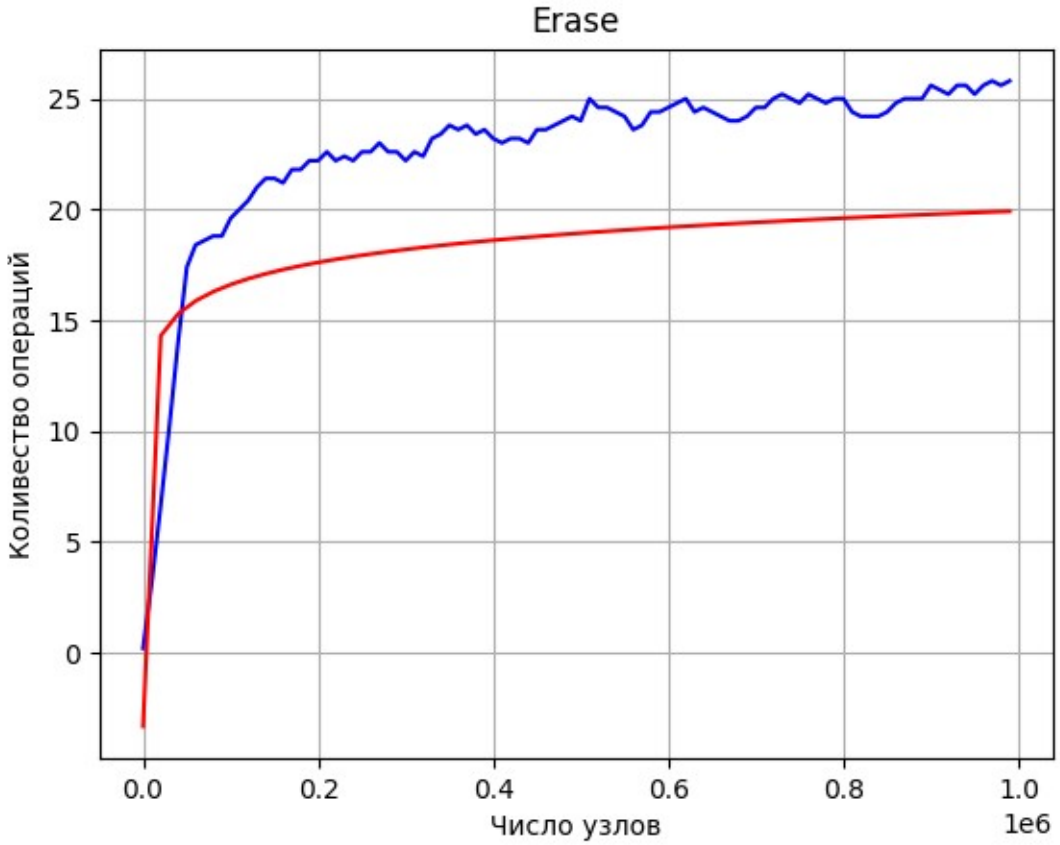



Рисунок 7 — График зависимоти количества операций от числа узлов удаления в среднем случае

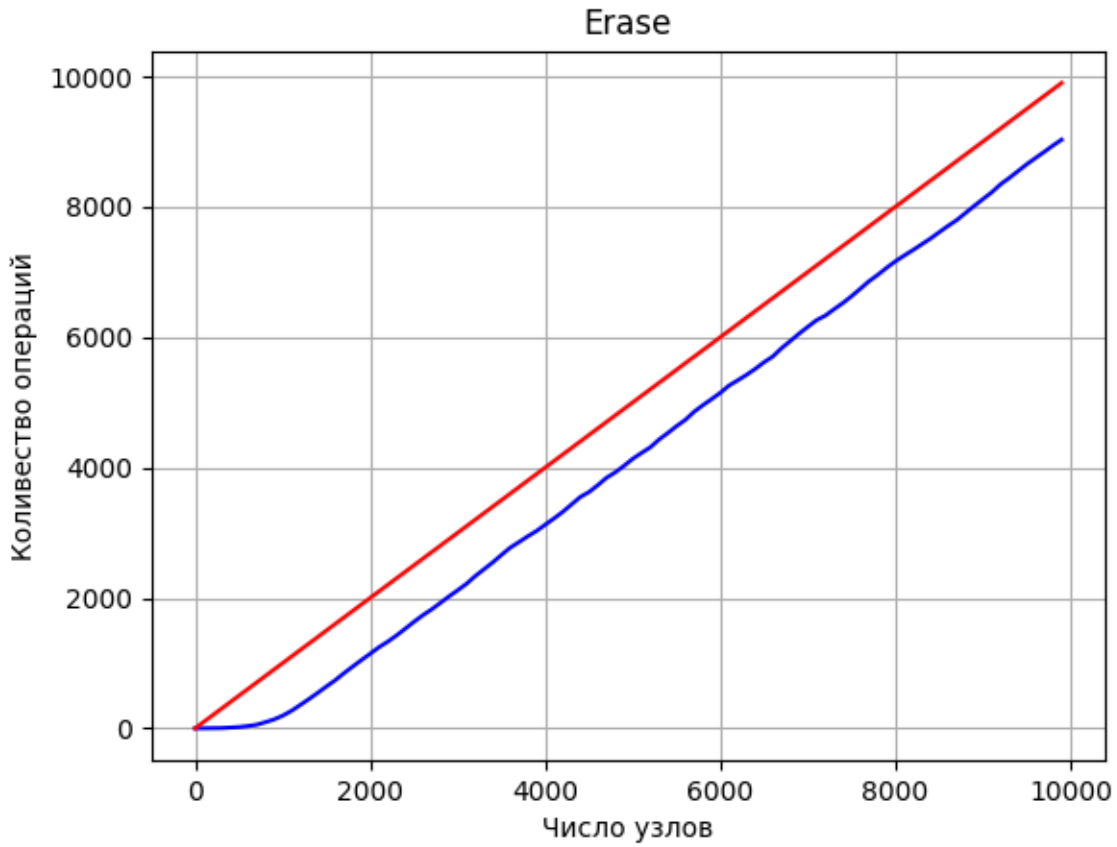



Рисунок 8 — График зависимоти количества операций
от числа узлов удаления в худшем случае

Полученные графики подтверждают теоретические оценки сложностей и имеют возможно некоторое незначительное отклонение от теории. Связано это с тем, что в теоретических сложностях еще участвует некоторая констнатна, но как известно при оценке сложности константа отбрасывается.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была реализована структура рандомизированная дерамида, а также операции вставки и удаления для нее. На основе числовыхметрик были построены графики.

Полученные практические результаты сравнили с теоретическими оценками. Таким образом, была доказана теоретическая оценка асимптотики работы операций вставки и удаления в дерамиде.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

*1. Дерамида URL: https://habr.com/ru/post/101818/ (дата обращения: 11.12.2020).*

*2. Дерамида URL: https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BA%D0%B0%D1%80%D1%82%D0%BE%D0%B2%D0%BE_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE (дата обращения: 11.12.2020).*

*3. Дерамида URL: https://e-maxx.ru/algo/treap (дата обращения: 11.12.2020).*

# ПРИЛОЖЕНИЕ А

# НАЗВАНИЕ ПРИЛОЖЕНИЯ

*Имя файла: main.cpp*

```
#include <iostream>
#include <memory>
#include <vector>
#include <chrono>
#include <fstream>



static int oper_counter = 0;

// структура элемента дерева
struct Elem_Pair{
   int key_elem, prior_elem;
};


// класс узла дерева
class bin_tree_node{
public:
   std::shared_ptr<bin_tree_node> left;
   std::shared_ptr<bin_tree_node> right;
   int key, prior;
   bin_tree_node(){};
   bin_tree_node(int key, int prior) : key(key), prior(prior), left(NULL), right(NULL){};
};



// генератор рандомных элементов
Elem_Pair *Elem_Generator(int count){
   Elem_Pair *Elem_Array = new Elem_Pair[count];
   for (int i = 0; i < count; i++){
      Elem_Array[i].key_elem = rand() % 1001;
      Elem_Array[i].prior_elem = rand() % 1001;
   }
   return Elem_Array;
}


// генератор возрастающих элементов
Elem_Pair *Increasing_Elem_Generator(int count){
   Elem_Pair *Elem_Array = new Elem_Pair[count];
   for (int i = 0; i < count; i++){
      Elem_Array[i].key_elem = i;
      Elem_Array[i].prior_elem = i;
   }
```

```
    return Elem_Array;
}



// функция разделения дерева в два поддерева
void Split(std::shared_ptr<bin_tree_node> t, int key, std::shared_ptr<bin_tree_node> &left,
std::shared_ptr<bin_tree_node> &right){
    //oper_counter++;
    if (!t){
        left = right = NULL;
    } else if (key < t -> key){
        Split(t->left, key, left, t->left);
        right = t;
    } else {
        Split(t->right, key, t->right, right);
        left = t;
    }
}



// функция вставки элемента в дерево
void Insert (std::shared_ptr<bin_tree_node> &t, std::shared_ptr<bin_tree_node> it) {
    oper_counter++;
    if (!t)
        t = it;
    else if (it->prior > t->prior)
        Split(t, it->key, it->left, it->right), t = it;
    else if (it -> key < t -> key){
        Insert(t->left, it);
    } else {
        Insert(t->right, it);
    }
}



// генератор рандомной дерамиды
std::shared_ptr<bin_tree_node> Treaps_Generator(int count){
    if (count < 0){
        std::cout << "there must be at least 1 element\n";
        return nullptr;
    }

    Elem_Pair *seq = Elem_Generator(count);
    std::shared_ptr<bin_tree_node> *Array_Items = new std::shared_ptr<bin_tree_node>[count];
    for (int i = 0; i < count; i++){
        Array_Items[i] = std::make_shared<bin_tree_node>();
        Array_Items[i] -> key = seq[i].key_elem;
        Array_Items[i] -> prior = seq[i].prior_elem;
    }
    for (int i = 1; i < count; i++){
```

```
        Insert(Array_Items[0], Array_Items[i]);
    }
    delete []seq;
    std::shared_ptr<bin_tree_node> head = std::make_shared<bin_tree_node>();
    head = Array_Items[0];
    delete []Array_Items;
    return head;
}

// генератор дерамиды с возрастающими элементами
std::shared_ptr<bin_tree_node> Increasing_Treaps_Generator(int count){
    if (count < 0){
        std::cout << "there must be at least 1 element\n";
        return nullptr;
    }

    Elem_Pair *seq = Increasing_Elem_Generator(count);
    std::shared_ptr<bin_tree_node> *Array_Items = new std::shared_ptr<bin_tree_node>[count];
    for (int i = 0; i < count; i++){
        Array_Items[i] = std::make_shared<bin_tree_node>();
        Array_Items[i] -> key = seq[i].key_elem;
        Array_Items[i] -> prior = seq[i].prior_elem;
    }
    for (int i = 1; i < count; i++){
        Insert(Array_Items[0], Array_Items[i]);
    }
    delete []seq;
    std::shared_ptr<bin_tree_node> head = std::make_shared<bin_tree_node>();
    head = Array_Items[0];
    delete []Array_Items;
    return head;
}


// создание дерамиды по переданным значениям
std::shared_ptr<bin_tree_node> Treaps_Building(Elem_Pair *seq, std::vector<int> vec){
    std::shared_ptr<bin_tree_node> Array_Items[vec.size() / 2];
    for (int i = 0; i < vec.size() / 2; i++){
        Array_Items[i] = std::make_shared<bin_tree_node>();
        Array_Items[i] -> key = seq[i].key_elem;
        Array_Items[i] -> prior = seq[i].prior_elem;
    }
    for (int i = 1; i < vec.size() / 2; i++){
        Insert(Array_Items[0], Array_Items[i]);
    }
    return Array_Items[0];
}


// функция объединения двух поддеревьев
void Merge(std::shared_ptr<bin_tree_node> &t, std::shared_ptr<bin_tree_node> left,
std::shared_ptr<bin_tree_node> right){
```

```
    if (!left || !right){
        t = left ? left : right;
    } else if (left -> prior > right -> prior){
        Merge(left -> right, left -> right, right);
        t = left;
    } else {
        Merge(right -> left, left, right -> left);
        t = right;
    }
}

void Erase(std::shared_ptr<bin_tree_node> &t, int key, int prior){
    oper_counter++;
    if (!t || t -> prior < prior){
        std::cout << "Element with stack parameters not found\n";
        return;
    }
    if (t -> key == key && t -> prior == prior){
        Merge(t, t -> left, t -> right);
        return;
    } else {
        if (key < t -> key){
            Erase(t -> left, key, prior);
        } else {
            Erase(t -> right, key, prior);
        }
    }
}


// проверка строки на валидность
bool String_Checker(std::string &str, int &index, std::vector<int> &vec){
    if (str[index] != '('){
        return false;
    }
    index++;
    std::string check_num = "";
    while (str[index] != ' ' && str[index]){
        check_num += str[index];
        index++;
    }
    if (!str[index]){
        return false;
    }
    if (!check_num.empty()) {
        char *endptr;
        const char *c_string = check_num.c_str();
        vec.push_back(strtol(c_string, &endptr, 10));
        if (*endptr) {
            return false;
        }
    } else {
```

```
        return false;
    }
    index++;
    check_num = "";
    while (str[index] != ')' && str[index]){
        check_num += str[index];
        index++;
    }
    if (!str[index]){
        return false;
    }
    if (!check_num.empty()) {
        char *endptr;
        const char *c_string = check_num.c_str();
        vec.push_back(strtol(c_string, &endptr, 10));
        if (*endptr) {
            return false;
        }
    } else {
        return false;
    }
    if (str[index + 1]){
        index++;
        return String_Checker(str, index, vec);
    } else {
        return true;
    }
}




// печать дерева на экран
void Display_Treap(std::shared_ptr<bin_tree_node> root, int space = 0, int height = 5) { //display
treap
    if (root == nullptr)
        return;
    space += height;
    Display_Treap(root->right, space);
    std::cout << '\n';
    for (int i = height; i  < space; i++)
        std::cout << ' ';
    std::cout << root -> key << "(" << root -> prior << ")\n";
    Display_Treap(root->left, space);
}




class Analysis{
```

```
    std::ofstream insert_middle_case;
    std::ofstream insert_worst_case;
    std::ofstream erase_middle_case;
    std::ofstream erase_worst_case;
public:
    void start_insert_middle(){
        std::shared_ptr<bin_tree_node> head;
        std::shared_ptr<bin_tree_node> insert_elem;
        insert_elem = std::make_shared<bin_tree_node>();
        insert_elem = Treaps_Generator(1);
        insert_middle_case.open("insert_middle_case.txt");
        for (int i = 1; i < 1000000; i += 10000){
            insert_middle_case << i;
            head = Treaps_Generator(i);
            oper_counter = 0;
            double value_oper = 0;
            for (int j = 0; j < 50; j++){
                Insert(head, insert_elem);
                value_oper += oper_counter;
                insert_elem = Treaps_Generator(1);
                value_oper = 0;
            }
            insert_middle_case << " " << oper_counter / 50 << '\n';
        }
        insert_middle_case.close();
    };
    void start_insert_worst(){
        std::shared_ptr<bin_tree_node> head;
        std::shared_ptr<bin_tree_node> insert_elem;
        insert_elem = std::make_shared<bin_tree_node>();
        insert_elem = Treaps_Generator(1);
        insert_worst_case.open("insert_worst_case.txt");
        for (int i = 1; i < 10000; i += 100){
            insert_worst_case << i;
            head = Increasing_Treaps_Generator(i);
            oper_counter = 0;
            double value_oper = 0;
            for (int j = 0; j < 50; j++){
                Insert(head, insert_elem);
                value_oper += oper_counter;
                insert_elem = Treaps_Generator(1);
                value_oper = 0;
            }
            insert_worst_case << " " << oper_counter / 50 << '\n';
        }
        insert_worst_case.close();
    };
    void start_erase_middle(){
        std::shared_ptr<bin_tree_node> head;
        std::shared_ptr<bin_tree_node> insert_elem;
        insert_elem = std::make_shared<bin_tree_node>();
        insert_elem = Treaps_Generator(1);
```

```
        erase_middle_case.open("erase_middle_case.txt");
        for (int i = 1; i < 1000000; i += 10000){
            erase_middle_case << i;
            head = Treaps_Generator(i);
            oper_counter = 0;
            double value_oper = 0;
            for (int j = 0; j < 50; j++){
                Erase(head, insert_elem -> key, insert_elem -> prior);
                value_oper += oper_counter;
                insert_elem = Treaps_Generator(1);
                value_oper = 0;
            }
            erase_middle_case << " " << oper_counter / 50 << '\n';
        }
        erase_middle_case.close();
    };
    void start_erase_worst(){
        std::shared_ptr<bin_tree_node> head;
        std::shared_ptr<bin_tree_node> insert_elem;
        insert_elem = std::make_shared<bin_tree_node>();
        insert_elem = Treaps_Generator(1);
        erase_worst_case.open("erase_worst_case.txt");
        for (int i = 1; i < 10000; i += 100){
            erase_worst_case << i;
            head = Increasing_Treaps_Generator(i);
            oper_counter = 0;
            double value_oper = 0;
            for (int j = 0; j < 50; j++){
                Erase(head, insert_elem -> key, insert_elem -> prior);
                value_oper += oper_counter;
                insert_elem = Treaps_Generator(1);
                value_oper = 0;
            }
            erase_worst_case << " " << oper_counter / 50 << '\n';
        }
        erase_worst_case.close();
    };

};



int main(int argc, char* argv[]) {
    srand(time(0));
    std::shared_ptr<bin_tree_node> head = std::make_shared<bin_tree_node>();
    if(argc == 1){
        std::cout << "Wrong expression\n";
        return 0;
    }
    //Analysis analys;
    //analys.start_erase_worst();
    //return 0;
```

```
    char *endptr;
    std::string str(argv[1]);
    char what_to_do_one = argv[2][0];
    int key = strtol(argv[3], &endptr, 10);
    int prior = strtol(argv[4], &endptr, 10);
    char what_to_do_two = argv[5][0];
    int index = 0;
    std::vector<int> vec;
    bool k = String_Checker(str, index, vec);
    Elem_Pair seq[vec.size() / 2];
    if (k){
        for (int i = 0; vec[i]; i += 2){
            seq[i / 2].key_elem = vec[i];
            seq[i / 2].prior_elem = vec[i + 1];
        }
        std::cout << "success\n";
        head = Treaps_Building(seq, vec);
        Display_Treap(head);
        while (true){
            char what_to_do;
            std::cout << "if you want to exit press 'e' and if you want to delete an item press 'd' or Insert
item press 'i': ";
            what_to_do  = what_to_do_one;
            if (what_to_do == 'e'){
                return 0;
            } else if (what_to_do == 'd'){
                std::cout << "Enter the key and priority of the item you want to remove: ";
                Erase(head, key, prior);
                std::cout << "\n\n\n";
                Display_Treap(head);
                what_to_do_one = what_to_do_two;
            } else if (what_to_do == 'i'){
                std::cout << "Enter the key and priority of the item you want to Insert: ";
                std::shared_ptr<bin_tree_node> insert_elem = std::make_shared<bin_tree_node>();
                insert_elem -> key = key;
                insert_elem -> prior = prior;
                Insert(head, insert_elem);
                std::cout << "\n\n\n";
                Display_Treap(head);
                what_to_do_one = what_to_do_two;
            } else {
                std::cout << "You entered an invalid character.\n";
                return 0;
            }
        }

    } else {
        std::cout << "wrong string\n";
        return 0;
    }
}
```

*Имя файла: main.py*

```
import matplotlib.pyplot as plt
import numpy as np


def moving_average(a, n=3) :
    ret = np.cumsum(a)
    ret[n:] = ret[n:] - ret[:-n]
    return ret[0:] / n

x = np.arange(0.1, 140, 0.01)

f = open("file_result.txt", 'r')

num = 0
secs = []
oper = []

for line in f:
    s = line.split()
    secs.append(float(s[0]))
    oper.append(float(s[1]))


oper = moving_average(oper, 5)



t = np.linspace(0.1, max(secs))

#y = np.log2(t)
y = t

plt.plot(secs, oper, "blue")
plt.plot(t, y, "red")

plt.xlabel("Число узлов")
plt.ylabel("Коливество операций")
plt.title("Erase")
plt.grid()
plt.show()
```

Имя файла: test_skript.py

```
import unittest
import subprocess


class tester(unittest.TestCase):
```

```
	def test1(self):
		with open('./Tests/test1.txt', 'r') as file:
			for line in file:
				s = line.replace('\n', ' ')
				print("Input: ", s)
				arguments = []
				arguments = s.split('_')
				self.assertIn('success', subprocess.check_output(["./cw", arguments[0],
arguments[1], arguments[2], arguments[3], arguments[4]], universal_newlines=True))
				print("Output:", end = ' ')
				print(subprocess.check_output(["./cw", arguments[0], arguments[1],
arguments[2], arguments[3], arguments[4]], universal_newlines=True) + '\n')

	def test2(self):
		with open('./Tests/test2.txt', 'r') as file:
			for line in file:
				s = line.replace('\n', ' ')
				print("Input: ", s)
				arguments = []
				arguments = s.split('_')
				self.assertIn('Wrong expression', subprocess.check_output(["./cw"],
universal_newlines=True))
				print("Output:", end = ' ')
				print(subprocess.check_output(["./cw"], universal_newlines=True) + '\
n')


	def test3(self):
		with open('./Tests/test3.txt', 'r') as file:
			for line in file:
				s = line.replace('\n', ' ')
				print("Input: ", s)
				arguments = []
				arguments = s.split('_')
				self.assertIn('success', subprocess.check_output(["./cw", arguments[0],
arguments[1], arguments[2], arguments[3], arguments[4]], universal_newlines=True))
				print("Output:", end = ' ')
				print(subprocess.check_output(["./cw", arguments[0], arguments[1],
arguments[2], arguments[3], arguments[4]], universal_newlines=True) + '\n')

	def test4(self):
		with open('./Tests/test4.txt', 'r') as file:
			for line in file:
				s = line.replace('\n', ' ')
				print("Input: ", s)
				arguments = []
				arguments = s.split('_')
				self.assertIn('You entered an invalid character.',
subprocess.check_output(["./cw", arguments[0], arguments[1], arguments[2], arguments[3],
arguments[4]], universal_newlines=True))
				print("Output:", end = ' ')
```

```
                print(subprocess.check_output(["./cw", arguments[0], arguments[1],
arguments[2], arguments[3], arguments[4]], universal_newlines=True) + '\n')

    def test5(self):
        with open('./Tests/test5.txt', 'r') as file:
            for line in file:
                s = line.replace('\n', ' ')
                print("Input: ", s)
                arguments = []
                arguments = s.split('_')
                self.assertIn('wrong string', subprocess.check_output(["./cw",
arguments[0], arguments[1], arguments[2], arguments[3], arguments[4]],
universal_newlines=True))
                print("Output:", end = ' ')
                print(subprocess.check_output(["./cw", arguments[0], arguments[1],
arguments[2], arguments[3], arguments[4]], universal_newlines=True) + '\n')




if __name__ == '__main__':
    unittest.main()
```