

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 БДП с
рандомизацией

Студент гр. 9304

Прокофьев М.Д.

Преподаватель

Филатов Ар.Ю.

Санкт-Петербург

2020

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Прокофьев М.Д.

Группа 9304

Тема работы: Исследование операций вставки и исключения в БДП с рандомизацией

Исходные данные

Содержание пояснительной записки:

- Аннотация
- Содержание
- Введение
- Формальная постановка задачи
- Описание структур данных и функций, описание алгоритма
- Тестирование
- Исследование
- Заключение
- Список использованных источников

Предполагаемый объем пояснительной записки:

Не менее 36 страниц.

Дата выдачи задания: 23.11.2020

Дата сдачи реферата: 29.12.2020

Дата защиты реферата: 29.12.2020

Студент

Прокофьев М.Д.

Преподаватель

Филатов Ар.Ю.

АННОТАЦИЯ

В курсовой работе исследуются операции вставки и удаления элементов в случайном БДП с рандомизацией. Проходит исследование с помощью тестирования работы алгоритма, путем проведения экспериментов с различными входными данными. Проводится анализ средних и худших случаев, составляются числовые метрики на основе которых строятся графики для сравнения с теоретическими оценками

SUMMARY

In the course work, the operations of inserting and deleting elements in a random BDP with randomization are investigated. The research is carried out by testing the algorithm, by conducting experiments with various input data. The analysis of the best and worst cases is carried out, numerical metrics are compiled on the basis of which graphs are built for comparison with theoretical estimates

СОДЕРЖАНИЕ

	Введение	5
1.	Формальная постановка задачи	6
2.	Ход выполнения работы	7
2.1	Описание алгоритма	7
2.2	Описание структур данных и функций	8
3.	Тестирование	10
4.	Исследование	14
4.1	План исследования	14
4.2	Теоретическая оценка сложности алгоритма	14
4.3	Вставка в случайном БДП с рандомизацией	14
4.4	Удаление в случайном БДП с рандомизацией	15
4.5	Результаты исследования	15
	Заключение	18
	Список использованных источников	19
	Приложение А. Исходный код	20

ВВЕДЕНИЕ

Цель работы:

Изучение случайного БДП с рандомизацией, как структуру данных. Создание программы, которая считывает случайное БДП с рандомизацией, после чего дает пользователю контроль над изменением дерева, путем удаления какого-либо элемента или вставки. Анализ средних и худших случаев по ходу работы алгоритма

1. ФОРМАЛЬНАЯ ПОСТАНОВКА ЗАДАЧИ

Реализовать структуру данных случайное БДП с рандомизацией и исследовать операцию вставки и исключения элемента дерева в среднем и худшем случае для проверки теоретической оценки этих операций

2. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

2.1. Описание алгоритма

Алгоритм добавления элемента в дерево:

При вставке элемента в дерево, изначально происходит спуск вниз при этом при каждом спуске есть вероятность $1/k$, где k – количество элементов дерева, что произойдет вставка в корень, вставка в корень нужна для балансировки дерева. При спуске выбирается левое или правое направление, это зависит от вставляемого элемента, если он больше чем элемент текущего поддерева, вставка идет право, иначе влево.

Алгоритм удаления элемента из дерева:

При удалении узла придерживается следующий алгоритм: Спуском вниз, определяется узел, который подвергается удалению, при нахождении - обнуляется. При этом, т.к. функция рекурсивна, при выходе из нее, происходит шаг “вверх” к родителю. И т.к. произошло какое-то удаление, то либо левый потомок, либо правый будут пустые у “родителя”. С этого момента происходит следующее: если этот узел(родитель) имеет только правого потомка, то он заменяется на минимальный элемент правого поддерева этого узла. В случае, если он имеет только левого потомка, то данный узел заменяется на максимальный элемент левой ветки этого узла.

2.2. Описание структур данных и функций

1. Class binSTree – класс для представления случайного БДП с рандомизацией.

Поля:

Char info – содержание элемента

Int count – счетчик для вставки и удаления элемента

Int number – высота узла

std::shared_ptr<binSTree> hd – Указатель на первый элемент дерева

std::shared_ptr<binSTree> lt – Указатель на левое поддерево

std::shared_ptr<binSTree> rt – Указатель на правое поддерево

2. Class rd – класс, который содержит функции для проведения исследования.

Поля:

std::ofstream r_file – файл для записи экспериментов.

void rs_Insert (int size, bool r_case) – функция для вставки элементов при проведении исследования, где size – размер дерева, а r_case – случай (1 – средний, 0 – худший).

void rs_Delete (int size, bool r_case) – функция для удаления элементов при проведении исследования, где size – размер дерева, а r_case – случай (1 – средний, 0 – худший).

void create_unsorted(int count, std::insert_vector<int>& insert_v) – функция для создания неотсортированного вектора, который используется при постройке БДП с рандомизацией

void create_sorted(int count, std::insert_vector<int>& insert_v) – функция для создания отсортированного вектора, который используется при постройке БДП с рандомизацией

3. Функции:

`void r_Delete(std::shared_ptr<binSTree>& ptr, char info)` – функция для удаления элемента `info` указанного поддерева `ptr`.

`void r_Insert(std::shared_ptr<binSTree>& b, char x)` – функция для вставки элемента `x` в указанное поддерево `b`.

`char f_min(std::shared_ptr<binSTree>& p, int& ct)` – функция для нахождения минимального элемента указанного поддерева `p`.

`char f_max(std::shared_ptr<binSTree>& p, int& ct)` – функция для нахождения максимального элемента указанного поддерева `p`.

`bool chance(int random)` – функция, используемая для возврата случайного значения `1/random`.

`void to_rotate_right(std::shared_ptr<binSTree>& t)` – функция для поворота дерева вправо.

`void to_rotate_left(std::shared_ptr<binSTree>& t)` – функция для поворота дерева влево.

`void Insert_to_begin(std::shared_ptr<binSTree>& b, char x)` – функция, используемая для вставки в корень.

`void fix_numb(std::shared_ptr<binSTree>& b)` – функция для перерасчета значения узлов указанного дерева.

Используемые функции представлены в приложении А.

3. ТЕСТИРОВАНИЕ

Для тестирования был написан скрипт `testing.py`, который генерирует случайные тесты для анализа работы программы. В качестве входных данных подается строка из элементов, после чего происходит вывод сгенерированного случайного БДП с рандомизацией, он изображен на Рисунке 1:

```
Input:
10 51 43 67 23 45 65 12 23 98 57 38 12 14

Picture:

      / 98
     /  \
    /    \ 67
   /      \ 65
  /        \ 57
 /          \ 51
/            \ 45
 \           \ 43
  \          / 38
   \        / 23
    \      / 14
     \    / 12
      \  / 10

Choose action:
'+' - for insert element
 '-' - for delete element
Or print "esc" for exit
```

Рисунок 1 – Вывод дерева

После чего можно набрать “+” для вставки элемента, после этого в программе нужно ввести вставляемый элемент, что представлено на Рисунке 2:

```
Choose action:
'+' - for insert element
 '-' - for delete element
Or print "esc" for exit
+
Input the element for insert:
50
```

Рисунок 2 – Ввод элемента для вставки

После чего, программа выдает результат вставки:

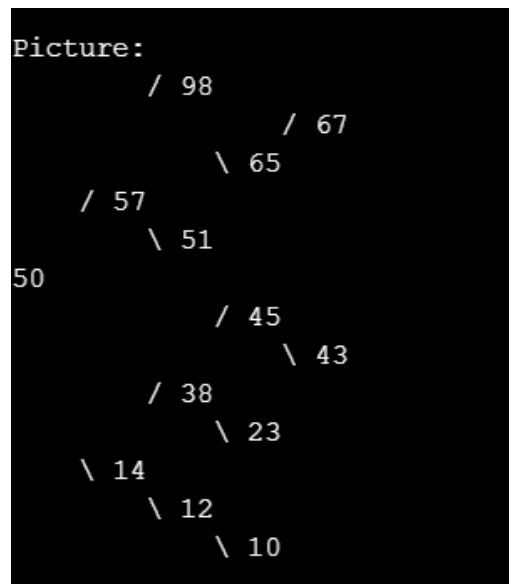


Рисунок 3 – Результат вставки

Также, можно набрать “-“ для удаления элемента:

```
Choose action:
'+' - for insert element
'-' - for delete element
Or print "esc" for exit
-
Input the element for delete:
12
```

Рисунок 4 – Ввод элемента для удаления

Где также программа выдаст результат:

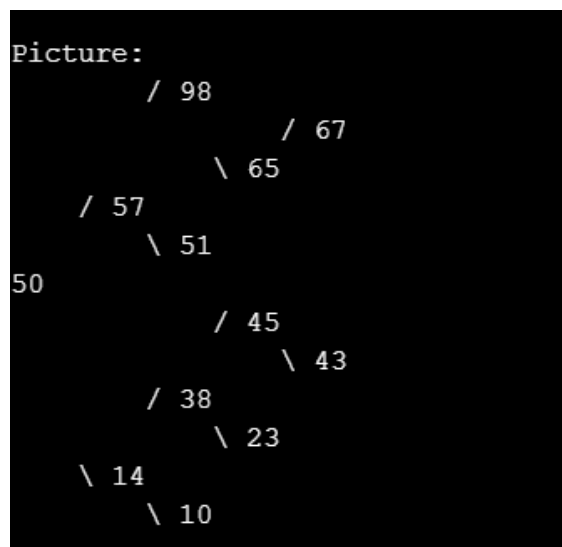
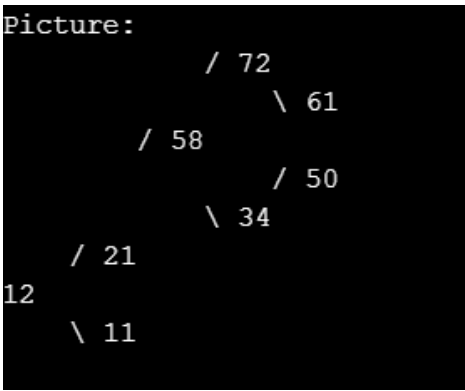
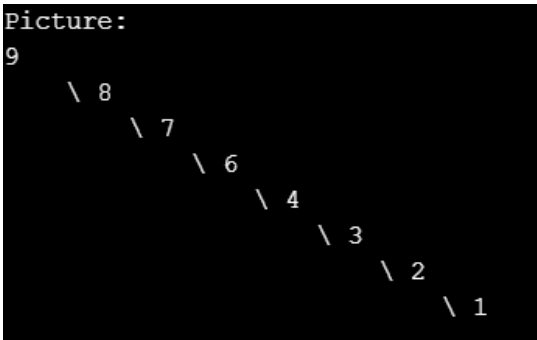
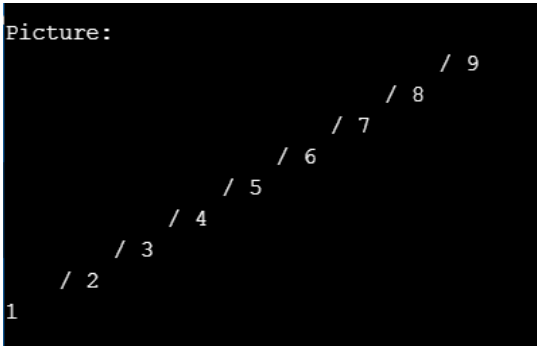
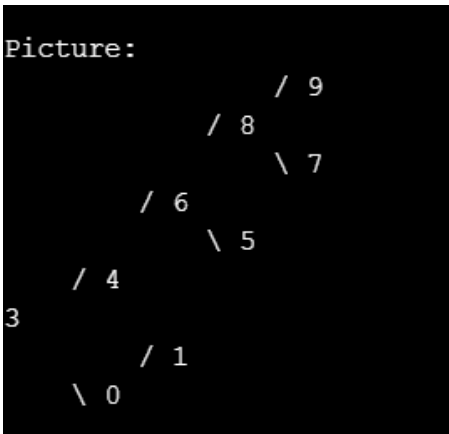


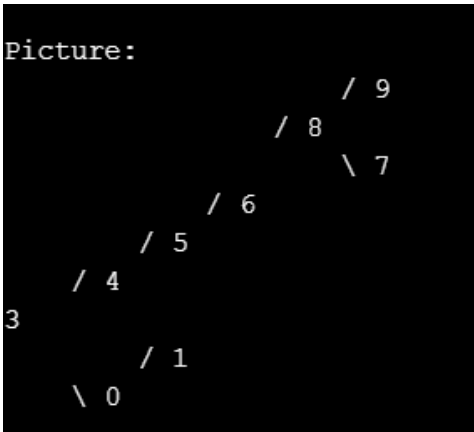
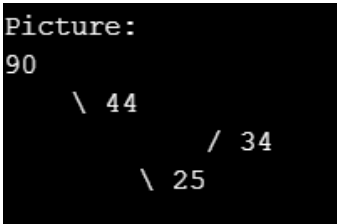
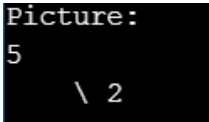
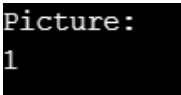
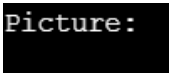
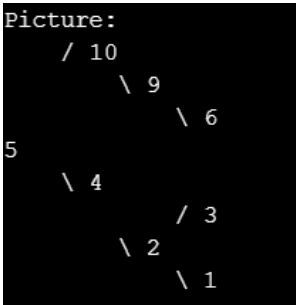
Рисунок 5 – Результат удаления

Исходный код скрипта представлен в приложении А.

Результаты тестирования представлены в таблице Б.1:

Таблица Б.1 — Результаты тестирования

№ п/п	Входные данные	Выходные данные (изображение)	Выходные данные
1.	50 61 72 34 58 21 11 + 12 esc		Finish
2.	1 2 3 4 5 6 7 8 9 - 5 esc		Finish
3.	9 8 7 6 5 4 3 2 1 + 0 esc		Finish
4.	1 3 5 7 9 0 8 6 4 2 - 2 esc		Finish

5.	1 3 5 7 9 0 8 6 4 2 - 2 esc (для проверки случайности)		Finish
6.	34 25 18 44 90 - 18 esc		Finish
7.	5 + 2 esc		Finish
8.	+ 1 esc		Finish
9.	8 - 8 esc		Finish
10.	1 6 3 2 9 4 10 + 5 esc		Finish

4. ИССЛЕДОВАНИЕ

4.1. План исследования:

План исследования предполагает следующие действия: генерация случайных входных данных разного количества, после чего проведение экспериментов по вставке и удалению элементов в соответствующем случайном БДП с рандомизацией. После чего – анализ худших и средних случаев, соответственно, при вставке и удалении элементов. Для анализа создан класс `bd`, в нем хранятся функции, необходимые для исследования.

4.2. Теоретическая оценка сложности алгоритмов:

Есть два случая при выполнении алгоритмов вставки или удаления в случайном БДП с рандомизацией: средний и худший. Средний предполагает, что в подаваемом массиве элементы перемешаны, при худшем – элементы упорядочены. Т.к. рассматривается БДП с рандомизацией, то алгоритм вставки или удаления, при среднем и худшем случае имеет сложность $O(\log_2 n)$. В обоих случаях алгоритм имеет одинаковую сложность, поскольку случайное БДП с рандомизацией обладает большей сбалансированностью.

4.3. Вставка в случайном БДП с рандомизацией:

В классе `bd` создана функция `rs_Insert()` для выполнения вставки при среднем и худшем случае. Функция имеет на вход переменную `r_case`, отвечающую за то средний случай рассматривается, или худший (при среднем – 1, при худшем – 0), а также количество элементов `size`.

Она работает по следующему алгоритму: изначально создается массив из элементов, который перемешивается или упорядочивается (в зависимости от значения переменной `r_case`), после чего, создается случайное БДП с рандомизацией на основе ранее созданного массива. При каждой вставке элемента в исследуемое БДП, происходит запись в файл номера элемента и количества шагов алгоритма вставки.

4.4. Удаление в случайном БДП с рандомизацией:

Кроме того, в классе `bd` создана функция `rs_Delete()`, которая аналогично `r_insert()` имеет на вход переменные `r_case` и `size`.

Работает следующим образом: Создается массив из упорядоченных или неотсортированных элементов (зависит от переменной `r_case`). После этого создается случайное БДП с рандомизацией, и элементы вставляются в ранее созданный массив. После этого происходит удаление элементов дерева с записью в файл номера удаляемого элемента и количество шагов, понадобившихся для его удаления

4.5. Результаты исследования:

Для визуализации результатов был написан Python-скрипт, отображающий графики, иллюстрирующие зависимость между количеством вызовов операций каждой из функций и количеством узлов:

1.1) Вставка при среднем случае:

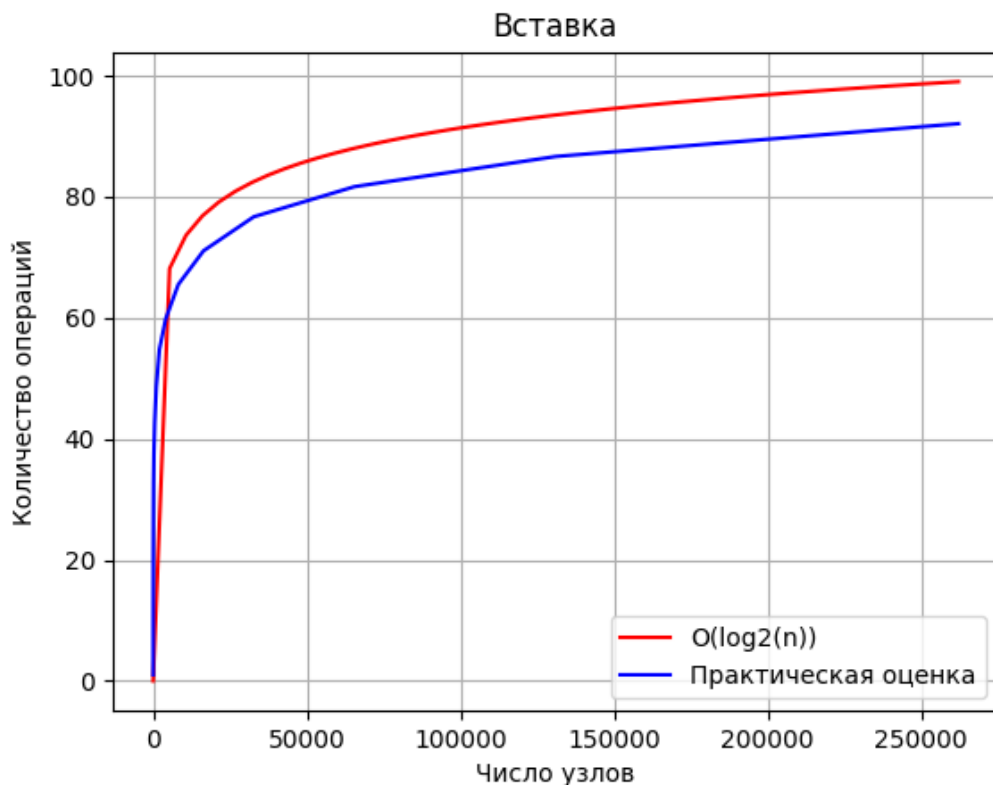


Рисунок 6 – Зависимость между количеством операций при вставке и числом узлов в среднем случае

1.2) Вставка при худшем случае:

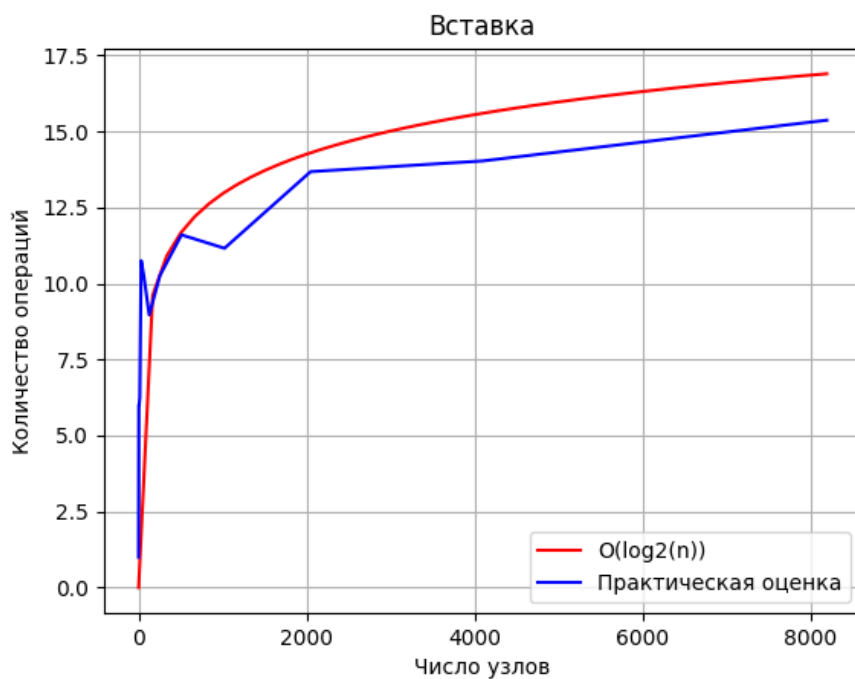


Рисунок 7 – Зависимость между количеством операций при вставке и числом узлов в худшем случае

2.2) Удаление элемента при среднем случае:

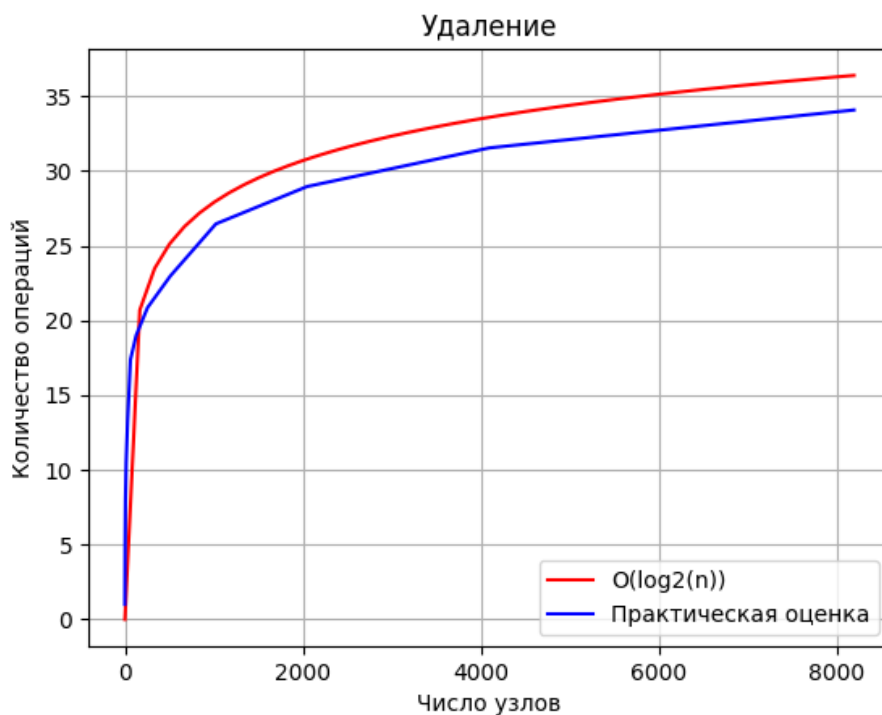


Рисунок 8 – Зависимость между количеством операций при удалении и числом узлов в среднем случае

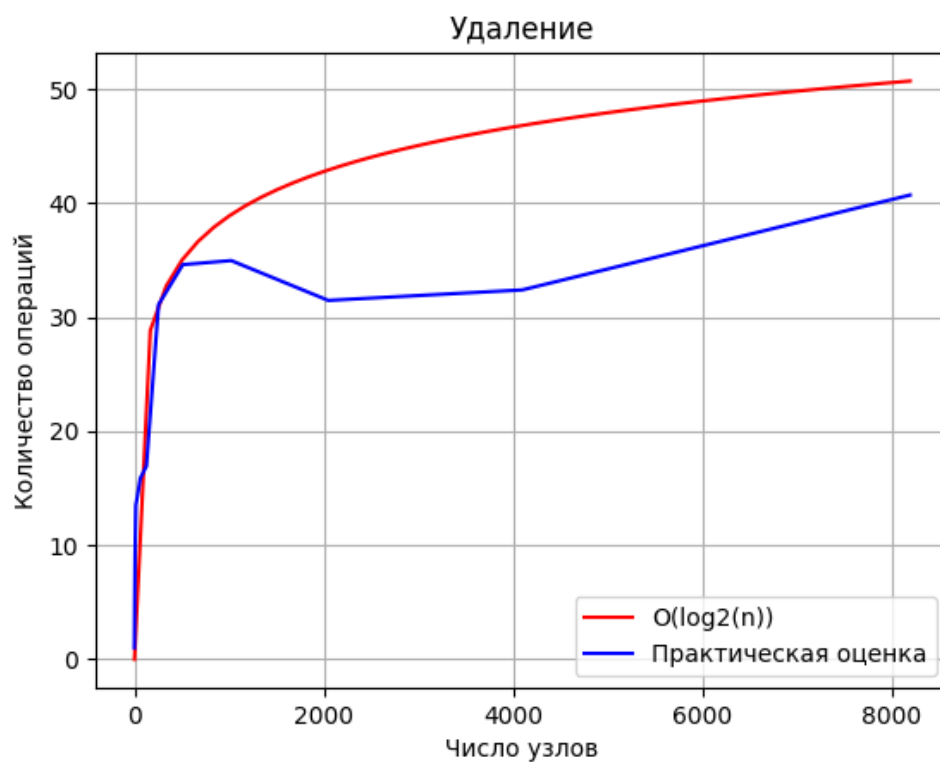


Рисунок 9 – Зависимость между количеством операций при удалении и числом узлов в худшем случае

По графикам зависимости, приведенным выше, видно, что практическая оценка во всех случаях приблизительно равняется $\log_2(n)$, что, в целом, подтверждает теоретическую оценку сложности алгоритма, приведённой в пункте 4.2.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы была реализована структура случайного БДП с рандомизацией.

Созданы функции вставки и удаления. Проведено исследование сложности алгоритма вставки и удаления при различных ситуациях: при среднем и худшем случае. Было выявлено, что при среднем и худшем случае зависимость между количеством операций в каждой из этих функций и количеством узлов приблизительно одинакова, что соответствует теоретической оценке.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. <https://habr.com/ru/post/145388/>
2. <https://myslide.ru/presentation/skachat-sluchajnye-binarnye-derevya-poiska-lekciya->
3. <http://proteus2001.narod.ru/gen/txt/12/randtree.html>
4. <https://intuit.ru/studies/courses/12181/1174/lecture/25260?page=2>

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД

Файл main.cpp

```
#include <iostream>
#include <memory>
#include <string.h>
#include <fstream>
#include <vector>
#include <sstream>
#include <ifstream>
#include <time.h>

static int c_ins = 0;
static int del_ins = 0;

class binSTree{
public:
    int info;
    int count;
    int number;
    std::shared_ptr<binSTree> hd { nullptr };
    std::shared_ptr<binSTree> lt { nullptr };
    std::shared_ptr<binSTree> rt { nullptr };
};

int chance(int random)
{
    return (rand()%random);
}
```

```

void to_rotate_right(std::shared_ptr<binSTree>& t)
{
    c_ins++;
    std::shared_ptr<binSTree> x;
    if(t==nullptr)
        std::cout<<"to_rotate_right(NULL)"<<std::endl;
    else
    {
        x=t->lt;
        t->lt=x->rt;
        x->rt=t;
        t=x;
    }
}

```

```

void to_rotate_left(std::shared_ptr<binSTree>& t)
{
    c_ins++;
    std::shared_ptr<binSTree> x;
    if(t==nullptr)
        std::cout<<"to_rotate_right(NULL)"<<std::endl;
    else
    {
        x=t->rt;
        t->rt = x->lt;
        x->lt=t;
        t=x;
    }
}

```

```

void Insert_to_begin(std::shared_ptr<binSTree>& b, int x)
{
    c_ins++;
    if(b==nullptr){
        b = std::make_shared<binSTree>();
        if(b!=nullptr){
            b->info=x;
            b->count=1;
        }else{
            std::cout << "1 Memory not enough\n";
            exit(1);
        }
    }

    }else if(x<b->info) {
        Insert_to_begin(b->lt, x);
        to_rotate_right(b);
    }else if(x>b->info){
        Insert_to_begin(b->rt, x);
        to_rotate_left(b);
    }
    else b->count++;
}

```

```

void r_Insert(std::shared_ptr<binSTree>& b, int x){
    c_ins++;
    if(b==nullptr)
    {
        b = std::make_shared<binSTree>();
        if(b!=nullptr)

```

```

    {
        b->info=x;
        b->count=1;
        b->number=1;
        return;
    }
    else
    {
        std::cout << "1 Memory not enough\n";
        exit(1);
    }
}
if(rand()%(b->number+1) == 0)
{
    Insert_to_begin(b,x);
    return;
}
if(x<b->info)
    r_Insert(b->lt, x);
else {
    r_Insert(b->rt, x);
}
}

void fix_numb(std::shared_ptr<binSTree>& b){
    if(b){
        if(b->lt && b->rt){
            b->number = 2;
            fix_numb(b->lt);
            fix_numb(b->rt);

```

```

        b->number += b->lt->number;
        b->number += b->rt->number;
    }
    if(b->lt && !b->rt){
        b->number = 1;
        fix_numb(b->lt);
        b->number += b->lt->number;
    }
    if(!b->lt && b->rt){
        b->number = 1;
        fix_numb(b->rt);
        b->number += b->rt->number;
    }
}
}

```

```

int f_min(std::shared_ptr<binSTree>& p, int& ct){
    std::shared_ptr<binSTree> temp = std::make_shared<binSTree>();
    temp = p;
    while(temp->lt)
        temp = temp->lt;
    ct = temp->count;
    return temp->info;
}

```

```

int f_max(std::shared_ptr<binSTree>& p, int& ct){
    std::shared_ptr<binSTree> temp = std::make_shared<binSTree>();
    temp = p;
    while(temp->rt)
        temp = temp->rt;
}

```



```

    ct = temp->count;
    return temp->info;
}

```

```

void r_Delete(std::shared_ptr<binSTree>& ptr,int info){
    del_ins++;
    if(ptr){
        if(ptr == ptr->hd && !ptr->lt && !ptr->rt){
            ptr->hd = nullptr;
            return;
        }
        if(info < ptr->info){
            r_Delete(ptr->lt,info);
        }else if (info > ptr->info){
            r_Delete(ptr->rt,info);
        }else if (info == ptr->info){
            if(!ptr->lt && !ptr->rt){
                ptr = nullptr;
            }else if (ptr->rt){
                int lt_time = f_min(ptr->rt, ptr->count);
                r_Delete(ptr->rt, lt_time);
                ptr->info = lt_time;
            }else if (ptr->lt){
                int rt_time = f_max(ptr->lt ,ptr->count);
                r_Delete(ptr->lt, rt_time);
                ptr->info = rt_time;
            }
        }
    }
}

```

```

        std::cout << "There is no input_intent " << info << " in tree.\n";
    }
}

void drawTree(std::shared_ptr<binSTree> curNode, int level, int direction)
{
    if(curNode)
    {
        drawTree(curNode->rt, level + 1, 1);
        for(int i = 0; i < level; i++) std::cout << "  ";
        if(direction == 1) std::cout << "/ ";
        else if(direction == 2) std::cout << "\\ ";
        std::cout << curNode->info << std::endl;
        drawTree(curNode->lt, level + 1, 2);
    }
}

void print(std::shared_ptr<binSTree> curNode){
    if(curNode == nullptr)
        return;
    print(curNode->lt);
    std::cout << curNode->info;
    print(curNode->rt);
}

class rd{
    std::ofstream r_file;
public:

```

```

void rs_Insert(int size = 16384, bool r_case){
    r_file.open("insert_test.txt");
    std::vector<int> insert_v;
    std::shared_ptr<binSTree> My_Tree = nullptr;
    if(r_case)
        create_unsorted(size, insert_v);
    else
        create_sorted(size, insert_v);
    for(int i = 0; i < size; i++) {
        c_ins = 0;
        r_Insert(My_Tree, insert_v[i]);
        fix_numb(My_Tree);
        r_file << i << " " << c_ins << "\n";
    }
    r_file.close();
}

```

```

void rs_Delete(int size = 8192, bool r_case){
    srand(time(0));
    r_file.open("delete_test.txt");
    std::vector<int> insert_v;
    std::shared_ptr<binSTree> My_Tree = nullptr;
    if(r_case)
        create_unsorted(size, insert_v);
    else
        create_sorted(size, insert_v);
    for(int j = 0; j < size; j++){
        r_Insert(My_Tree, insert_v[j]);
        fix_numb(My_Tree);
    }
}

```

```

    }
    for(int i = 0; i < size; i++) {
        del_ins = 0;
        r_Delete(My_Tree, insert_v[insert_v.size() - 1 - i]);
        r_file << size - 1 - i << " " << del_ins << "\n";
    }
    r_file.close();
}

```

```

void create_unsorted(int count, std::vector<int>& insert_v) {
    int temp;
    for(int i = 0; i < count; i++)
        insert_v.push_back(i);
    for(int i = count - 1; i >= 1; i--) {
        int j = rand()%(count);
        temp = insert_v[j];
        insert_v[j] = insert_v[i];
        insert_v[i] = temp;
    }
}

```

```

void create_sorted(int count, std::vector<int>& insert_v){
    for (int i = count; i > 0; i--)
        insert_v.push_back(i);
}
};

```

```

int main(int argc, char* argv[])
{

```

```

srand(time(nullptr));
std::ifstream in("input.txt");
std::string s;
getline(in, s);
std::cout << "Input:" << "\n" << s << "\n";
std::shared_ptr<binSTree> My_Tree = nullptr;
std::istringstream n(s);
int input_int;
while (n >> input_int) {
    r_Insert(My_Tree, input_int);
}
std::string choice;
std::string int_str;
while(true)
{
    std::cout << "\nPicture:\n";
    drawTree(My_Tree, 0, 0);
    std::cout << "\nChoose action:\n+' - for insert element\n'- - for delete
element\nOr print \"esc\" for exit\n";
    getline(in, choice);
    if(choice == "+")
    {
        std::cout << "Input the input_intent for insert:\n";
        getline(in, int_str);
        int input_int2 = atoi(int_str.c_str());
        std::cin >> input_int2;
        r_Insert(My_Tree, input_int2);
    }
    if(choice == "-")
    {

```

```

        std::cout << "Input the input_intent for delete:\n";
        getline(in, int_str);
        int input_int2 = atoi(int_str.c_str());
        std::cin >> input_int2;
        r_Delete(My_Tree, input_int2);
    }
    if(choice == "esc")
    {
        break;
    }

}
std::cout << "Finish";
return 0;
}

```