МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: Рандомизированное БДП

Студентка гр. 8381 _____ Гречко В.Д.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студентка Гречко В.Д.

Группа 8381

Тема работы: Рандомизированное БДП

Исходные данные: необходимо провести исследование алгоритма вставки и удаления в рандоминизированное бинарное дерево поиска, включающее генерацию входных данных, использование их для измерения количественных характеристик алгоритмов, сравнение экспериментальных результатов с теоретическими.

Содержание пояснительной записки:
«Содержание», «Введение», «Задание», «Описание программы», «Тестирование», «Исследование», «Заключение», «Список использованных источников».

Предполагаемый объем пояснительной записки:
Не менее 30 страниц.

Дата выдачи задания:

Дата сдачи реферата:

Дата защиты реферата:

Студент _____ Гречко В.Д.

Преподаватель _____ Жангиров Т.Р.

АННОТАЦИЯ

В ходе выполнения курсовой работы была разработана программа с GUI, позволяющая исследовать алгоритм вставки в рандоминизированное бинарное дерево поиска, а также удаление заданного элемента. Программа обладает следующей функциональностью: создание файла, содержащего входные данные для построения бдп, создание рандоминизированного бдп по заданному файлу, вывод полученных результатов в файл.

SUMMARY

In the course work a program with a GUI was developed that allows you to examine the algorithm for inserting into a randomized binary search tree, as well as deleting a given element. The program has the following functionality: creating a file containing input data for building a BDP, creating a randomized BDP for a given file, outputting the results to a file.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы

Реализация и экспериментальное машинное исследование алгоритмов кодирования (Фано-Шеннона, Хаффмана), быстрого поиска на основе БДП или хеш-таблиц, сортировок.

Основные задачи

Генерация входных данных, использование их для измерения количественных характеристик структур данных, алгоритмов, действий, сравнение экспериментальных результатов с теоретическими.

Методы решения

Разработка программы велась на базе операционной системы Elementary OS в среде разработки QtCreator. Для создания графической оболочки использовался редактор интерфейса в QtCreator и система сигналов-слотов Qt.

1. ЗАДАНИЕ

Необходимо провести исследование алгоритма вставки и удаления в рандоминизированном бинарном дереве поиска в среднем и худшем случаях.

Исследование должно содержать:

1. Анализ задачи, цели, технологию проведения и план экспериментального исследования.
2. Генерацию представительного множества реализаций входных данных (с заданными особенностями распределения (для среднего и для худшего случаев)).
3. Выполнение исследуемых алгоритмов на сгенерированных наборах данных. При этом в ходе вычислительного процесса фиксируется как характеристики (например, время) работы программы, так и количество произведенных базовых операций алгоритма.
4. Фиксацию результатов испытаний алгоритма, накопление статистики.
5. Представление результатов испытаний, их интерпретацию и сопоставление с теоретическими оценками.

ОПИСАНИЕ ПРОГРАММЫ

2.1. Описание интерфейса пользователя

Интерфейс программы представляет из себя: панель ввода данных для генерации файла, панель ввода данных для построения дерева, панель ввода данных для исследования алгоритма удаления из бдп и панель вывода. Основные виджеты панели генерации файла и их назначение представлены в табл. 1.

Таблица 1 – Основные виджеты панели генерации файла

Класс объекта	Название виджета	Назначение
L i	num	Поле ввода размера бдп
	generate	Кнопка запуска генерации файла

Основные виджеты панели создания бдп и их назначение представлены в табл. 2.

Таблица 2 – Основные виджеты панели создания бдп

Класс объекта	Название виджета	Назначение
L i	Input_tree	Поле ввода данных для считывания
	Choose_file	Кнопка выбора файла для считывания
L i	Input_del	Поле для ввода элемента, подлежащего удалению
	PrintTree	Кнопка создания бдп
	Delete_elem	Кнопка запуска удаления элемента
QGraphicsView	graphicsView	Поле для графического представления дерева

Описание основного класса для рандоминизированных бдп

Для реализации бдп был создан класс BinTree. Основные методы класса представлены в табл. 5.

Таблица 5 – Основные функции работы с бинарным деревом

Функция	Назначение
void BinTree()	Создает пустое бинарное дерево
Node* insert(Node* p, int k)	Рандомизированная вставка нового узла с ключом k в дерево p
int max_depth(Node *hd)	Возвращает максимальную глубину дерева
Node* insertroot(Node* p, int k)	Вставка нового узла с ключом k в корень дерева p
Node* rotateright(Node* p)	Правый поворот вокруг узла p
Node* rotateleft(Node* q)	Левый поворот вокруг узла q
i	Получение размера дерева
void fixsize(Node* p)	Установление корректного размера дерева
Node* join(Node* p, Node* q)	Объединение двух деревьев
Node* remove(Node* p, int k)	Удаление из дерева p первого найденного узла с ключом k

Программа имеет возможность графического отображения полученного бинарного дерева с помощью виджета QGraphicsView. Функции, необходимые для графического представления дерева, представлены в табл. 6.

Таблица 6 – Функции, связующие графический интерфейс и алгоритмы

Функция	Назначение
QGraphicsScene *graphic(BinTree *tree, QGraphicsScene *&scene, int depth)	По заданному бинарному дереву выполняет рисование в объекте QGraphicsScene
int treePainter(QGraphicsScene *&scene, Node *node, int w, int h, int wDelta, int hDelta, QPen &pen, QBrush &brush, QFont &font, int depth)	Рекурсивный алгоритм обхода дерева и рисования узлов в заданном объекте

Описание алгоритма вставки и удаления в рандоминизированном бдп

Известно, что если заранее перемешать как следует все ключи и потом построить из них дерево (ключи вставляются по стандартной схеме в полученном после перемешивания порядке), то построенное дерево окажется неплохо сбалансированным (его высота будет порядка $2\log_2 n$ против $\log_2 n$ для идеально с б а лансированного дерева). Любой вводимый ключ может оказаться корнем с в е р о я т н о с т ь ю $\frac{1}{n+1}$ (n — размер дерева до вставки), следовательно выполняется с указанной вероятностью вставка в корень, а с вероятностью $1-\frac{1}{n+1}$ — рекурсивную вставка в правое или левое поддерево в зависимости от значения ключа в корне.

Удаление происходит по ключу — ищется узел с заданным ключом и этот узел удаляется из дерева. Основное свойство дерева поиска — любой ключ в левом поддереве меньше корневого ключа, а в правом поддереве — больше корневого ключа. Это свойство позволяет очень просто организовать поиск заданного ключа, перемещаясь от корня вправо или влево в зависимости от значения корневого ключа. Далее происходит объединение левого и правого поддеревьев найденного узла, удаляется узел и возвращается корень объединенного дерева.

Описание генерирования входных значений

Для возможности генерации входных данных было реализовано поле ввода размера требуемого массива. Далее, происходит генерация значений с помощью функции random () и проверка на уникальность значения. В конце данные записываются в файл.

3. ТЕСТИРОВАНИЕ

Вид программы

Вид программы после запуска представлен на рис. 2.

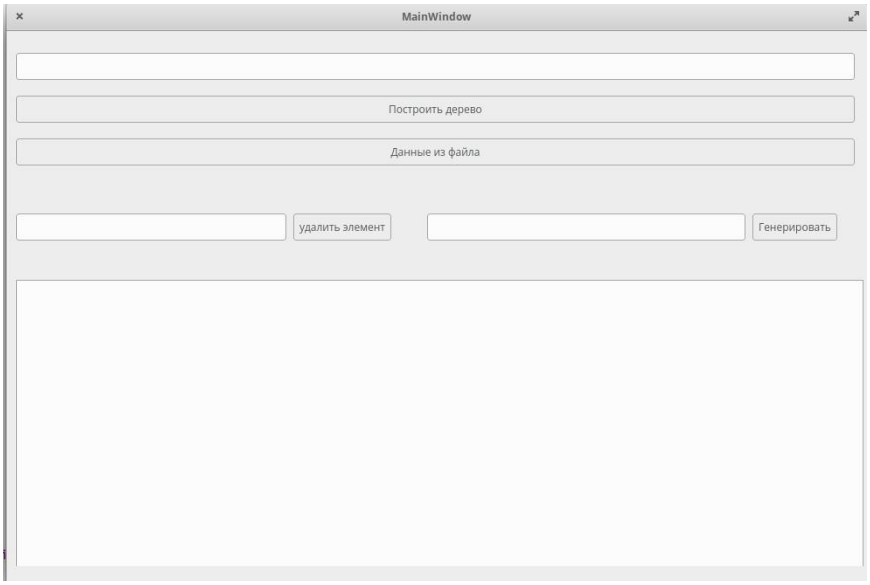

Рисунок 2 – Вид программы после запуска

Вид программы после создания хеш-таблицы представлен на рис. 3.

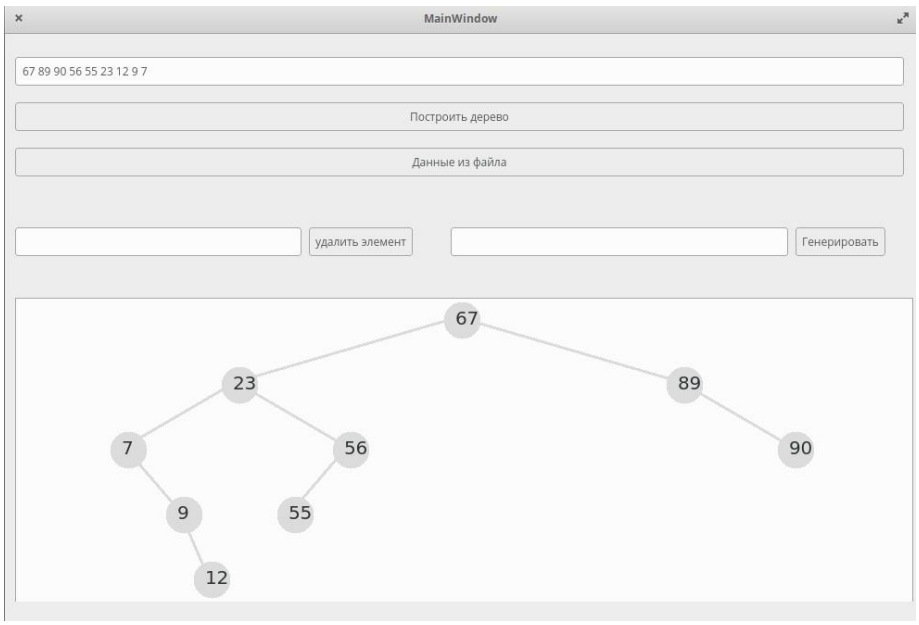

Рисунок 3 – Вид программы после создания хеш-таблицы

4. ИССЛЕДОВАНИЕ

4.1. План экспериментального исследования.

Для проведения исследования сложности алгоритма вставки в рандоминизированное бдп необходимо понимать, что на одном и том же наборе данных при повторном запуске будут получаться различные бдп. Следовательно, необходимо провести исследование не только для различного количества исходных данных, но и для одним и тех же. Опорным элементом при исследовании алгоритма станет высота получаемого дерева и число совершенных итераций. По аналогичному плану будет проводиться и исследование алгоритма удаления. После накопления данных необходимо провести сравнение результатов и сделать выводы об эффективности алгоритмов. Кроме того, объединив всю статистику, следует сравнить полученные экспериментальные зависимости от теоретических и сделать выводы о сложности алгоритма вставки в среднем и худшем случаях.

План проведения исследования:

- Получение информации о зависимости высоты дерева и числа итераций
  - ✓ Для одного и того же набора данных
  - ✓ Для различных наборов
- Анализ собранной информации, выводы о зависимостях эффективности алгоритма от указанных параметров
- Анализ собранной информации, сравнение экспериментальных значений с теоретическими, выводы о сложности алгоритма вставки и удаления.

Исследование зависимостей от полученной высоты дерева для алгоритма вставки

Так как при построении бдп вставка в корень определяется рандомно, то при одних и тех же значениях полученные деревья будет различаться. Был проведён ряд тестов: сначала для одного набора данных, а далее построение зависимостей для изменяющего размера входных данных.

С учетом того, что размер входного массива начинался от 500 элементов, его содержимое, а также содержимое файла можно приведено в приложении.

Далее будут представлены только результирующие графики. Для сравнения высоты в одном и том же наборе данных на графике отмечены три точки: максимальная, минимальная и средняя арифметическая высота дерева. Также построен теоретическое значение получаемого результата для большей н

Таблица 7 – Легенда

Цвет	Значение
Красный	Максимальное значение
Синий	Среднее значение
Жёлтый	Наименьшее значение
Зелёный	Теоретическое значение

Для массива, состоящего из 500 элементов, результат представлен в табл. 7.

Таблица 8 – Результаты тестирования

Максимальная высота	Минимальная высота	Средняя высота

Важно отметить, что количество бдп, не превышающих среднюю высоту, составило 26 из 50 возможных. Далее был построен сравнительный график, где представленная теоретическая функция - $c \log_2 n$. Кроме того, в полученных результатах значение максимальной высоты дерева встречалось единожды, в то время как минимальное шесть раз. Полученные данные представлены на рис. 4.

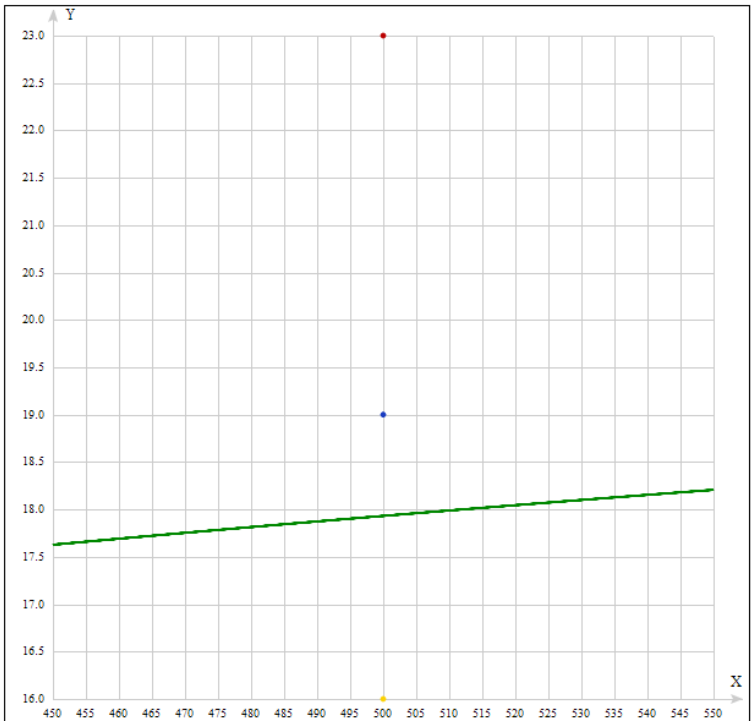

Рисунок 4 – Результаты тестирования 8 относительно теоретических значений

Для массива, состоящего из 1000 элементов, результат представлен в табл.

Таблица 9 – Результаты тестирования

Максимальная высота	Минимальная высота	Средняя высота
25	16	20

Важно отметить, что количество бдп, лежащих в диапозоне от средней высоты до минимальной, составило 31 из 50 возможных. Кроме того, в полученных результатах значение максимальной высоты дерева встречалось единожды.

Для массива, состоящего из 2000 элементов, результат представлен в табл.

Таблица 10 – Результаты тестирования

Максимальная высота	Минимальная высота	Средняя высота

Важно отметить, что количество бдп, лежащих в диапозоне от средней высоты до минимальной, составило 32 из 50 возможных. Кроме того, в полученных результатах значение максимальной и минимальной высоты дерева встречалось единожды.

Для массива, состоящего из 3000 элементов, результат представлен в табл.

Таблица 11 – Результаты тестирования

Максимальная высота	Минимальная высота	Средняя высота

Важно отметить, что количество бдп, не превышающих среднюю высоту, составило 35 из 50 возможных. Кроме того, в полученных результатах значение максимальной высоты дерева встречалось единожды, в то время как минимальное четырежды. Полученные данные представлены на рис. 5.

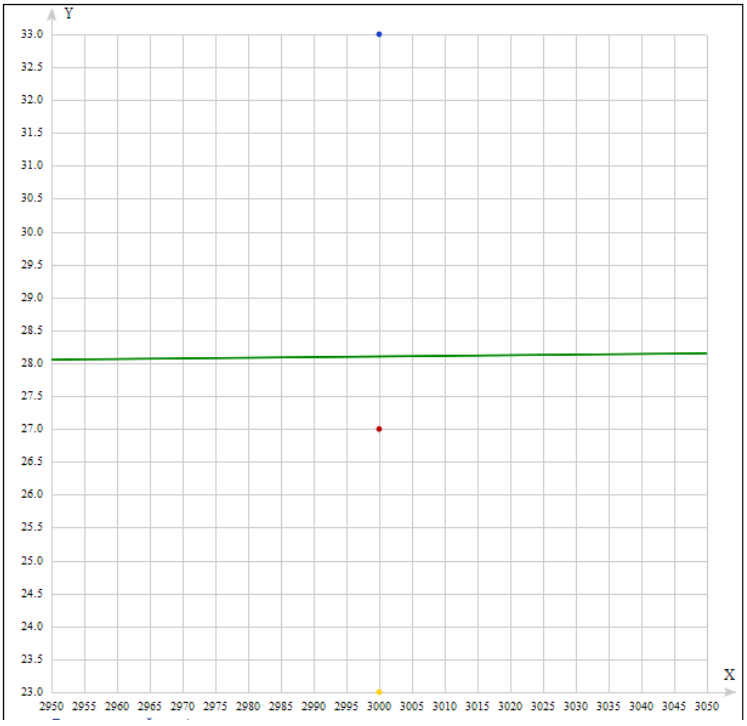

Рисунок 5 – Результаты тестирования 11 относительно теоретических значений

Для массива, состоящего из 4000 элементов, результат представлен в табл.

Таблица 12 – Результаты тестирования

Максимальная высота	Минимальная высота	Средняя высота

Важно отметить, что количество бдп, лежащих в диапозоне от средней высоты до минимальной, составило 35 из 50 возможных. Кроме того, в полученных результатах значение максимальной высоты дерева встречалось единожды, в то время как минимальное два раза.

Для массива, состоящего из 5000 элементов, результат представлен в табл.

Таблица 13 – Результаты тестирования

Максимальная высота	Минимальная высота	Средняя высота

Важно отметить, что количество бдп, не превышающих среднюю высоту, составило 33 из 50 возможных.

Для массива, состоящего из 7000 элементов, результат представлен в табл.

Таблица 14 – Результаты тестирования

Максимальная высота	Минимальная высота	Средняя высота

Важно отметить, что количество бдп, не превышающих среднюю высоту, составило 27 из 50 возможных. Полученные данные представлены на рис. 6.

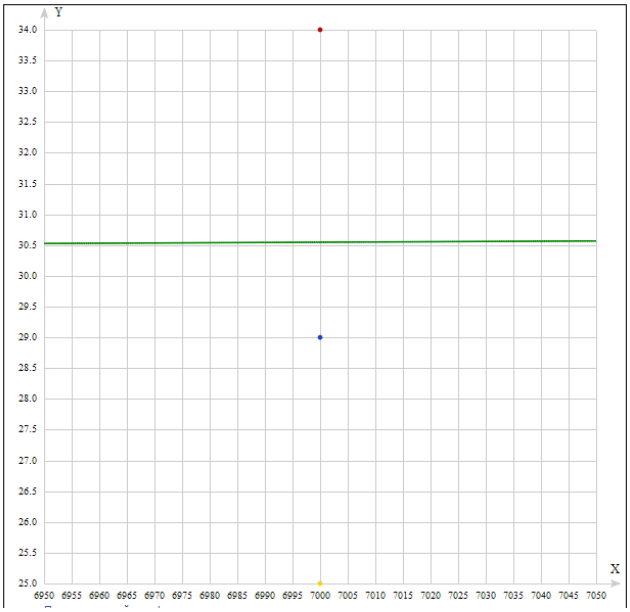

Рисунок 6 – Результаты тестирования 14 относительно теоретических значений

Далее, были получены результаты для набора данных размером от 500 до 7000 элементов с шагом 500, для построения общей картины. Полученные результаты сведены в общую таблицу (табл. 15) и на её основе было построено 4 графика: максимальная, минимальная, средняя и теоретическая высота (рис. 7).

Таблица 15 – Результаты тестирования

Размер массива	Минимальная высота	Средняя высота	Максимальная высота
	16	20	25
	21	23	28
	22	25	31

Продолжение таблицы 15

	22	27	32
	25	28	33
	25	29	36
	25	29	36
	26	29	37

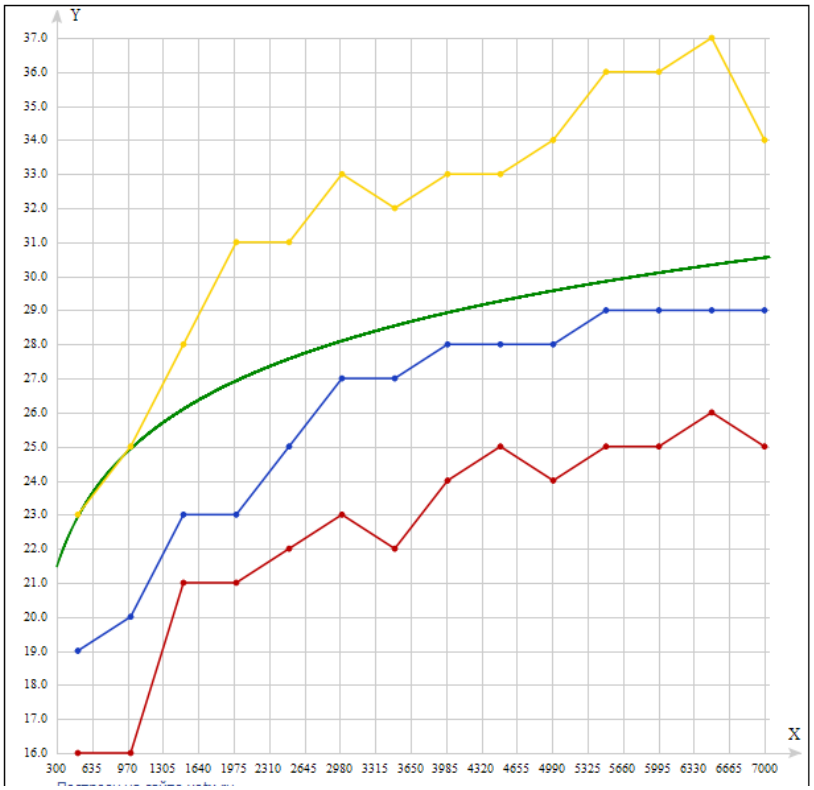

Рисунок 7 – График зависимости высоты дерева при построении дерева

Исходя из полученных результатов можно увидеть, что сложность алгоритма вставки является логарифмической. Стоит отметить, что худший случай сгенерировать самостоятельно не получится, так как алгоритм использует рандоминизацию для определения вероятности вставки в корень и, далее, высота

дерева фиксируется. Это приводит к тому, что вероятность получения несбарансированного дерева оказывается пренебрежимо малой при больших размерах деревьев. Как можно видеть из полученных результатов максимальная высота дерева также имеет логарифмическую сложность, но при других константах.

4.3. Исследование зависимостей от количества итераций для алгоритма вставки

Был проведен ряд тестов, где переменным параметром бралось количество итераций, необходимое для построения бинарного дерева поиска. Также, как и в прошлом подпункте сначала исследовалось количество итераций на одном и том же наборе исходных данных, а затем общие результаты сводились в таблицу.

Ниже представлены табл. 16 и рис.8, иллюстрирующие полученные результаты.

Таблица 16 – Результаты тестирования

Размер массива	Минимальное число итераций	Среднее арифметическое число итераций	Максимальное число итераций
	11850	12651	13993
	19077	20235	21374
	34764	36222	39661
	49309	52970	58127

Продолжение таблицы 16

	67411	70094	75059

	83954	88761	98453
	92722	97488	105155
	101613	107615	113993

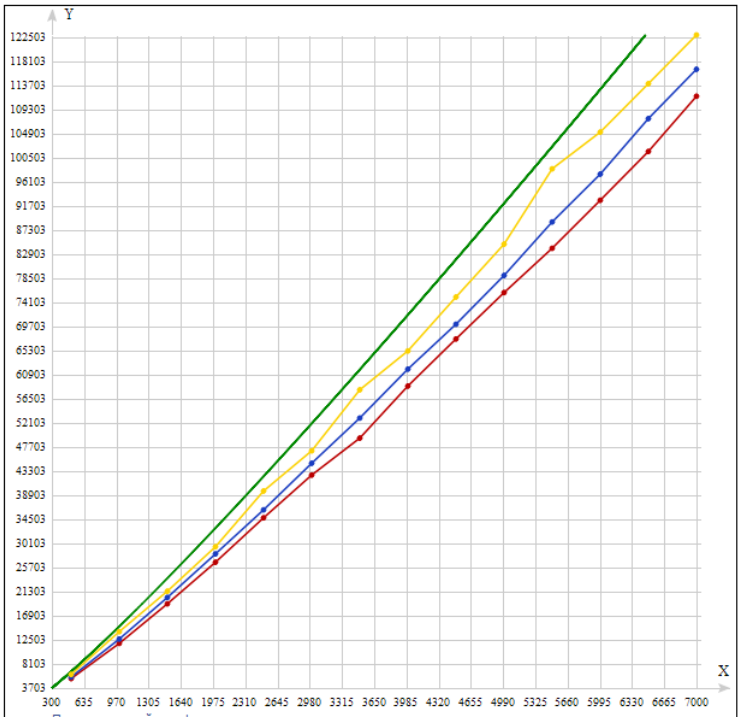

Рисунок 8 – График зависимости количества итераций от размера исходных данных

Количество итераций необходимое для вставки одного элемента в дерево с

о

с Из графиков можно увидеть, что количество итераций даже в своём

Наибольшем количестве не превосходит указанную сложность. Опять же, стоит

отметить, что худший случай был взят во время исследования алгоритма на

одном и том же наборе данных.

л В целом можно сделать вывод, что алгоритм вставки в

рандоминизированном бинарном дереве поиска является крайне эффективным и

е

т

n. Учитывая, что дерево имеет размер n, итоговая сложность алгоритма вставки для всего дерева составит $n \cdot \log_2 n$

с большой вероятностью строит сбалансированное дерево, избегая самого худшего случая – вырождения дерева в односвязный список.

Исследование зависимостей от количества итераций для алгоритма удаления

Для проведения тестов генерировался индекс числа, которое требовалось удалить и затем запускался ряд тестов для одного и того же набора значений. Так как генерируемое каждый раз дерево получалось различным, выбранный элемент также менял своё местоположение. На основе этих данных была сведена таблица 17, где указано наибольшее, наименьшее и среднее арифметическое число итераций.

Таблица 17 – Результаты тестирования

Размер массива	Минимальное число итераций	Среднее арифметическое число итераций	Максимальное число итераций

Продолжение таблицы 17

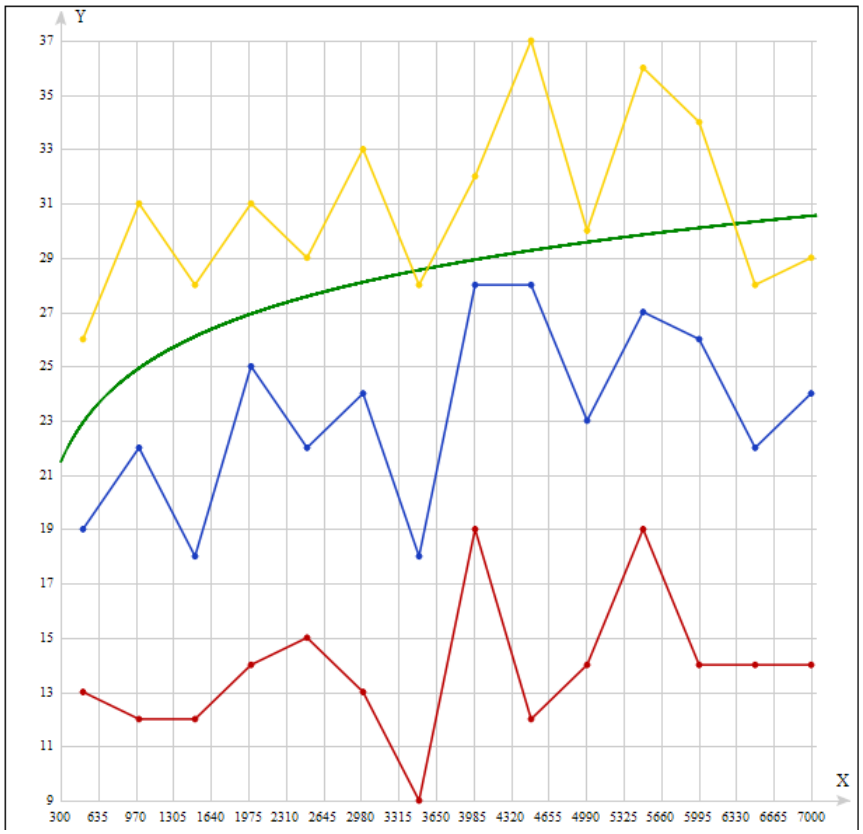

Рисунок 9 – График зависимости количества итераций от размера исходных данных

В результате можно сделать вывод, что несмотря на довольно резкие скачки получаемых значений (происходящие из-за случайной генерации удаляемого числа и повторного запуска теста для этого же набора данных: что меняет его положение в дереве), алгоритм удаления одного элемента из бинарного дерева поиска обладает логарифмической сложностью.

Выводы об исследовании алгоритма

В результате исследования было подтвержденно, что сложность алгоритмов вставки и удаления в рандоминизированном бинарном дереве поиска не превышает логарифмическую. Использование такого бинарного дерева поиска позволит избежать полностью несбалансированных случаев, те вырождения дерева в односвязный список.

Одним из основополагающих моментов эффективного поиска в таком бинарном дереве (для дальнейшего удаления элемента, к примеру) является правило хранения ключей: любой ключ в левом поддереве меньше корневого ключа, а в правом поддереве — больше корневого ключа.

Следует отметить, что значительную роль в построении такого дерева играет генерация вероятностей: именно от неё зависит наиболее оптимальная высота получаемого дерева. Тем не менее, как показывает исследование, даже худший случай получаемого значения высоты не является критическим и дерево продолжает оставаться достаточно сбалансированным.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была разработана программа, которая обладает следующей функциональностью: создание файла, содержащего входные данные для построения бинарного дерева поиска, создание рандоминизированного бинарного дерева поиска по заданному файлу, удаление выбранного пользователем элемента, а также запись результатов исследований в файл. С помощью программы было проведено исследование различных случаев алгоритма вставки в бинарное дерево и алгоритма удаления заданного значения. В ходе исследования была выявлена зависимость эффективности алгоритма от различных параметров. В результате было выявлено, что на эффективность влияет генерация вероятности вставки в корень, так как в дальнейшем это определяет дальнейшее построение дерева. В среднем случае дерево получается хорошо сбалансированным, особенно на больших объёмах данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Bjarne Stroustrup. A Tour of C++. M.: Addison-Wesley, 2018. 217 с.
2. T
3. Qt Documentation // Qt. URL: https://doc.qt.io/qt-5/index.html (дата обращения: 18.12.2000)
4. Рандомизированные деревья поиска URL: p
5. The Height of a Random Binary Search Tree // BRUCE REED URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.152.1289&rep=rep1&type=pdf (дата обращения: 18.12.2000)

s
f
o
r
G
e
e
k
s
h
t
t
p
s
w
w
w
g
e
e
k

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. MAIN.CPP

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ. MAINWINDOW.H

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QGraphicsItem>
#include <QGraphicsView>
#include <QGraphicsEffect>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private slots:
    void on_printTree_clicked();

    void on_delete_elem_clicked();

    void on_choose_file_clicked();


private:
    Ui::MainWindow *ui;
    QGraphicsScene *scene;
};

#endif // MAINWINDOW_H
```

ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД ПРОГРАММЫ. MAINWINDOW.CPP

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include<bin_struct.h>
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    scene = new QGraphicsScene;
    ui->graphicsView->setScene(scene);
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_printTree_clicked()
{

    QString data = ui->input_tree->text();
    QStringList abc = data.split(' ');
    int* mas = new int[100];
    int i =0;
    bool go = true;
    for (auto x:abc){
         bool convertOK;
         x.toInt(&convertOK);
         if(convertOK == false){
             go = false;

         }
         else{
            mas[i] = x.toInt();
            i++;
         }
    }
    if(!go){
        QMessageBox::warning(this,"Error", "Incorrect input");
        return;
    }
    BinTree* BT = new (BinTree);
    for(int j = 0; j < i; j++){
        BT->Head = BT->insert(BT->Head, mas[j]);
    }

    int depth = BT->max_depth(BT->Head);
    graphic(BT, scene, depth);

}

void MainWindow::on_delete_elem_clicked()
```

```
{
    QString data = ui->input_tree->text();
    QStringList abc = data.split(' ');
    int* mas = new int[100];
    int i =0;
    bool go = true;
    for (auto x:abc){
         bool convertOK;
         x.toInt(&convertOK);
         if(convertOK == false){
             go = false;

         }
         else{
            mas[i] = x.toInt();
            i++;
         }
    }
    if(!go){
        QMessageBox::warning(this,"Error", "Incorrect input");
        return;
    }
    BinTree* BT = new (BinTree);
    for(int j = 0; j < i; j++){
        BT->Head = BT->insert(BT->Head, mas[j]);
    }
    QString elem_ = ui->input_del->text();
    bool convert_;
    int elem = elem_.toInt(&convert_);
    if(!convert_){
        QMessageBox::warning(this,"Error", "Incorrect input");
        return;
    }
    BT->Head = BT->remove(BT->Head, elem);
    int depth = BT->max_depth(BT->Head);
    QString out;
    for(int l = 0; l < i - 1; l++){
        if (mas[l] == elem){
            for(int k = l; k < i - 1; k++){
                mas[k] = mas[k + 1];
            }
        }
        out.append(QString::number(mas[l]));
        if(l != i -2) out.append(" ");
    }
    ui->input_tree->setText(out);
    graphic(BT, scene, depth);
}

void MainWindow::on_choose_file_clicked()
{
    QString file_name = QFileDialog().getOpenFileName();
        if(!file_name.isNull()){
            QFile file(file_name);
            if(file.open(file.ReadOnly)){
                QString data  = file.readAll();
                 if (data == "") {
```

```
                    QMessageBox::critical(this, "Error!",
"P’PIPµPґPëC,Pµ PґPµCЂPµPIPs");
                    return;
                }
                else{
                    QStringList abc = data.split(' ');
                    int* mas = new int[100];
                    int i =0;
                    bool go = true;
                    for (auto x:abc){
                         bool convertOK;
                         x.toInt(&convertOK);
                        if(convertOK == false){
                             go = false;
                         }
                         else{
                           mas[i] = x.toInt();
                            i++;
                          }
                    }
                    if(!go){
                            QMessageBox::warning(this,"Error", "Incor-
rect input");
                            return;
                    }
                    BinTree* BT = new (BinTree);
                    for(int j = 0; j < i; j++){
                        BT->Head = BT->insert(BT->Head, mas[j]);
                    }
                    int depth = BT->max_depth(BT->Head);
                    graphic(BT, scene, depth);
                    }
                }


        }
        else QMessageBox::warning(this,"Error", "Not Found");
}
```

ПРИЛОЖЕНИЕ Г

ИСХОДНЫЙ КОД ПРОГРАММЫ. BIN_STRUCT.H

```
#ifndef BIN_STRUCT_H
#define BIN_STRUCT_H
#include <QGraphicsItem>
#include <QGraphicsView>
#include <QGraphicsEffect>
#include <QString>
#include <QFileDialog>
#include <QMessageBox>
#include <QTextEdit>
#include <QMainWindow>
#include <QStandardPaths>
#include <QtGui>
#include <QColorDialog>
#include <QInputDialog>
#include <QPushButton>
#include <QStringList>

struct Node // СЃС‚СЂСѓРєС‚СѓСЂР° РґР»СЏ РїСЂРµРґСЃС‚Р°РІР»РµРЅРёСЏ
СѓР·Р»РѕРІ РґРµСЂРµРІР°
{
    int key;
    int size;
    Node* left;
    Node* right;
    Node(int k) { key = k; left = right = 0; size = 1; }

};


class BinTree
{
private:
    Node* Current = nullptr;
public:
    Node* Head = nullptr;
    BinTree();
    Node* insert(Node* p, int k);
    Node* insertroot(Node* p, int k);
    Node* rotateright(Node* p);
    Node* rotateleft(Node* q);
    int max_depth(Node *hd);
    int getsize(Node* p);
    void fixsize(Node* p);
    Node* join(Node* p, Node* q);
    Node* remove(Node* p, int k);

};
int treePainter(QGraphicsScene *scene, Node *node, int w, int h, int
wDelta, int hDelta, QPen &pen, QBrush &brush, QFont &font, int depth,
int c);

QGraphicsScene *graphic(BinTree *tree, QGraphicsScene *scene, int
depht);
```

```
#endif // BIN_STRUCT_H
```

ПРИЛОЖЕНИЕ Д

ИСХОДНЫЙ КОД ПРОГРАММЫ. TREE_FUN.CPP

```
#include <bin_struct.h>

BinTree::BinTree(){
    Head = nullptr;
    Current = Head;
}
int BinTree::max_depth(Node *hd){
    if (hd == NULL) return 0;
    else{
        int lDepth = max_depth(hd->left);
        int rDepth = max_depth(hd->right);

        if (lDepth > rDepth)  return(lDepth + 1);
        else return(rDepth + 1);
    }
}
int BinTree::getsize(Node* p) // PsP±PµCЂC,PєP° PґP»CЦ PїPsP»CЦ size,
CЂP°P±PsC,P°PµC, CЃ PїCѓCЃC,C‹PjPë PґPµCЂPµPICЊCЦPjPë (t=NULL)
{
    if( !p ) return 0;
    return p->size;
}

void BinTree::fixsize(Node* p) // CѓCЃC,P°PSPsPIP»PµPSPëPµ
PєPsCЂCЂPµPєC,PSPsPiPs CЂP°P·PjPµCЂP° PґPµCЂPµPIP°
{
    p->size = getsize(p->left)+getsize(p->right)+1;
}

 Node* BinTree::rotateright(Node* p) // PїCЂP°PIC‹P№ PїPsPIPsCЂPsC,
PIPsPєCЂCѓPi CѓP·P»P° p
{
    Node* q = p->left;
    if( !q ) return p;
    p->left = q->right;
    q->right = p;
    q->size = p->size;
    fixsize(p);
    return q;
}
Node* BinTree::rotateleft(Node* q) // P»PµPIC‹P№ PїPsPIPsCЂPsC, PIPsPєCЂCѓPi
CѓP·P»P° q
{
    Node* p = q->right;
    if( !p ) return q;
    q->right = p->left;
    p->left = q;
    p->size = q->size;
    fixsize(q);
    return p;
}
Node* BinTree::insertroot(Node* p, int k) // PICЃC,P°PIPєP° PSPsPIPsPiPs
CѓP·P»P° CЃ PєP»CЂC‡PsPj k PI PєPsCЂPµPSCЊ PґPµCЂPµPIP° p
{
    if( !p ) return new Node(k);
    if( k < p->key )
    {
        p->left = insertroot(p->left,k);
```

```
        return rotateright(p);
    }
    else
    {
        p->right = insertroot(p->right,k);
        return rotateleft(p);
    }
}
Node* BinTree::insert(Node* p, int k) // СЂР°PSPrPsPjPёP·PёСЂPsPIP°PSPSP°СЦ
PICЌC,P°PIPєP° PSPsPIPsPiPs CґP·P»P° СЃ PєP»СЂC‡PsPj k PI PrPµСЂPµPIPs p
{
    if( !p ) return new Node(k);
    if( rand()%(p->size+1)==0 )
        return insertroot(p,k);
    if( p->key>k )
        p->left = insert(p->left,k);
    else
        p->right = insert(p->right,k);
    fixsize(p);
    return p;
}
Node*BinTree:: join(Node* p, Node* q) // PsP±СЉPµPrPёPSPµPSPёPµ PrPICґC…
PrPµСЂPµPICЊPµPI
{
    if( !p ) return q;
    if( !q ) return p;
    if( rand()%(p->size+q->size) < p->size )
    {
        p->right = join(p->right,q);
        fixsize(p);
        return p;
    }
    else
    {
        q->left = join(p,q->left);
        fixsize(q);
        return q;
    }
}
Node*BinTree::remove(Node* p, int k) // СґPrP°P»PµPSPёPµ PёP· PrPµСЂPµPIP° p
PїPµСЂPIPsPiPs PSP°PNºPrPµPSPSPsPiPs СґP·P»P° СЃ PєP»СЂC‡PsPj k
{
    if( !p ) return p;
    if( p->key==k )
    {
        Node* q = join(p->left,p->right);
        delete p;
        return q;
    }
    else if( k<p->key )
        p->left = remove(p->left,k);
    else
        p->right = remove(p->right,k);
    return p;
}
```

ПРИЛОЖЕНИЕ Е

ИСХОДНЫЙ КОД ПРОГРАММЫ. MAIN_FUN.H

```
#ifndef MAIN_FUN_H
#define MAIN_FUN_H
#include <bin_struct.h>


int treePainter(QGraphicsScene *scene, Node *node, int w, int h, int
wDelta, int hDelta, QPen &pen, QBrush &brush, QFont &font, int depth);

QGraphicsScene *graphic(BinTree *tree, QGraphicsScene *scene, int
depht);

#endif // MAIN_FUN_H
```

ПРИЛОЖЕНИЕ Ж

ИСХОДНЫЙ КОД ПРОГРАММЫ. DRAW_TREE.CPP

```
#include<bintree.h>
#include<functionstree.h>
#include<cmath>
QGraphicsScene *graphic(BinTree *tree, QGraphicsScene *&scene, int depth)
{
    if (tree == nullptr)
        return scene;
    scene->clear();
    QPen pen;
    QColor color;
    color.setRgb(220, 220, 220);
    pen.setColor(color);
    QBrush brush (color);
    QFont font;
    font.setFamily("Tahoma");
    pen.setWidth(3);
    int wDeep = static_cast<int>(pow(2, depth + 2));
    int hDelta = 70;
    int wDelta = 15;
    font.setPointSize(wDelta);
    int width = (wDelta*wDeep)/2;
    treePainter(scene, tree->Head, width/2, hDelta, wDelta, hDelta, pen, brush,
font, wDeep);
    return scene;
}

int treePainter(QGraphicsScene *&scene, Node *node, int w, int h, int wDelta,
int hDelta, QPen &pen, QBrush &brush, QFont &font, int depth)
{
    if ((node == nullptr) || (node->data == '^'))
        return 0;
    QString out;
    out += node->data;
    QGraphicsTextItem *textItem = new QGraphicsTextItem;
    textItem->setPos(w, h);
    textItem->setPlainText(out);
    textItem->setFont(font);
    scene->addEllipse(w-wDelta/2, h, wDelta*5/2, wDelta*5/2, pen, brush);
    if ((node->left != nullptr) && (node->left->data != '^') )
        scene->addLine(w+wDelta/2, h+wDelta, w-(depth/2)*wDelta+wDelta/2,
h+hDelta+wDelta, pen);
    if (node->right != nullptr)
        scene->addLine(w+wDelta/2, h+wDelta, w+(depth/2)*wDelta+wDelta/2,
h+hDelta+wDelta, pen);
    scene->addItem(textItem);
    treePainter(scene, node->left, w-(depth/2)*wDelta, h+hDelta, wDelta, hDelta,
pen, brush, font, depth/2);
    treePainter(scene, node->right, w+(depth/2)*wDelta, h+hDelta, wDelta,
hDelta, pen, brush, font, depth/2);
    return 0;
}
```