



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего
образования

«МИРЭА - Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий
Кафедра математического обеспечения и стандартизации
информационных технологий

ОТЧЕТ

ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 5

**«Сбалансированные деревья поиска и их применение для поиска данных в
файле»**

по дисциплине

«Структуры и алгоритмы обработки данных»

Выполнил студент группы *ИКБО-03-22*

Хохлинов Д.И.

Принял

Сорокин А.В.

Практическая
работа выполнена

«__»_____2023 г.

«Зачтено»

«__»_____2023 г.

Москва 2023

СОДЕРЖАНИЕ

1 ЗАДАНИЕ 1	3
1.1 Постановка задачи	3
1.2 Разработка приложения	3
1.2.1 Структура двоичного файла	3
1.2.2 Функционал приложения	4
1.2.3 Код программы	6
1.2.4 Тестирование	17
2 ЗАДАНИЕ 2	21
2.1 Постановка задачи	21
2.2 Разработка приложения	21
2.2.1 Структура двоичного файла	21
2.2.2 Функционал приложения	21
2.2.3 Код программы	24
2.2.4 Тестирование	42
3 ЗАДАНИЕ 3	46
3.1 Постановка задачи	46
3.2 Составление таблицы	46
4 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	49

1 ЗАДАНИЕ 1

1.1 Постановка задачи

Разработать приложение, которое использует бинарное дерево поиска (БДП) для поиска записи с ключом в файле, структура которого представлена ниже.

1. Разработать класс (или библиотеку функций) «Бинарное дерево поиска». Тип информационной части узла дерева: ключ и ссылка на запись в файле (как в практическом задании 2). Методы: включение элемента в дерево, поиск ключа в дереве, удаление ключа из дерева, отображение дерева.

2. Разработать класс (библиотеку функций) управления файлом (если не создали в практическом задании 2). Включить методы: создание двоичного файла записей фиксированной длины из заранее подготовленных данных в текстовом файле; поиск записи в файле с использованием БДП; остальные методы по вашему усмотрению.

3. Разработать и протестировать приложение.

4. Подготовить отчет

1.2 Разработка приложения

1.2.1 Структура двоичного файла

Для выполнения задания использовалась следующая структура:

```
struct record {  
    char phone[11];  
    char address[100];  
};
```

Количество памяти, занимаемое полями этой структуры:

- phone – 11 байт;
- address – 100 байт.

Итого один экземпляр занимает 111 байт.

Структура двоичного файла, используемого для хранения записей такого типа, изображена на рисунке 1.

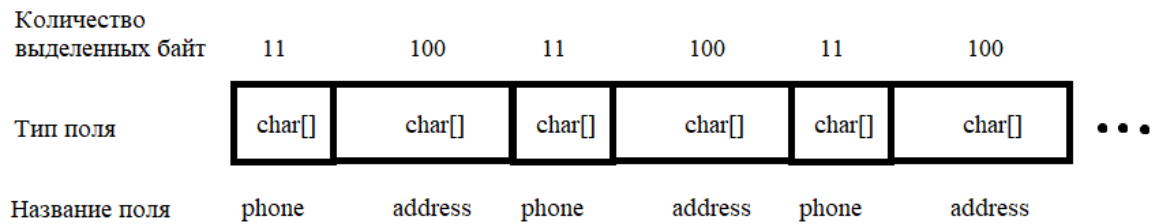


Рисунок 1 – Структура двоичного файла, используемого приложением

1.2.2 Функционал приложения

Для выполнения задания использовались следующие классы (листинг 1):

Листинг 1 – Используемые в задании 1 классы

```
class BinTreeNode {
    int64_t key;
    int recordNumber = -1;
    BinTreeNode* parent = nullptr;
    BinTreeNode* left = nullptr;
    BinTreeNode* right = nullptr;
public:
    BinTreeNode(BinTreeNode* parent, int64_t key, int recordNumber);
    int64_t getKey();
    int getRecordNumber();
    BinTreeNode* getParent();
    BinTreeNode* getLeft();
    BinTreeNode* getRight();
    void setKey(int64_t key);
    void setRecordNumber(int recordNumber);
    void setParent(BinTreeNode* parent);
    void setLeft(BinTreeNode* left);
    void setRight(BinTreeNode* right);
    void show(int level = 0);
    ~BinTreeNode();
};

class BinTree {
    BinTreeNode* root = nullptr;
public:
    BinTree(int64_t startValue, int startRN);
    void addElement(int64_t key, int RN);
    void removeElement(int64_t key);
    int findElement(int64_t key);
    void showTree();
};
```

Продолжение листинга 1

```
~BinTree();  
};
```

Узел `BinTreeNode` имеет информационную часть (ключ и номер записи в файле) и указатели на родителя, левого и правого потомка, а также методы:

- 1) Конструктор с аргументами – указателем на родителя (для корня – нулевой), ключом и номером записи;
- 2) Методы, возвращающие значения частных полей класса;
- 3) Методы, устанавливающие значения частных полей класса;
- 4) Метод для вывода текущего узла и его потомков на экран;
- 5) Деструктор – удаляет левого и правого потомка, а затем – и сам узел.

Класс `BinTree`, описывающий дерево, имеет указатель на корень и следующие методы:

- 1) Конструктор с аргументами – ключом и номером записи. Создает корневой элемент с заданной информационной частью;
- 2) Метод `addElement` – добавляет узел с заданной информационной частью в дерево (если узла с заданным ключом еще нет в дереве);
- 3) Метод `removeElement` – удаляет узел с заданным ключом из дерева (если такой узел есть в дереве);
- 4) Метод `findElement` – выполняет поиск узла с заданным ключом. Возвращает номер записи, записанный в узле, при успешном поиске, и 0 при неудачном.
- 5) Метод `showTree` – вызывает метод `show` корневого элемента, если он есть; в противном случае выводит соответствующее сообщение.
- 6) Деструктор – удаляет корневой элемент.

Для чтения двоичного файла используется модуль BinFileReader с следующими функциями:

- прочитать определенную запись файла;
- прочитать следующую запись файла;
- открыть файл;
- закрыть файл;
- открыть файл повторно;
- инициализация модуля.

1.2.3 Код программы

Запишем код программы на языке C++ (листинги 2-7):

Листинг 2 – Record.h

```
#pragma once
#ifndef __RECORD_H__
#define __RECORD_H__
#include <iostream>
#include <string>

struct record {
    char phone[11];
    char address[100];
    record();
    record(char* phone, char* address);
    bool operator == (record* right);
    bool operator != (record* right);
};
#endif
```

Листинг 3 – Record.cpp

```
#include "Record.h"

record::record()
{
    for (int i = 0; i < 11; i++)
    {
        this->phone[i] = 0;
        this->address[i] = 0;
    }
    for (int i = 11; i < 100; i++)
```

Продолжение листинга 3

```
{
    this->address[i] = 0;
}
strcpy_s(this->phone, "1234567890");
strcpy_s(this->address, "Generic address");
}

record::record(char* phone, char* address)
{
    for (int i = 0; i < 11; i++)
    {
        this->phone[i] = 0;
        this->address[i] = 0;
    }
    for (int i = 11; i < 100; i++)
    {
        this->address[i] = 0;
    }
    strcpy_s(this->phone, phone);
    strcpy_s(this->address, address);
}

bool record::operator==(record* right)
{
    if (strcmp(this->phone, right->phone)) return false;
    if (strcmp(this->address, right->address)) return false;
    return true;
}

bool record::operator!=(record* right)
{
    return !(*this == right);
}
```

Листинг 4 – BinTree.h

```
#pragma once
#ifndef __BIN_TREE_H__
#define __BIN_TREE_H__
#include <iostream>
#include <string>

using namespace std;

class BinTreeNode {
    int64_t key;
    int recordNumber = -1;
    BinTreeNode* parent = nullptr;
    BinTreeNode* left = nullptr;
    BinTreeNode* right = nullptr;
public:
    BinTreeNode(BinTreeNode* parent, int64_t key, int recordNumber);
    int64_t getKey();
    int getRecordNumber();
}
```

Продолжение листинга 4

```
    BinTreeNode* getParent();
    BinTreeNode* getLeft();
    BinTreeNode* getRight();
    void setKey(int64_t key);
    void setRecordNumber(int recordNumber);
    void setParent(BinTreeNode* parent);
    void setLeft(BinTreeNode* left);
    void setRight(BinTreeNode* right);
    void show(int level = 0);
    ~BinTreeNode();
};

class BinTree {
    BinTreeNode* root = nullptr;
public:
    BinTree(int64_t startValue, int startRN);
    void addElement(int64_t key, int RN);
    void removeElement(int64_t key);
    int findElement(int64_t key);
    void showTree();
    ~BinTree();
};
#endif
```

Листинг 5 – BinTree.h

```
#pragma once
#include "BinTree.h"

BinTreeNode::BinTreeNode(BinTreeNode* parent, int64_t key, int
recordNumber) :
parent(parent), key(key), recordNumber(recordNumber)
{

}

int64_t BinTreeNode::getKey() {
    return this->key;
}

int BinTreeNode::getRecordNumber() {
    return this->recordNumber;
}

BinTreeNode* BinTreeNode::getParent() {
    return this->parent;
}

BinTreeNode* BinTreeNode::getLeft() {
    return this->left;
}

BinTreeNode* BinTreeNode::getRight() {
    return this->right;
}
```


Продолжение листинга 5

```
}

void BinTreeNode::setKey(int64_t key)
{
    this->key = key;
}

void BinTreeNode::setRecordNumber(int recordNumber)
{
    this->recordNumber = recordNumber;
}

void BinTreeNode::setParent(BinTreeNode * parent)
{
    this->parent = parent;
}

void BinTreeNode::setLeft(BinTreeNode* left) {
    this->left = left;
    if (left)
        left->setParent(this);
}

void BinTreeNode::setRight(BinTreeNode* right) {
    this->right = right;
    if (right)
        right->setParent(this);
}

void BinTreeNode::show(int level) {
    if (this->right) {
        this->right->show(level + 1);
    }
    for (int i = 0; i < level; i++)
    {
        cout << "-----|";
    }
    cout << "< " << this->key << " >\n";
    if (this->left) {
        this->left->show(level + 1);
    }
}

BinTreeNode::~BinTreeNode() {
    if (this->left)
        delete this->left;
    if (this->right)
        delete this->right;
}

BinTree::BinTree(int64_t startValue, int startRN)
{
    this->root = new BinTreeNode(nullptr, startValue, startRN);
}
```

Продолжение листинга 5

```
void BinTree::addElement(int64_t key, int RN)
{
    if (!findElement(key))
    {
        if (this->root == nullptr) {
            this->root = new BinTreeNode(nullptr, key, RN);
        }
        else {
            BinTreeNode* cur = this->root;
            while (cur)
            {
                if (key < cur->getKey())
                {
                    if (cur->getLeft() == nullptr)
                    {
                        cur->setLeft(new BinTreeNode(cur, key,
RN));
                        break;
                    }
                    else {
                        cur = cur->getLeft();
                    }
                }
                else
                {
                    if (cur->getRight() == nullptr)
                    {
                        cur->setRight(new BinTreeNode(cur,
key, RN));
                        break;
                    }
                    else {
                        cur = cur->getRight();
                    }
                }
            }
        }
    }
    else
    {
        cout << "Элемент с ключом " << key << " уже существует в
дереве." << endl;
    }
}

void BinTree::removeElement(int64_t key)
{
    if (findElement(key))
    {
        BinTreeNode* cur = this->root;
        while (cur) {
            if (cur->getKey() == key)
            {
                BinTreeNode* par = cur->getParent();
                if (par) //если удаляемый узел - не корень
```

Продолжение листинга 5

```

        {
>getRight() == nullptr))
        {
            if ((cur->getLeft() == nullptr) && (cur-
        {
            if (par->getLeft() == cur)
                par->setLeft(nullptr);
            else
                par->setRight(nullptr);
            delete cur;
        }
        else if (cur->getRight() == nullptr)
        {
            if (par->getLeft() == cur)
                par->setLeft(cur->getLeft());
            else
                par->setRight(cur->getLeft());
            cur->setLeft(nullptr);
            delete cur;
        }
        else if (cur->getLeft() == nullptr)
        {
            if (par->getLeft() == cur)
                par->setLeft(cur->getRight());
            else
                par->setRight(cur->getRight());
            cur->setRight(nullptr);
            delete cur;
        }
        else {
            BinTreeNode* temp = cur->getRight();
            while (temp->getLeft())
                temp = temp->getLeft();
            int64_t tempKey = temp->getKey();
            int tempRN = temp->getRecordNumber();
            this->removeElement(tempKey);
            cur->setKey(tempKey);
            cur->setRecordNumber(tempRN);
        }
    }
    else { //если удаляемый узел - корень
        if ((cur->getLeft() == nullptr) && (cur-
>getRight() == nullptr))
        {
            this->root = nullptr;
            delete cur;
        }
        else if (cur->getRight() == nullptr)
        {
            this->root = cur->getLeft();
            this->root->setParent(nullptr);
            cur->setLeft(nullptr);
            delete cur;
        }
        else if (cur->getLeft() == nullptr)
        {

```

Продолжение листинга 5

```
        this->root = cur->getRight();
        this->root->setParent(nullptr);
        cur->setRight(nullptr);
        delete cur;
    }
    else {
        BinTreeNode* temp = cur->getRight();
        while (temp->getLeft())
            temp = temp->getLeft();
        int64_t tempKey = temp->getKey();
        int tempRN = temp->getRecordNumber();
        this->removeElement(tempKey);
        cur->setKey(tempKey);
        cur->setRecordNumber(tempRN);
    }
}
break;
}
else if (key < cur->getKey())
{
    cur = cur->getLeft();
}
else
{
    cur = cur->getRight();
}
}
else {
    cout << "Элемент с ключом " << key << " не существует в
дереве." << endl;
}
}

int BinTree::findElement(int64_t key)
{
    BinTreeNode* cur = this->root;
    bool found = false;
    while (cur)
    {
        if (cur->getKey() == key)
        {
            found = true;
            break;
        }
        else if (key < cur->getKey())
        {
            cur = cur->getLeft();
        }
        else
        {
            cur = cur->getRight();
        }
    }
    if (found)
```

Продолжение листинга 5

```
        {
            return cur->getRecordNumber();
        }
        return 0;
    }

void BinTree::showTree()
{
    if (this->root)
        this->root->show();
    else
        cout << "Дерево пусто." << endl;
}

BinTree::~BinTree()
{
    delete root;
}
```

Листинг 6 – BinFileReader.h

```
#pragma once
#ifndef __BIN_FILE_READER_H__
#define __BIN_FILE_READER_H__
#include <iostream>
#include <string>
#include <conio.h>
#include <fstream>
#include "Record.h"

using namespace std;

namespace BinFileEditor {
    fstream file;
    string _path;

    void closeFile()
    {
        if (file.good())
        {
            cout << "Ошибки ввода-вывода не обнаружены." << endl;
        }
        file.close();
    }

    bool openFile(string path)
    {
        file = fstream(path.c_str(), ios::in | ios::out |
ios::binary);
        if (!file)
        {
            closeFile();
            return false;
        }
    }
}
```

Продолжение листинга 6

```
        return true;
    }

    bool reopenFile()
    {
        file.close();
        return openFile(_path);
    }

    bool readByIndex(int index, record &res)
    {
        record temp;
        reopenFile();
        for (int i = 0; i < index; i++)
        {
            if (file.eof()) {
                return false;
            }
            file.read((char*)&temp, sizeof(record));
        }
        if (!strcmp(temp.address, "Generic address"))
        {
            return false;
        }
        res = temp;
        return true;
    }

    bool readNext(record &res)
    {
        record temp;
        if (!file.eof())
            file.read((char*)&temp, sizeof(record));
        else return false;
        if (!strcmp(temp.address, "Generic address"))
        {
            return false;
        }
        res = temp;
        return true;
    }

    bool init(string path)
    {
        _path = path;
        return openFile(path);
    }
}
#endif
```

Листинг 7 – main.cpp

```
#include <iostream>
#include <chrono>
#include "BinTree.h"
#include "BinFileReader.h"

using namespace std;

int main()
{
    setlocale(LC_ALL, "ru");
    BinFileEditor::init("test_input.txt");
    record temp;
    BinFileEditor::readNext(temp);
    BinTree tree = BinTree(stoull(string(temp.phone)), 1);
    int i = 2;
    while (BinFileEditor::readNext(temp))
    {
        if (strcmp(temp.phone, ""))
            tree.addElement(stoull(string(temp.phone)), i++);
    }
    bool running = true;
    char choose = ' ';
    int recordIndex = 0;
    int64_t key = 0;
    int pos;
    auto start = chrono::steady_clock::now();
    auto end = chrono::steady_clock::now();
    while (running) {
        cout << "Выберите действие: \n"
              << "[S] - показать бинарное дерево\n"
              << "[A] - добавить запись в дерево\n"
              << "[R] - удалить запись из дерева\n"
              << "[F] - найти запись по ключу с помощью дерева\n"
              << "[O] - найти запись по ключу с помощью дерева и
вывести ее на экран\n"
              << "[Q] - выйти\n";
        choose = _getch();
        switch (tolower(choose)) {
            case 'q':
                running = false;
                break;
            case 's':
                cout << "Бинарное дерево поиска:\n";
                tree.showTree();
                break;
            case 'a':
                cout << "Введите номер записи из файла, которую нужно
добавить в дерево: ";
                cin >> recordIndex;
                if (BinFileEditor::readByIndex(recordIndex, temp))
                {
                    tree.addElement(stoull(string(temp.phone)),
recordIndex);
                }
                else {
```

Продолжение листинга 7

```
        cout << "Записи с номером " << recordIndex << "
не существует в файле.\n";
    }
    break;
    case 'r':
        cout << "Введите номер телефона, запись с которым
нужно удалить из дерева: ";
        cin >> key;
        tree.removeElement(key);
        break;
    case 'f':
        cout << "Введите искомый номер телефона: ";
        cin >> key;
        pos = tree.findElement(key);
        if (pos > 0)
        {
            cout << "Искомый номер телефона найден в записи с
номером " << pos << "\n";
        }
        else
        {
            cout << "Искомый номер телефона не найден\n";
        }
        break;
    case 'o':
        cout << "Введите искомый номер телефона: ";
        cin >> key;
        start = chrono::steady_clock::now();
        pos = tree.findElement(key);
        end = chrono::steady_clock::now();
        if (pos > 0)
        {
            cout << "Искомый номер телефона найден в записи с
номером " << pos << "\n";
            BinFileEditor::readByIndex(pos, temp);
            cout << "Номер телефона: " << temp.phone <<
"\nАдрес: " << temp.address << "\n";
            cout << "Время поиска записи: " <<
chrono::duration_cast<chrono::microseconds> (end - start).count() << "
мкс\n";
        }
        else
        {
            cout << "Искомый номер телефона не найден\n";
            cout << "Время поиска записи: " <<
chrono::duration_cast<chrono::microseconds> (end - start).count() << "
мкс\n";
        }
        break;
    default:
        cout << "Неизвестное действие.\n";
    }
    if (running) {
        system("pause");
        system("cls");
    }
```


Продолжение листинга 7

```
    }  
  }  
  return 0;  
}
```

1.2.4 Тестирование

Проведем тестирование написанной программы. В качестве тестового файла использовался следующий двоичный файл (рисунок 2):

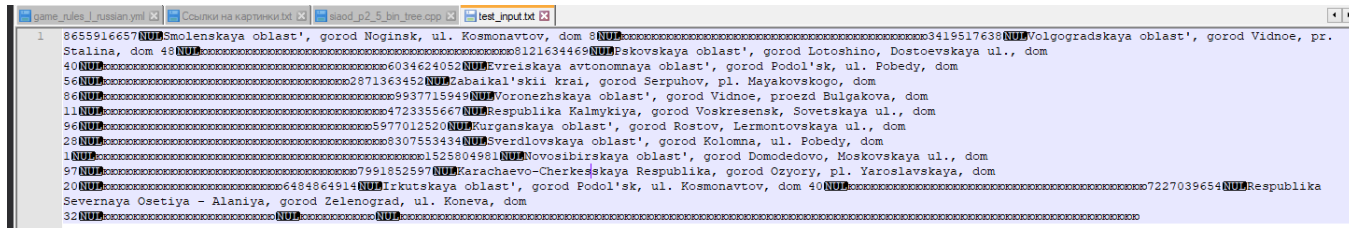


Рисунок 2 – Двоичный файл, используемый при тестировании

Ход тестирования приведен на рисунках 3-11:

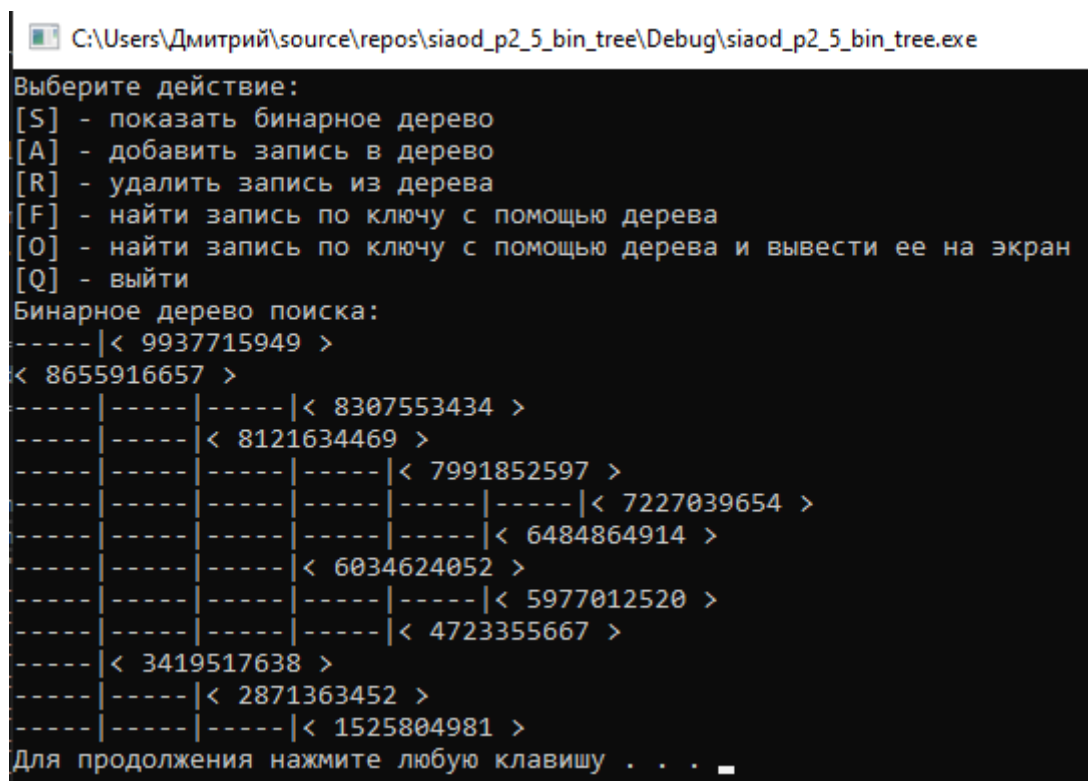


Рисунок 3 – Двоичное дерево поиска, построенное при считывании файла

```
C:\Users\Дмитрий\source\repos\siaod_p2_5_bin_tree\Debug\siaod_p2_5_bin_tree.exe
Выберите действие:
[S] - показать бинарное дерево
[A] - добавить запись в дерево
[R] - удалить запись из дерева
[F] - найти запись по ключу с помощью дерева
[O] - найти запись по ключу с помощью дерева и вывести ее на экран
[Q] - выйти
Введите номер телефона, запись с которым нужно удалить из дерева: 8121634469
Для продолжения нажмите любую клавишу . . .
```

Рисунок 4 – Удаление узла из двоичного дерева поиска

```
C:\Users\Дмитрий\source\repos\siaod_p2_5_bin_tree\Debug\siaod_p2_5_bin_tree.exe
Выберите действие:
[S] - показать бинарное дерево
[A] - добавить запись в дерево
[R] - удалить запись из дерева
[F] - найти запись по ключу с помощью дерева
[O] - найти запись по ключу с помощью дерева и вывести ее на экран
[Q] - выйти
Бинарное дерево поиска:
-----|< 9937715949 >
< 8655916657 >
-----|-----|< 8307553434 >
-----|-----|-----|< 7991852597 >
-----|-----|-----|-----|< 7227039654 >
-----|-----|-----|-----|< 6484864914 >
-----|-----|-----|< 6034624052 >
-----|-----|-----|-----|< 5977012520 >
-----|-----|-----|< 4723355667 >
-----|< 3419517638 >
-----|-----|< 2871363452 >
-----|-----|< 1525804981 >
Для продолжения нажмите любую клавишу . . .
```

Рисунок 5 – Дерево после удаления узла

```
C:\Users\Дмитрий\source\repos\siaod_p2_5_bin_tree\Debug\siaod_p2_5_bin_tree.exe
Выберите действие:
[S] - показать бинарное дерево
[A] - добавить запись в дерево
[R] - удалить запись из дерева
[F] - найти запись по ключу с помощью дерева
[O] - найти запись по ключу с помощью дерева и вывести ее на экран
[Q] - выйти
Введите номер записи из файла, которую нужно добавить в дерево: 3
Для продолжения нажмите любую клавишу . . .
```

Рисунок 6 – Добавление существующей записи в дерево (в данном случае была добавлена удаленная выше запись)

```
C:\Users\Дмитрий\source\repos\siaod_p2_5_bin_tree\Debug\siaod_p2_5_bin_tree.exe

Выберите действие:
[S] - показать бинарное дерево
[A] - добавить запись в дерево
[R] - удалить запись из дерева
[F] - найти запись по ключу с помощью дерева
[O] - найти запись по ключу с помощью дерева и вывести ее на экран
[Q] - выйти

Бинарное дерево поиска:
-----|< 9937715949 >
< 8655916657 >
-----|-----|< 8307553434 >
-----|-----|-----|-----|< 8121634469 >
-----|-----|-----|-----|< 7991852597 >
-----|-----|-----|-----|-----|< 7227039654 >
-----|-----|-----|-----|-----|< 6484864914 >
-----|-----|-----|< 6034624052 >
-----|-----|-----|-----|< 5977012520 >
-----|-----|-----|-----|< 4723355667 >
-----|< 3419517638 >
-----|-----|< 2871363452 >
-----|-----|-----|< 1525804981 >
Для продолжения нажмите любую клавишу . . .
```

Рисунок 7 – Дерево после вставки узла

```
C:\Users\Дмитрий\source\repos\siaod_p2_5_bin_tree\Debug\siaod_p2_5_bin_tree.exe

Выберите действие:
[S] - показать бинарное дерево
[A] - добавить запись в дерево
[R] - удалить запись из дерева
[F] - найти запись по ключу с помощью дерева
[O] - найти запись по ключу с помощью дерева и вывести ее на экран
[Q] - выйти

Введите номер записи из файла, которую нужно добавить в дерево: 56
Записи с номером 56 не существует в файле.
Для продолжения нажмите любую клавишу . . .
```

Рисунок 8 – Попытка вставить несуществующую запись

```
C:\Users\Дмитрий\source\repos\siaod_p2_5_bin_tree\Debug\siaod_p2_5_bin_tree.exe
Выберите действие:
[S] - показать бинарное дерево
[A] - добавить запись в дерево
[R] - удалить запись из дерева
[F] - найти запись по ключу с помощью дерева
[O] - найти запись по ключу с помощью дерева и вывести ее на экран
[Q] - выйти
Введите номер телефона, запись с которым нужно удалить из дерева: 56
Элемент с ключом 56 не существует в дереве.
Для продолжения нажмите любую клавишу . . .
```

Рисунок 9 – Попытка удалить несуществующий узел

```
C:\Users\Дмитрий\source\repos\siaod_p2_5_bin_tree\Debug\siaod_p2_5_bin_tree.exe
Выберите действие:
[S] - показать бинарное дерево
[A] - добавить запись в дерево
[R] - удалить запись из дерева
[F] - найти запись по ключу с помощью дерева
[O] - найти запись по ключу с помощью дерева и вывести ее на экран
[Q] - выйти
Введите искомый номер телефона: 56
Искомый номер телефона не найден
Время поиска записи: 1 мкс
Для продолжения нажмите любую клавишу . . .
```

Рисунок 10 – Поиск несуществующего узла

```
C:\Users\Дмитрий\source\repos\siaod_p2_5_bin_tree\Debug\siaod_p2_5_bin_tree.exe
Выберите действие:
[S] - показать бинарное дерево
[A] - добавить запись в дерево
[R] - удалить запись из дерева
[F] - найти запись по ключу с помощью дерева
[O] - найти запись по ключу с помощью дерева и вывести ее на экран
[Q] - выйти
Введите искомый номер телефона: 8121634469
Искомый номер телефона найден в записи с номером 3
Номер телефона: 8121634469
Адрес: Pskovskaya oblast', gorod Lotoshino, Dostoevskaya ul., dom 40
Время поиска записи: 2 мкс
Для продолжения нажмите любую клавишу . . .
```

Рисунок 11 – Успешный поиск

По результатам тестирования приложение работает верно, следовательно, использованные алгоритмы являются правильными.

2 ЗАДАНИЕ 2

2.1 Постановка задачи

Разработать приложение, которое использует сбалансированное дерево поиска, предложенное в варианте, для доступа к записям файла.

1. Разработать класс СДП с учетом дерева варианта. Структура информационной части узла дерева включает ключ и ссылку на запись в файле (адрес места размещения). Основные методы: включение элемента в дерево; поиск ключа в дереве с возвратом ссылки; удаление ключа из дерева; вывод дерева в форме дерева (с отображением структуры дерева).

2. Разработать приложение, которое создает и управляет СДП в соответствии с заданием.

3. Выполнить тестирование.

4. Определить среднее число выполненных поворотов (число поворотов на общее число вставленных ключей) при включении ключей в дерево при формировании дерева из двоичного файла.

5. Оформить отчет

2.2 Разработка приложения

2.2.1 Структура двоичного файла

Структура двоичного файла такая же, как и в задании 1 (см. пункт 1.2.1).

2.2.2 Функционал приложения

Для выполнения задания использовались следующие классы (листинг 8):

Листинг 8 – Используемые в задании 2 классы

```
enum RedBlackTreeColor { Black = 0, Red = 1 };  
  
class RedBlackTreeNode {  
    int64_t key;  
    int recordNumber;  
    RedBlackTreeNode* parent;  
    RedBlackTreeNode* left = nullptr;
```

Продолжение листинга 8

```
    RedBlackTreeNode* right = nullptr;
    RedBlackTreeColor color;
public:
    void invertSelf();

    RedBlackTreeNode(RedBlackTreeNode* parent, int64_t key, int
recordNumber, RedBlackTreeColor color);

    void clearParent(); //sets parent to nullptr
    void setLeft(RedBlackTreeNode* left);
    void setRight(RedBlackTreeNode* right);
    void setColor(RedBlackTreeColor color);
    void setKey(int64_t key);
    void setRN(int RN);

    RedBlackTreeNode* getLeft();
    RedBlackTreeNode* getRight();
    RedBlackTreeNode* getParent();
    RedBlackTreeColor getColor();
    int64_t getKey();
    int getRN();

    void show(int level = 0);

    ~RedBlackTreeNode();
};

class RedBlackTree {
    RedBlackTreeNode* root;

    void leftRotate(RedBlackTreeNode* pivot);
    void rightRotate(RedBlackTreeNode* pivot);

    void checkAddConditions(RedBlackTreeNode* startNode);
    void checkRemoveConditions(RedBlackTreeNode* startNode, bool
deletedIsLeft);
    void checkRoot();

    int rotations = 0;
    int nodes = 0;
public:
    RedBlackTree(RedBlackTreeNode* root);
    void addElement(int64_t key, int RN);
    void removeElement(int64_t key);
    int findElement(int64_t key);
    void showTree();
    ~RedBlackTree();
};
```

RedBlackTreeColor представляет собой перечисление двух используемых цветов: черного и красного.

Узел `RedBlackTreeNode` имеет информационную часть, идентичную таковой у узла из задания 1, а именно – ключ и номер записи. Методы для взаимодействия с узлом таковы:

- 1) Конструктор с аргументами – указатель на родителя, ключ, номер записи, цвет;
- 2) Метод `invertSelf()` – меняет цвет узла на противоположный (черный узел станет красным, и наоборот);
- 3) Метод `clearParent()` – устанавливает указатель на родителя равным `nullptr`;
- 4) Методы для получения значений полей класса;
- 5) Методы для установки значений полей класса;
- 6) Метод для вывода текущего узла и его потомков на экран;
- 7) Деструктор – удаляет левого и правого потомка узла, а затем – и сам узел.

Класс `RedBlackTree` имеет указатель на корневой элемент и следующие методы, недоступные для вызова пользователем:

- 1) Левый/правый повороты относительно переданного в качестве аргумента узла;
- 2) Методы проверки свойств красно-черного дерева при вставке и удалении элемента;
- 3) Метод проверки корня на соответствие свойствам красно-черного дерева;

Методы, доступные пользователю:

1) Конструктор с аргументами – ключом и номером записи. Создает корневой элемент с заданной информационной частью и окрашивает его в черный цвет;

2) Метод `addElement` – добавляет узел с заданной информационной частью в дерево (если узла с заданным ключом еще нет в дереве), сохраняя свойства красно-черного дерева;

3) Метод `removeElement` – удаляет узел с заданным ключом из дерева (если такой узел есть в дереве), сохраняя свойства красно-черного дерева;

4) Метод `findElement` – выполняет поиск узла с заданным ключом. Возвращает номер записи, записанный в узле, при успешном поиске, и 0 при неудачном.

5) Метод `showTree` – вызывает метод `show` корневого элемента, если он есть; в противном случае выводит соответствующее сообщение.

6) Деструктор – удаляет корневой элемент.

Для чтения двоичного файла используется модуль `BinFileReader`, описанный в пункте 1.2.2.

2.2.3 Код программы

Запишем код программы на языке C++ (листинги 9-). Файлы `BinFileReader.h`, `Record.h` и `Record.cpp` аналогичны таковым в пункте 1.2.3.

Листинг 9 – `RedBlackTree.h`

```
#pragma once
#ifndef __RED_BLACK_TREE_H__
#define __RED_BLACK_TREE_H__
#include <iostream>
#include <string>
using namespace std;

enum RedBlackTreeColor { Black = 0, Red = 1 };

class RedBlackTreeNode {
    int64_t key;
    int recordNumber;
    RedBlackTreeNode* parent;
```


Продолжение листинга 9

```
    RedBlackTreeNode* left = nullptr;
    RedBlackTreeNode* right = nullptr;
    RedBlackTreeColor color;
public:
    void invertSelf();

    RedBlackTreeNode(RedBlackTreeNode* parent, int64_t key, int
recordNumber, RedBlackTreeColor color);

    void clearParent(); //sets parent to nullptr
    void setLeft(RedBlackTreeNode* left);
    void setRight(RedBlackTreeNode* right);
    void setColor(RedBlackTreeColor color);
    void setKey(int64_t key);
    void setRN(int RN);

    RedBlackTreeNode* getLeft();
    RedBlackTreeNode* getRight();
    RedBlackTreeNode* getParent();
    RedBlackTreeColor getColor();
    int64_t getKey();
    int getRN();

    void show(int level = 0);

    ~RedBlackTreeNode();
};

class RedBlackTree {
    RedBlackTreeNode* root;

    void leftRotate(RedBlackTreeNode* pivot);
    void rightRotate(RedBlackTreeNode* pivot);

    void checkAddConditions(RedBlackTreeNode* startNode);
    void checkRemoveConditions(RedBlackTreeNode* startNode, bool
deletedIsLeft);
    void checkRoot();

    int rotations = 0;
    int nodes = 0;
public:
    RedBlackTree(RedBlackTreeNode* root);
    void addElement(int64_t key, int RN);
    void removeElement(int64_t key);
    int findElement(int64_t key);
    void showTree();
    ~RedBlackTree();
};
#endif
```

Листинг 10 – RedBlackTree.cpp

```
#include "RedBlackTree.h"

void RedBlackTreeNode::invertSelf()
{
    if (this->color == RedBlackTreeColor::Red)
        this->color = RedBlackTreeColor::Black;
    else
        this->color = RedBlackTreeColor::Red;
}

RedBlackTreeNode::RedBlackTreeNode(RedBlackTreeNode* parent, int64_t
key, int recordNumber, RedBlackTreeColor color)
{
    this->parent = parent;
    this->key = key;
    this->recordNumber = recordNumber;
    this->color = color;
}

void RedBlackTreeNode::clearParent()
{
    this->parent = nullptr;
}

void RedBlackTreeNode::setLeft(RedBlackTreeNode* left)
{
    this->left = left;
    if (left)
        left->parent = this;
}

void RedBlackTreeNode::setRight(RedBlackTreeNode* right)
{
    this->right = right;
    if (right)
        right->parent = this;
}

void RedBlackTreeNode::setColor(RedBlackTreeColor color)
{
    this->color = color;
}

void RedBlackTreeNode::setKey(int64_t key) {
    this->key = key;
}

void RedBlackTreeNode::setRN(int RN) {
    this->recordNumber = RN;
}

RedBlackTreeNode* RedBlackTreeNode::getLeft() {
    return this->left;
}
```

Продолжение листинга 10

```
RedBlackTreeNode* RedBlackTreeNode::getRight() {
    return this->right;
}

RedBlackTreeNode* RedBlackTreeNode::getParent() {
    return this->parent;
}

RedBlackTreeColor RedBlackTreeNode::getColor() {
    return this->color;
}

int64_t RedBlackTreeNode::getKey() {
    return this->key;
}

int RedBlackTreeNode::getRN() {
    return this->recordNumber;
}

void RedBlackTreeNode::show(int level)
{
    if (this->right) {
        this->right->show(level + 1);
    }
    for (int i = 0; i < level; i++)
    {
        cout << "-----|";
    }
    cout << "< " << this->key << " > ";
    if (this->color == RedBlackTreeColor::Black) {
        cout << "Black\n";
    }
    else {
        cout << "Red\n";
    }
    if (this->left) {
        this->left->show(level + 1);
    }
}

RedBlackTreeNode::~~RedBlackTreeNode() {
    if (this->left)
        delete left;
    if (this->right)
        delete right;
}

RedBlackTree::RedBlackTree(RedBlackTreeNode* root) {
    this->root = root;
}

RedBlackTree::~~RedBlackTree() {
    delete root;
}
```

Продолжение листинга 10

```
}

void RedBlackTree::showTree() {
    if (root)
    {
        root->show();
        cout << "Среднее число поворотов (повороты / число узлов): "
<< (double)rotations / nodes << endl;
    }
    else
        cout << "Дерево пусто.\n";
}

void RedBlackTree::leftRotate(RedBlackTreeNode* pivot) {
    rotations++;
    RedBlackTreeNode* pivotParent = pivot->getParent();
    RedBlackTreeNode* temp = pivot->getRight();
    pivot->setRight(temp->getLeft());
    if (temp)
        temp->setLeft(pivot);
    if (pivotParent)
    {
        bool isLeft = (pivotParent->getLeft() == pivot);
        if (isLeft)
            pivotParent->setLeft(temp);
        else
            pivotParent->setRight(temp);
    }
    else {
        temp->clearParent();
        this->root = temp;
    }
}

void RedBlackTree::rightRotate(RedBlackTreeNode* pivot) {
    rotations++;
    RedBlackTreeNode* pivotParent = pivot->getParent();
    RedBlackTreeNode* temp = pivot->getLeft();
    pivot->setLeft(temp->getRight());
    if (temp)
        temp->setRight(pivot);
    if (pivotParent)
    {
        bool isLeft = (pivotParent->getLeft() == pivot);
        if (isLeft)
            pivotParent->setLeft(temp);
        else
            pivotParent->setRight(temp);
    }
    else {
        temp->clearParent();
        this->root = temp;
    }
}
```

Продолжение листинга 10

```
int RedBlackTree::findElement(int64_t key)
{
    RedBlackTreeNode* cur = this->root;
    bool found = false;
    while (cur)
    {
        if (cur->getKey() == key)
        {
            found = true;
            break;
        }
        else if (key < cur->getKey())
        {
            cur = cur->getLeft();
        }
        else
        {
            cur = cur->getRight();
        }
    }
    if (found)
    {
        return cur->getRN();
    }
    return 0;
}

void RedBlackTree::addElement(int64_t key, int RN) {
    if (!findElement(key))
    {
        if (this->root == nullptr) {
            this->root = new RedBlackTreeNode(nullptr, key, RN,
RedBlackTreeColor::Black);
        }
        else {
            RedBlackTreeNode* cur = this->root;
            while (cur)
            {
                if (key < cur->getKey())
                {
                    if (cur->getLeft() == nullptr)
                    {
                        cur->setLeft(new RedBlackTreeNode(cur,
key, RN,
                                RedBlackTreeColor::Red));
                        checkAddConditions(cur->getLeft());
                        break;
                    }
                    else {
                        cur = cur->getLeft();
                    }
                }
                else
                {
                    if (cur->getRight() == nullptr)

```

Продолжение листинга 10

```
        {
            cur->setRight(new
RedBlackTreeNode(cur, key, RN,
                    RedBlackTreeColor::Red));
            checkAddConditions(cur->getRight());
            break;
        }
        else {
            cur = cur->getRight();
        }
    }
}
nodes++;
}
else {
    cout << "Элемент с ключом " << key << " уже существует в
дереве.\n";
}
}

void RedBlackTree::removeElement(int64_t key)
{
    if (findElement(key))
    {
        RedBlackTreeNode* cur = this->root;
        while (cur)
        {
            if (cur->getKey() == key)
                break;
            else if (key < cur->getKey())
                cur = cur->getLeft();
            else
                cur = cur->getRight();
        }
        bool isRed = (cur->getColor() == Red);
        RedBlackTreeNode* par = cur->getParent();
        bool isLeft = false;
        if (isRed)
        {
            if (par) //если удаляемый узел - не корень
            {
                isLeft = (par->getLeft() == cur);
                if ((cur->getLeft() == nullptr) && (cur-
>getRight() == nullptr))
                {
                    if (par->getLeft() == cur)
                        par->setLeft(nullptr);
                    else
                        par->setRight(nullptr);
                    delete cur;
                }
                else if (cur->getRight() == nullptr)
                {
                    if (par->getLeft() == cur)
```

Продолжение листинга 10

```
        par->setLeft(cur->getLeft());
    else
        par->setRight(cur->getLeft());
    cur->setLeft(nullptr);
    delete cur;
}
else if (cur->getLeft() == nullptr)
{
    if (par->getLeft() == cur)
        par->setLeft(cur->getRight());
    else
        par->setRight(cur->getRight());
    cur->setRight(nullptr);
    delete cur;
}
else {
    RedBlackTreeNode* temp = cur->getRight();
    while (temp->getLeft())
        temp = temp->getLeft();
    int64_t tempKey = temp->getKey();
    int tempRN = temp->getRN();
    this->removeElement(tempKey);
    cur->setKey(tempKey);
    cur->setRN(tempRN);
}
}
else { //если удаляемый узел - корень
    if ((cur->getLeft() == nullptr) && (cur->
>getRight() == nullptr))
    {
        this->root = nullptr;
        delete cur;
    }
    else if (cur->getRight() == nullptr)
    {
        this->root = cur->getLeft();
        this->root->clearParent();
        cur->setLeft(nullptr);
        delete cur;
    }
    else if (cur->getLeft() == nullptr)
    {
        this->root = cur->getRight();
        this->root->clearParent();
        cur->setRight(nullptr);
        delete cur;
    }
    else {
        RedBlackTreeNode* temp = cur->getRight();
        while (temp->getLeft())
            temp = temp->getLeft();
        int64_t tempKey = temp->getKey();
        int tempRN = temp->getRN();
        this->removeElement(tempKey);
        cur->setKey(tempKey);
```

Продолжение листинга 10

```

        cur->setRN(tempRN);
    }
}
else {
    if (par) //если удаляемый узел - не корень
    {
        isLeft = (par->getLeft() == cur);
        if ((cur->getLeft() == nullptr) && (cur-
>getRight() == nullptr))
        {
            if (par->getLeft() == cur)
                par->setLeft(nullptr);
            else
                par->setRight(nullptr);
            delete cur;
            checkRemoveConditions(par, isLeft);
        }
        else if (cur->getRight() == nullptr)
        {
            RedBlackTreeNode* temp = cur->getLeft();
            cur->setKey(temp->getKey());
            cur->setRN(temp->getRN());
            cur->setLeft(nullptr);
            delete temp;
        }
        else if (cur->getLeft() == nullptr)
        {
            RedBlackTreeNode* temp = cur->getRight();
            cur->setKey(temp->getKey());
            cur->setRN(temp->getRN());
            cur->setRight(nullptr);
            delete temp;
        }
        else {
            RedBlackTreeNode* temp = cur->getRight();
            while (temp->getLeft())
                temp = temp->getLeft();
            int64_t tempKey = temp->getKey();
            int tempRN = temp->getRN();
            this->removeElement(tempKey);
            cur->setKey(tempKey);
            cur->setRN(tempRN);
        }
    }
    else { //если удаляемый узел - корень
        if ((cur->getLeft() == nullptr) && (cur-
>getRight() == nullptr))
        {
            this->root = nullptr;
            delete cur;
        }
        else if (cur->getRight() == nullptr)
        {
            RedBlackTreeNode* temp = cur->getLeft();

```


Продолжение листинга 10

```

        cur->setKey(temp->getKey());
        cur->setRN(temp->getRN());
        cur->setLeft(nullptr);
        delete temp;
    }
    else if (cur->getLeft() == nullptr)
    {
        RedBlackTreeNode* temp = cur->getRight();
        cur->setKey(temp->getKey());
        cur->setRN(temp->getRN());
        cur->setRight(nullptr);
        delete temp;
    }
    else {
        RedBlackTreeNode* temp = cur->getRight();
        while (temp->getLeft())
            temp = temp->getLeft();
        int64_t tempKey = temp->getKey();
        int tempRN = temp->getRN();
        this->removeElement(tempKey);
        cur->setKey(tempKey);
        cur->setRN(tempRN);
    }
}

}

}
else {
    cout << "Элемент с ключом " << key << " не существует в
дереве.\n";
}
}

void RedBlackTree::checkAddConditions(RedBlackTreeNode* startNode)
{
    if (!startNode) return;
    RedBlackTreeNode* parent = startNode->getParent();
    if (!parent)
    {
        if (startNode->getColor() == Red)
            startNode->setColor(Black);
    }
    else {
        if (parent->getColor() == RedBlackTreeColor::Black) {
            return;
        }
        RedBlackTreeNode* gp = parent->getParent();
        if (gp)
        {
            RedBlackTreeNode* uncle = (gp->getLeft() == startNode-
>getParent()) ?
                gp->getRight() : gp->getLeft();
            if (uncle)
            {
                if (uncle->getColor() == RedBlackTreeColor::Red)

```

Продолжение листинга 10

```
gp->setColor(Red);
parent->setColor(Black);
uncle->setColor(Black);
checkAddConditions(gp);
}
else {
    bool parentIsLeft = gp->getLeft() == parent;
    bool nodeIsLeft = parent->getLeft() ==
startNode;

    if (parentIsLeft && nodeIsLeft)
    {
        rightRotate(gp);
        gp->invertSelf();
        parent->invertSelf();
    }
    else if (parentIsLeft && !nodeIsLeft)
    {
        leftRotate(parent);
        checkAddConditions(parent);
    }
    else if (!parentIsLeft && nodeIsLeft)
    {
        rightRotate(parent);
        checkAddConditions(parent);
    }
    else {
        leftRotate(gp);
        gp->invertSelf();
        parent->invertSelf();
    }
}
}
else {
    bool parentIsLeft = gp->getLeft() == parent;
    bool nodeIsLeft = parent->getLeft() == startNode;
    if (parentIsLeft && nodeIsLeft)
    {
        rightRotate(gp);
        gp->invertSelf();
        parent->invertSelf();
    }
    else if (parentIsLeft && !nodeIsLeft)
    {
        leftRotate(parent);
        rightRotate(gp);
        gp->invertSelf();
        startNode->invertSelf();
    }
    else if (!parentIsLeft && nodeIsLeft)
    {
        rightRotate(parent);
        leftRotate(gp);
        gp->invertSelf();
        startNode->invertSelf();
    }
}
```

Продолжение листинга 10

```
        else {
            leftRotate(gp);
            gp->invertSelf();
            parent->invertSelf();
        }
    }
}
else {
    parent->setColor(Black);
}
}
checkRoot();
}

void RedBlackTree::checkRemoveConditions(RedBlackTreeNode * startNode,
bool isLeft)
{
    if (!isLeft) //удаляем правого потомка
    {
        RedBlackTreeNode* left = startNode->getLeft();
        RedBlackTreeNode* left_left;
        RedBlackTreeNode* left_right;
        RedBlackTreeNode* left_right_left = nullptr;
        RedBlackTreeNode* left_right_right;
        RedBlackTreeColor left_color = Black,
            left_left_color = Black,
            left_right_color = Black,
            left_right_left_color = Black,
            left_right_right_color = Black;
        if (left)
        {
            left_color = left->getColor();
            left_left = left->getLeft();
            left_right = left->getRight();
            if (left_left)
            {
                left_left_color = left_left->getColor();
            }
            if (left_right)
            {
                left_right_color = left_right->getColor();
                left_right_left = left_right->getLeft();
                left_right_right = left_right->getRight();
                if (left_right_left)
                    left_right_left_color = left_right_left-
>getColor();
                if (left_right_right)
                    left_right_right_color = left_right_right-
>getColor();
            }
        }

        //case1: start red, start->left black, start->left-
>childs black
        if ((startNode->getColor() == Red) &&
            (left_color == Black) &&
```

Продолжение листинга 10

```
        (left_left_color == Black) &&
        (left_right_color == Black))
    {
        startNode->setColor(Black);
        left->setColor(Red);
    }
    //case2: start red, start->left black, start->left-
>left red
    else if ((startNode->getColor() == Red) &&
        (left_color == Black) &&
        (left_left_color == Red))
    {
        startNode->setColor(Black);
        left->setColor(Red);
        rightRotate(startNode);
    }
    //case2.5: start red, start->left black, start->left-
>right red
    else if ((startNode->getColor() == Red) &&
        (left_color == Black) &&
        (left_right_color == Red))
    {
        startNode->setColor(Black);
        //left_right->setColor(Black);
        leftRotate(left);
        rightRotate(startNode);
    }
    //case3: start black, start->left red, start->left-
>right->childs black
    else if ((startNode->getColor() == Black) &&
        (left_color == Red) &&
        (left_right_left_color == Black) &&
        (left_right_right_color == Black))
    {
        left->setColor(Black);
        left_right->setColor(Red);
        rightRotate(startNode);
    }
    //case4: start black, start->left red, start->left-
>right->left red
    else if ((startNode->getColor() == Black) &&
        (left_color == Red) &&
        (left_right_left_color == Red))
    {
        if (left_right_left)
            left_right_left->setColor(Black);
        leftRotate(left);
        rightRotate(startNode);
    }
    //case5: start black, start->left black, start->left-
>right red
    else if ((startNode->getColor() == Black) &&
        (left_color == Black) &&
        (left_right_color == Red))
    {
```

Продолжение листинга 10

```

        left_right->setColor(Black);
        leftRotate(left);
        rightRotate(startNode);
    }
    //case6: start black, start->left black, start->left-
>left red
    else if ((startNode->getColor() == Black) &&
        (left_color == Black) &&
        (left_left_color == Red))
    {
        left_left->setColor(Black);
        rightRotate(startNode);
    }
    //case7: all black
    else {
        left->setColor(Red);
        if (startNode->getParent())
            checkRemoveConditions(startNode->
>getParent(),
                                (startNode->getParent()->getRight() ==
startNode));
    }
    }
    else
    {
        //do nothing
    }
}
else { //удаляем левого потомка
    RedBlackTreeNode* right = startNode->getRight();
    RedBlackTreeNode* right_right;
    RedBlackTreeNode* right_left;
    RedBlackTreeNode* right_left_right = nullptr;
    RedBlackTreeNode* right_left_left;
    RedBlackTreeColor right_color = Black,
        right_right_color = Black,
        right_left_color = Black,
        right_left_right_color = Black,
        right_left_left_color = Black;
    if (right)
    {
        right_color = right->getColor();
        right_right = right->getRight();
        right_left = right->getLeft();
        if (right_right)
        {
            right_right_color = right_right->getColor();
        }
        if (right_left)
        {
            right_left_color = right_left->getColor();
            right_left_right = right_left->getRight();
            right_left_left = right_left->getLeft();
            if (right_left_right)

```

Продолжение листинга 10

```

        right_left_right_color = right_left_right-
>getColor();
        if (right_left_left)
            right_left_left_color = right_left_left-
>getColor();
    }

    //case1: start red, start->right black, start->right-
>childs black
    if ((startNode->getColor() == Red) &&
        (right_color == Black) &&
        (right_right_color == Black) &&
        (right_left_color == Black))
    {
        startNode->setColor(Black);
        right->setColor(Red);
    }
    //case2: start red, start->right black, start->right-
>right red
    else if ((startNode->getColor() == Red) &&
        (right_color == Black) &&
        (right_right_color == Red))
    {
        startNode->setColor(Black);
        right->setColor(Red);
        leftRotate(startNode);
    }
    //case2.5: start red, start->right black, start-
>right->left red
    else if ((startNode->getColor() == Red) &&
        (right_color == Black) &&
        (right_left_color == Red))
    {
        startNode->setColor(Black);
        //right_left->setColor(Black);
        rightRotate(right);
        leftRotate(startNode);
    }
    //case3: start black, start->right red, start->right-
>left->childs black
    else if ((startNode->getColor() == Black) &&
        (right_color == Red) &&
        (right_left_right_color == Black) &&
        (right_left_left_color == Black))
    {
        right->setColor(Black);
        right_left->setColor(Red);
        leftRotate(startNode);
    }
    //case4: start black, start->right red, start->right-
>left->right red
    else if ((startNode->getColor() == Black) &&
        (right_color == Red) &&
        (right_left_right_color == Red))
```

Продолжение листинга 10

```
{
    if (right_left_right)
        right_left_right->setColor(Black);
    rightRotate(right);
    leftRotate(startNode);
}
//case5: start black, start->right black, start-
>right->left red
else if ((startNode->getColor() == Black) &&
        (right_color == Black) &&
        (right_left_color == Red))
{
    right_left->setColor(Black);
    rightRotate(right);
    leftRotate(startNode);
}
//case6: start black, start->right black, start-
>right->right red
else if ((startNode->getColor() == Black) &&
        (right_color == Black) &&
        (right_right_color == Red))
{
    right_right->setColor(Black);
    leftRotate(startNode);
}
//case7: all black
else {
    right->setColor(Red);
    if (startNode->getParent())
        checkRemoveConditions(startNode->
>getParent(),
                                (startNode->getParent()->getRight() ==
startNode));
}
}
else
{
    //do nothing
}
}

void RedBlackTree::checkRoot()
{
    if (this->root->getColor() == RedBlackTreeColor::Red)
        this->root->invertSelf();
}
```

Листинг 11 – main.cpp

```
#include <iostream>
#include <chrono>
#include "RedBlackTree.h"
#include "BinFileReader.h"
using namespace std;

int main()
{
    setlocale(LC_ALL, "ru");

    BinFileEditor::init("test_input.txt");
    record temp;
    BinFileEditor::readNext(temp);
    RedBlackTree tree = RedBlackTree(new RedBlackTreeNode(nullptr,
stoi(string(temp.phone)), 1, Black));
    int i = 2;
    while (BinFileEditor::readNext(temp))
    {
        if (strcmp(temp.phone, ""))
        {
            tree.addElement(stoull(string(temp.phone)), i++);
        }
    }
    bool running = true;
    char choose = ' ';
    int recordIndex = 0;
    int64_t key = 0;
    int pos;
    auto start = chrono::steady_clock::now();
    auto end = chrono::steady_clock::now();
    while (running)
    {
        cout << "Выберите действие: \n"
        << "[S] - показать бинарное дерево\n"
        << "[A] - добавить запись в дерево\n"
        << "[R] - удалить запись из дерева\n"
        << "[F] - найти запись по ключу с помощью дерева\n"
        << "[O] - найти запись по ключу с помощью дерева и
вывести ее на экран\n"
        << "[Q] - выйти\n";
        choose = _getch();
        switch (tolower(choose)) {
            case 'q':
                running = false;
                break;
            case 's':
                cout << "Бинарное дерево поиска:\n";
                tree.showTree();
                break;
            case 'a':
                cout << "Введите номер записи из файла, которую нужно
добавить в дерево: ";
                cin >> recordIndex;
                if ((BinFileEditor::readByIndex(recordIndex, temp)) &&
(strcmp(temp.phone, "")))
```


Продолжение листинга 11

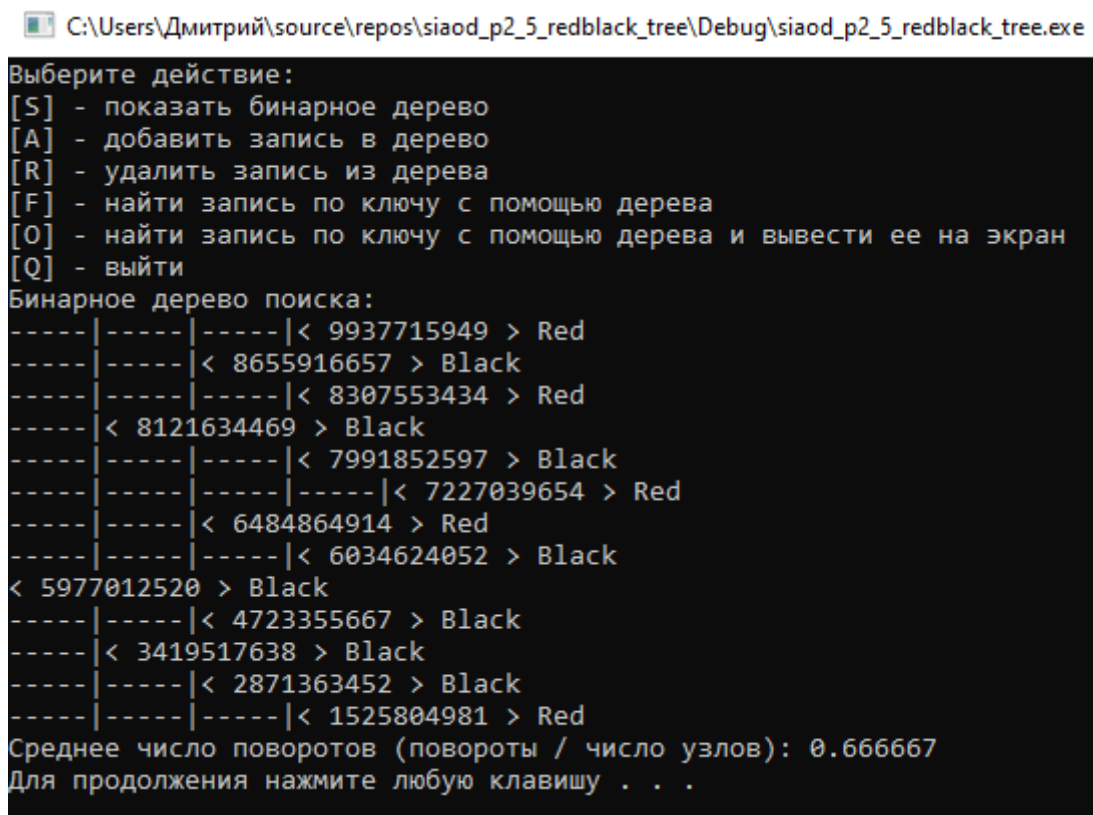
```
{
    tree.addElement(stoull(string(temp.phone)),
recordIndex);
}
else {
    cout << "Записи с номером " << recordIndex << "
не существует в файле.\n";
}
break;
case 'r':
    cout << "Введите номер телефона, запись с которым
нужно удалить из дерева: ";
    cin >> key;
    tree.removeElement(key);
    break;
case 'f':
    cout << "Введите искомый номер телефона: ";
    cin >> key;
    pos = tree.findElement(key);
    if (pos > 0)
    {
        cout << "Искомый номер телефона найден в записи с
номером " << pos << "\n";
    }
    else
    {
        cout << "Искомый номер телефона не найден\n";
    }
    break;
case 'o':
    cout << "Введите искомый номер телефона: ";
    cin >> key;
    start = chrono::steady_clock::now();
    pos = tree.findElement(key);
    end = chrono::steady_clock::now();
    if (pos > 0)
    {
        cout << "Искомый номер телефона найден в записи с
номером " << pos << "\n";
        BinFileEditor::readByIndex(pos, temp);
        cout << "Номер телефона: " << temp.phone <<
"\nАдрес: " << temp.address << "\n";
        cout << "Время поиска записи: " <<
chrono::duration_cast<chrono::microseconds> (end - start).count() << "
мкс\n";
    }
    else
    {
        cout << "Искомый номер телефона не найден\n";
        cout << "Время поиска записи: " <<
chrono::duration_cast<chrono::microseconds> (end - start).count() << "
мкс\n";
    }
    break;
default:
```

Продолжение листинга 11

```
        cout << "Неизвестное действие.\n";
    }
    if (running) {
        system("pause");
        system("cls");
    }
}
}
```

2.2.4 Тестирование

Проведем тестирование программы, аналогичное предыдущему: выполняются те же действия, двоичный файл тот же (рисунки 12-20):



```
C:\Users\Дмитрий\source\repos\siaod_p2_5_redblack_tree\Debug\siaod_p2_5_redblack_tree.exe
Выберите действие:
[S] - показать бинарное дерево
[A] - добавить запись в дерево
[R] - удалить запись из дерева
[F] - найти запись по ключу с помощью дерева
[O] - найти запись по ключу с помощью дерева и вывести ее на экран
[Q] - выйти
Бинарное дерево поиска:
-----|-----|-----|< 9937715949 > Red
-----|-----|< 8655916657 > Black
-----|-----|-----|< 8307553434 > Red
-----|< 8121634469 > Black
-----|-----|-----|< 7991852597 > Black
-----|-----|-----|-----|< 7227039654 > Red
-----|-----|< 6484864914 > Red
-----|-----|-----|< 6034624052 > Black
< 5977012520 > Black
-----|-----|< 4723355667 > Black
-----|< 3419517638 > Black
-----|-----|< 2871363452 > Black
-----|-----|-----|< 1525804981 > Red
Среднее число поворотов (повороты / число узлов): 0.666667
Для продолжения нажмите любую клавишу . . .
```

Рисунок 12 – Красно-черное дерево, построенное на основе входного файла

```
C:\Users\Дмитрий\source\repos\siaod_p2_5_redblack_tree\Debug\siaod_p2_5_redblack_tree.exe
Выберите действие:
[S] - показать бинарное дерево
[A] - добавить запись в дерево
[R] - удалить запись из дерева
[F] - найти запись по ключу с помощью дерева
[O] - найти запись по ключу с помощью дерева и вывести ее на экран
[Q] - выйти
Введите номер телефона, запись с которым нужно удалить из дерева: 6484864914
Для продолжения нажмите любую клавишу . . . █
```

Рисунок 13 – Удаление узла из КЧД

```
C:\Users\Дмитрий\source\repos\siaod_p2_5_redblack_tree\Debug\siaod_p2_5_redblack_tree.exe
Выберите действие:
[S] - показать бинарное дерево
[A] - добавить запись в дерево
[R] - удалить запись из дерева
[F] - найти запись по ключу с помощью дерева
[O] - найти запись по ключу с помощью дерева и вывести ее на экран
[Q] - выйти
Бинарное дерево поиска:
-----|-----|-----|< 9937715949 > Red
-----|-----|< 8655916657 > Black
-----|-----|-----|< 8307553434 > Red
-----|< 8121634469 > Black
-----|-----|-----|< 7991852597 > Black
-----|-----|< 7227039654 > Red
-----|-----|-----|< 6034624052 > Black
< 5977012520 > Black
-----|-----|< 4723355667 > Black
-----|< 3419517638 > Black
-----|-----|< 2871363452 > Black
-----|-----|-----|< 1525804981 > Red
```

Рисунок 14 – КЧД после удаления узла

```
C:\Users\Дмитрий\source\repos\siaod_p2_5_redblack_tree\Debug\siaod_p2_5_redblack_tree.exe
Выберите действие:
[S] - показать бинарное дерево
[A] - добавить запись в дерево
[R] - удалить запись из дерева
[F] - найти запись по ключу с помощью дерева
[O] - найти запись по ключу с помощью дерева и вывести ее на экран
[Q] - выйти
Введите номер записи из файла, которую нужно добавить в дерево: 12
Для продолжения нажмите любую клавишу . . . █
```

Рисунок 15 – Вставка узла в КЧД

```
C:\Users\Дмитрий\source\repos\siaod_p2_5_redblack_tree\Debug\siaod_p2_5_redblack_tree.exe
Выберите действие:
[S] - показать бинарное дерево
[A] - добавить запись в дерево
[R] - удалить запись из дерева
[F] - найти запись по ключу с помощью дерева
[O] - найти запись по ключу с помощью дерева и вывести ее на экран
[Q] - выйти
Бинарное дерево поиска:
-----|-----|-----|< 9937715949 > Red
-----|-----|< 8655916657 > Black
-----|-----|-----|< 8307553434 > Red
-----|< 8121634469 > Black
-----|-----|-----|< 7991852597 > Black
-----|-----|< 7227039654 > Red
-----|-----|-----|-----|< 6484864914 > Red
-----|-----|-----|< 6034624052 > Black
< 5977012520 > Black
-----|-----|< 4723355667 > Black
-----|< 3419517638 > Black
-----|-----|< 2871363452 > Black
-----|-----|-----|< 1525804981 > Red
```

Рисунок 16 – КЧД после вставки узла

```
C:\Users\Дмитрий\source\repos\siaod_p2_5_redblack_tree\Debug\siaod_p2_5_redblack_tree.exe
Выберите действие:
[S] - показать бинарное дерево
[A] - добавить запись в дерево
[R] - удалить запись из дерева
[F] - найти запись по ключу с помощью дерева
[O] - найти запись по ключу с помощью дерева и вывести ее на экран
[Q] - выйти
Введите номер телефона, запись с которым нужно удалить из дерева: 1
Элемент с ключом 1 не существует в дереве.
Для продолжения нажмите любую клавишу . . .
```

Рисунок 17 – Попытка удаления несуществующего узла

```
C:\Users\Дмитрий\source\repos\siaod_p2_5_redblack_tree\Debug\siaod_p2_5_redblack_tree.exe
Выберите действие:
[S] - показать бинарное дерево
[A] - добавить запись в дерево
[R] - удалить запись из дерева
[F] - найти запись по ключу с помощью дерева
[O] - найти запись по ключу с помощью дерева и вывести ее на экран
[Q] - выйти
Введите номер записи из файла, которую нужно добавить в дерево: 111
Записи с номером 111 не существует в файле.
Для продолжения нажмите любую клавишу . . .
```

Рисунок 18 – Попытка вставки несуществующей записи

```
C:\Users\Дмитрий\source\repos\siaod_p2_5_redblack_tree\Debug\siaod_p2_5_redblack_tree.exe
Выберите действие:
[S] - показать бинарное дерево
[A] - добавить запись в дерево
[R] - удалить запись из дерева
[F] - найти запись по ключу с помощью дерева
[O] - найти запись по ключу с помощью дерева и вывести ее на экран
[Q] - выйти
Введите искомый номер телефона: 7227039654
Искомый номер телефона найден в записи с номером 13
Номер телефона: 7227039654
Адрес: Respublika Severnaya Osetiya - Alaniya, gorod Zelenograd, ul. Koneva, dom 32
Время поиска записи: 1 мкс
Для продолжения нажмите любую клавишу . . .
```

Рисунок 19 – Успешный поиск записи в дереве

```
C:\Users\Дмитрий\source\repos\siaod_p2_5_redblack_tree\Debug\siaod_p2_5_redblack_tree.exe
Выберите действие:
[S] - показать бинарное дерево
[A] - добавить запись в дерево
[R] - удалить запись из дерева
[F] - найти запись по ключу с помощью дерева
[O] - найти запись по ключу с помощью дерева и вывести ее на экран
[Q] - выйти
Введите искомый номер телефона: 1
Искомый номер телефона не найден
Время поиска записи: 1 мкс
Для продолжения нажмите любую клавишу . . .
```

Рисунок 20 – Неудачный поиск записи в дереве

По результатам тестирования приложение работает верно, следовательно, использованные алгоритмы являются правильными.

3 ЗАДАНИЕ 3

3.1 Постановка задачи

Выполнить анализ алгоритма поиска записи с заданным ключом при применении структур данных:

- хеш – таблица;
- бинарное дерево поиска;
- КЧД

Требования по выполнению задания

1. Протестировать на данных:

- а) небольшого объема – выполнено в первых двух заданиях;
- б) большого объема.

2. Построить хеш-таблицу из чисел файла.

3. Осуществить поиск введенного целого числа в двоичном дереве поиска, в сбалансированном дереве и в хеш-таблице. Оформить таблицу результатов.

4. Провести анализ алгоритма поиска ключа на исследованных поисковых структурах на основе данных, представленных в таблице.

5. Оформить отчет

3.2 Составление таблицы

Составим таблицу для сравнения эффективности различных структур данных (таблица 1).

В файле выполнялся поиск записей с следующими ключами:

- Начало файла: 9937715949 (запись 6);
- Середина файла: 3419848325;
- Конец файла: 9308531605 (запись 10000);

- Несуществующая запись: 8005553535.

Таблица 1 – Таблица результатов

Вид поисковой структуры	Количество загруженных элементов	№	Время поиска записи (мкс):			
			В начале файла	В середине файла	В конце файла	Не имеется в файле
Хэш- таблица	10000	1	7	44	49	64
		2	7	44	47	65
		3	7	48	54	63
		4	8	44	50	64
		5	7	45	50	64
Среднее значение			7,2	45	50	64
Бинарное дерево поиска	10000	1	1	3	3	4
		2	1	3	3	4
		3	1	3	2	4
		4	1	3	3	4
		5	1	3	3	4
Среднее значение			1	3	2,8	4
КЧД	10000	1	2	4	4	3
		2	2	4	3	3
		3	2	4	4	3
		4	2	3	4	3
		5	2	4	4	4
Среднее значение			2	3,8	3,8	3,2

По таблице результатов видно, что бинарное дерево поиска и КЧД имеют примерно одинаковую эффективность и показывают себя значительно лучше хэш-таблицы. Хэш-таблица не показала ожидаемую

эффективность, близкую к $O(1)$, так как используемая хэш-функция не является идеальной и стала причиной возникновения множества коллизий.

4 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Задание на самостоятельную работу: https://online-edu.mirea.ru/pluginfile.php?file=%2F1144100%2Fmod_assign%2Fintroattachment%2F0%2FСиАОД%20Самостоятельная%20работа%205%20%28сб%20алансированные%20деревья%20поиска%29.pdf&, дата обращения: 09.11.23
2. Структуры и алгоритмы обработки данных – Лекция 2.5: https://online-edu.mirea.ru/pluginfile.php?file=%2F1126225%2Fmod_folder%2Fcontent%2F0%2F%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D1%8B%20%D0%B8%20%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%8B%20%D0%BE%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8%20%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85%D0%9B%D0%B5%D0%BA%D1%86%D0%B8%D1%8F_2.5.pptx, дата обращения: 09.11.23
3. <https://habr.com/ru/companies/otus/articles/472040/>, дата обращения: 09.11.23
4. <https://habr.com/ru/companies/otus/articles/521034/>, дата обращения: 09.11.23