

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](#_Toc150382669)

[1.1 Постановка задачи 3](#_Toc150382670)

[1.2 Разработка приложения 3](#_Toc150382671)

[1.2.1 Структура двоичного файла 3](#_Toc150382672)

[1.2.2 Функционал приложения 4](#_Toc150382673)

[1.2.3 Код программы 6](#_Toc150382674)

[1.2.4 Тестирование 17](#_Toc150382675)

[2 ЗАДАНИЕ 2 21](#_Toc150382676)

[2.1 Постановка задачи 21](#_Toc150382677)

[2.2 Разработка приложения 21](#_Toc150382678)

[2.2.1 Структура двоичного файла 21](#_Toc150382679)

[2.2.2 Функционал приложения 21](#_Toc150382680)

[2.2.3 Код программы 24](#_Toc150382681)

[2.2.4 Тестирование 42](#_Toc150382682)

[3 ЗАДАНИЕ 3 46](#_Toc150382683)

[3.1 Постановка задачи 46](#_Toc150382684)

[3.2 Составление таблицы 46](#_Toc150382685)

[4 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 49](#_Toc150382686)

# **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.

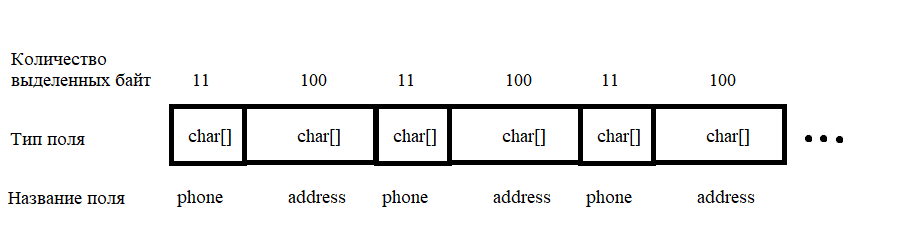


Рисунок 1 – Структура двоичного файла, используемого приложением

### **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 приватных полей класса;

3) Методы, устанавливающие значения приватных 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 С++ (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

{

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)

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):

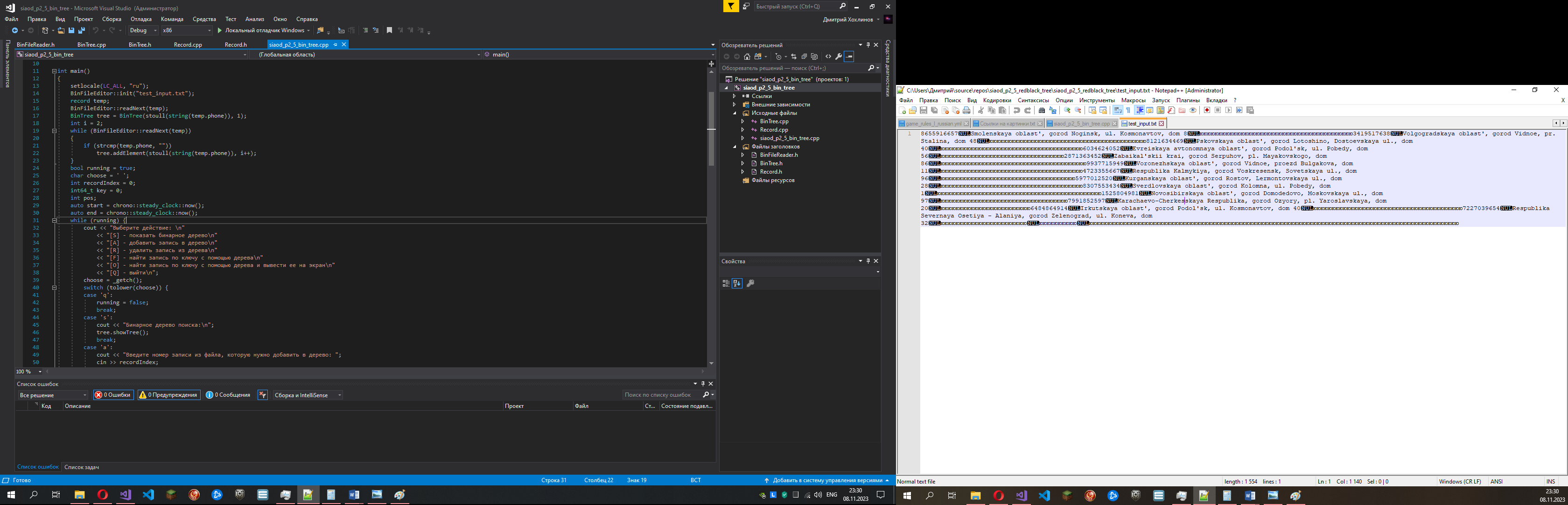


Рисунок 2 – Двоичный файл, используемый при тестировании

Ход тестирования приведен на рисунках 3-11:

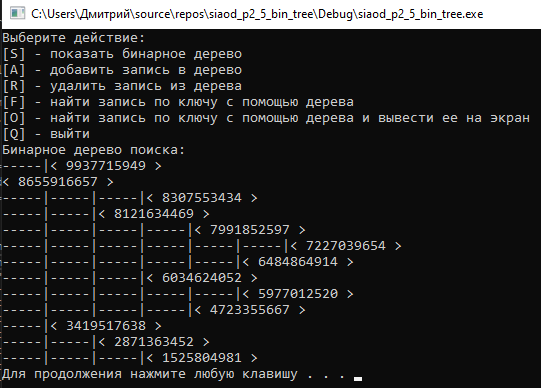


Рисунок 3 – Двоичное дерево поиска, построенное при считывании файла

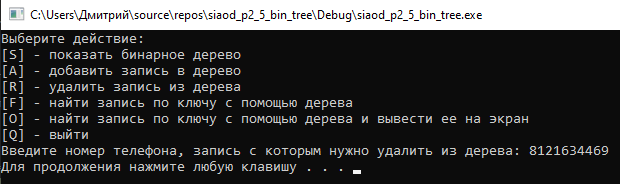


Рисунок 4 – Удаление узла из двоичного дерева поиска

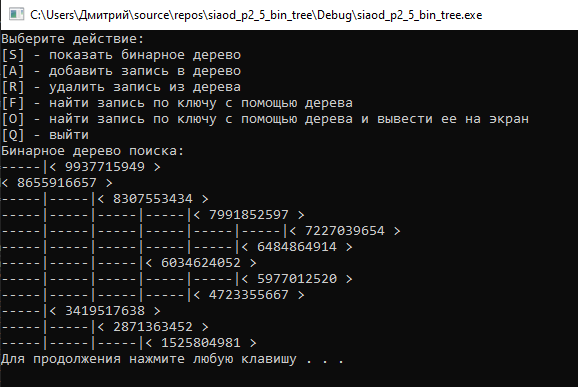


Рисунок 5 – Дерево после удаления узла

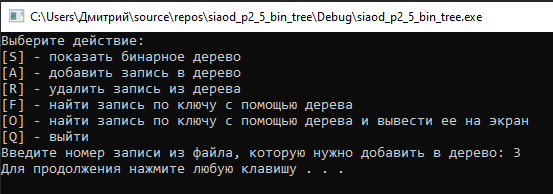


Рисунок 6 – Добавление существующей записи в дерево (в данном случае была добавлена удаленная выше запись)

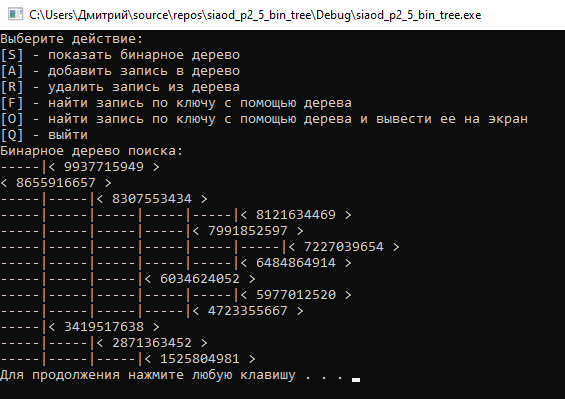


Рисунок 7 – Дерево после вставки узла

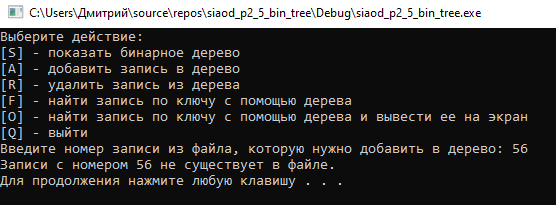


Рисунок 8 – Попытка вставить несуществующую запись

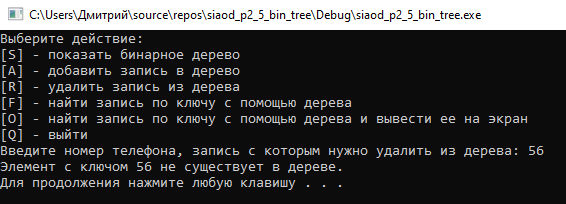


Рисунок 9 – Попытка удалить несуществующий узел

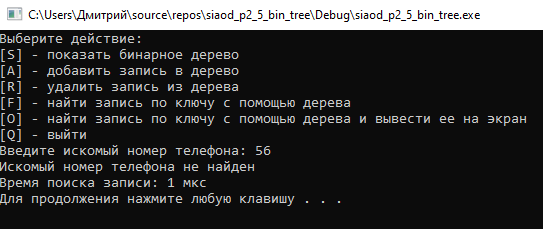


Рисунок 10 – Поиск несуществующего узла

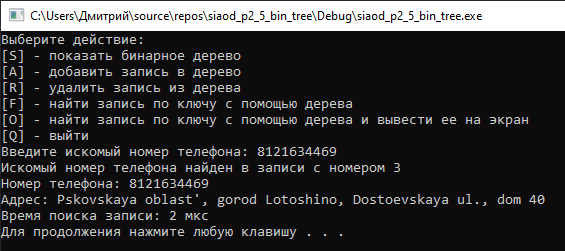


Рисунок 11 – Успешный поиск

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 С++ (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, stoull(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):

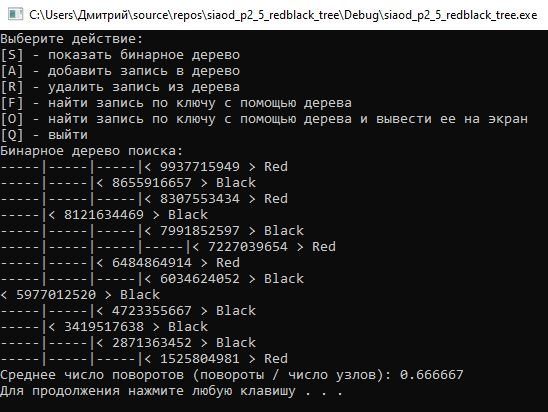


Рисунок 12 – Красно-черное дерево, построенное на основе входного файла

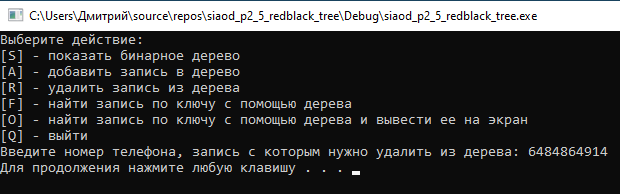


Рисунок 13 – Удаление узла из КЧД

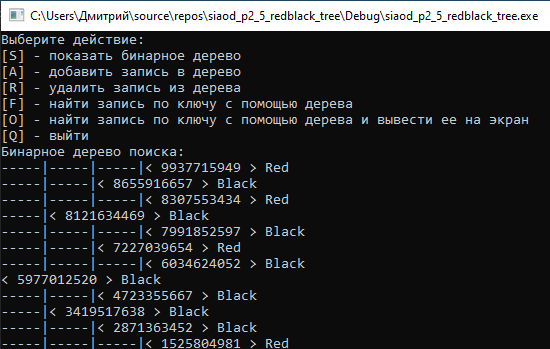


Рисунок 14 – КЧД после удаления узла

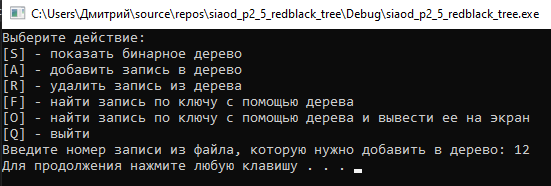


Рисунок 15 – Вставка узла в КЧД

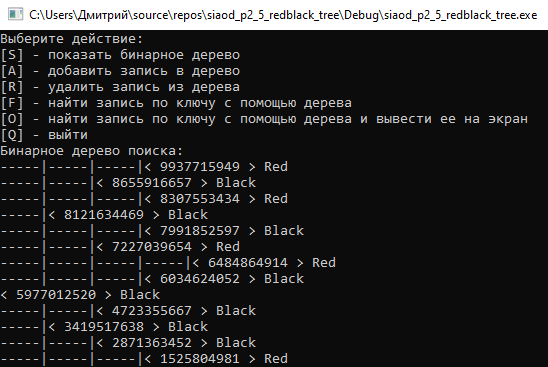


Рисунок 16 – КЧД после вставки узла

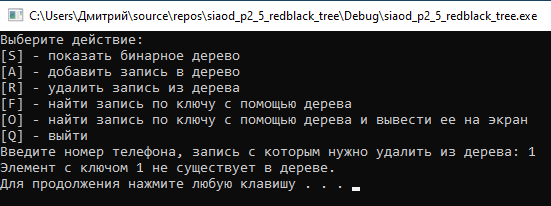


Рисунок 17 – Попытка удаления несуществующего узла

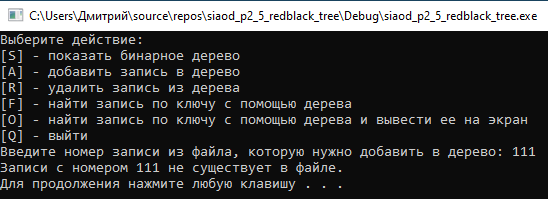


Рисунок 18 – Попытка вставки несуществующей записи

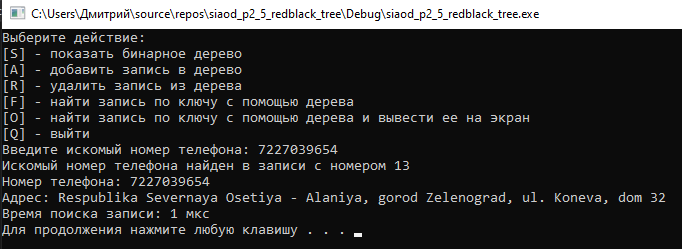


Рисунок 19 – Успешный поиск записи в дереве

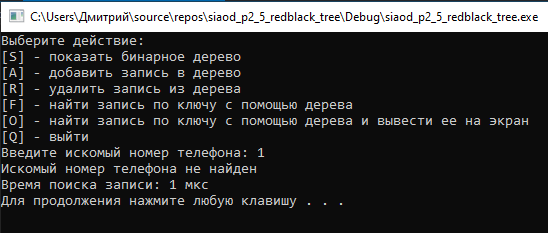


Рисунок 20 – Неудачный поиск записи в дереве

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 О(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%28сбалансированные%20деревья%20поиска%29.pdf&amp>, 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