МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ 7_1
по дисциплине
«Структуры и алгоритмы обработки данных»

Тема. Сбалансированные деревья поиска (СДП) и их применение для поиска данных в файле.

Выполнил студент группы ИКБО-60-23 Шеенко В.А

Принял старший преподаватель Скворцова Л.А.

Москва 2024

СОДЕРЖАНИЕ

1 ЦЕЛЬ РАБОТЫ

- Получить навыки в разработки и реализации алгоритмов управления бинарным деревом поиска и сбалансированными бинарными деревьями поиска (АВЛ – деревьями);
- Получить навыки в применении файловых потоков прямого доступа к данным файла;
- Получить навыки в применении сбалансированного дерева поиска для прямого доступа к записям файла.

2 ЗАДАНИЕ №1

2.1 Постановка задачи

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

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.

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.

Разработать и протестировать приложение.

2.2 Ход решения

Для решения поставленной задачи необходимо реализовать класс бинарного дерева, класс для работы с бинарным файлом и несколько структур, которые будут представлять данные записей и узлов.

Таблица 1 – Структура записи

Структура записи	Учет нарушений ПДД. Структура записи о нарушении ПДД: номер автомобиля, фамилия и инициалы владельца, модель, дата нарушения, место нарушения (текстом), статья (КоАП), наказание (сумма штрафа).

Данные для обработки, формата из задания

```
N038EK34;Kuznetsov P.P.;Nissan Teana;03.09.2024;ul. Lenina, d. 12;20.1
p.1;1200;
P743YA23;Kuznetsov K.K.;Honda Accord;28.09.2024;ul. Lenina, d. 12;12.5
p.1;1200;
S990RG77;Popov V.V.;Mercedes-Benz E200;07.09.2024;ul. Pushkina, d. 7;19.3
p.2;500;
D370JF50;Popov I.I.;BMW X5;28.09.2024;ul. Sovetskaya, d. 15;19.3 p.2;700;
K810RX23;Sidorov K.K.;Nissan Teana;13.09.2024;ul. Sovetskaya, d. 15;20.1
```

```
p.1;300;
K079CD50;Petrov V.V.;Honda Accord;06.09.2024;pr. Mira, d. 5;12.5 p.1;1200;
J997UO50;Ivanov K.K.;Nissan Teana;10.09.2024;pr. Mira, d. 5;19.3 p.2;1000;
K711DK23;Mikhailov A.A.;Nissan Teana;11.09.2024;ul. Pushkina, d. 7;19.3
p.2;300;
T735YE50;Fedorov S.S.;Honda Accord;22.09.2024;ul. Lenina, d. 12;12.9 p.1;500;
B597LW61;Popov V.V.;Nissan Teana;15.09.2024;ul. Sovetskaya, d. 15;12.9
p.1;1200;
R783NO50;Fedorov M.M.;Mercedes-Benz E200;17.09.2024;ul. Lenina, d. 12;12.5
p.1;300;
S108PK23;Smirnov S.S.;Lada Granta;19.09.2024;ul. Sovetskaya, d. 15;12.9
p.1;1200;
R963PF34;Mikhailov V.V.;Toyota Camry;20.09.2024;pr. Mira, d. 5;12.5 p.1;500;
T074LP61;Smirnov S.S.;BMW X5;06.09.2024;ul. Lenina, d. 12;12.8 p.2;300;
Q123GH99;Fedorov V.V.;Nissan Teana;19.09.2024;ul. Lenina, d. 12;12.5 p.1;300;
M809NL34;Sokolov A.A.;Mercedes-Benz E200;17.09.2024;ul. Pushkina, d. 7;20.1
p.1;500;
H336JT99;Sokolov V.V.;Honda Accord;04.09.2024;ul. Gagarina, d. 22;12.9
p.1;700;
K036ZY34;Mikhailov V.V.;Honda Accord;13.09.2024;pr. Mira, d. 5;12.9 p.1;700;
H055EU34;Petrov A.A.;Nissan Teana;12.09.2024;ul. Sovetskaya, d. 15;20.1
p.1;1200;
A715EL77;Kuznetsov I.I.;Nissan Teana;12.09.2024;ul. Lenina, d. 12;12.9
p.1;500;
```

Основные методы класса BinarySearchTree (класс для работы с бинарным деревом поиска, из прошлых практических):

1. Insert – вставка нового узла;
2. Find – поиск узла по ключу;
3. Remove – удаление узла по ключу;
   a) Способ удаления зависит от положения целевого узла, если целевой узел является листом дерево, то удаляется без специальных действий. Если целевой узел имеет только левое или правое поддерево, то узел удаляется, а поддерево присваивается родительскому узлу вместо целевого. Если целевой узел имеет и левое, и правое поддерево, то этот узел заменяется на минимальный узел из правого поддерева или на наибольший из левого поддерева.
4. PrintTree – вывод дерева в консоль.

BinFileWorker (класс для работы с бинарным файлом):

1. TranslateToBin – перевод из текстового формата в бинарный;
2. TranslateToTxt – перевод из бинарного формата в текстовый;

3. InitTreeByBin – создает бинарное дерево из данных бинарного файла;

4. PrintTree – вывод дерева, состоящего из данных бинарного файла, в консоль;

5. GetRecordByInd – получение записи по порядковому номеру из бинарного файла;

6. FindRecord – поиск записи по ключу с использованием бинарного дерева поиска;

7. PrintBin – вывод записей в консоль из бинарного файла.

2.3 Программная реализация

2.3.1 Структура ViolationRecord

Взято из прошлых практических.

```
#ifndef CODE_VIOLATIONRECORD_H
#define CODE_VIOLATIONRECORD_H

#include <vector>
#include <string>
#include <cstring>

inline std::vector<std::string> Split(std::string s, const std::string&
separator) {
    std::vector<std::string> tokens;
    size_t pos = 0;
    std::string token;

    while ((pos = s.find(separator)) != std::string::npos) {
        token = s.substr(0, pos);

        if (!token.empty())
            tokens.push_back(token);

        s.erase(0, pos + separator.length());
    }

    if (!s.empty())
        tokens.push_back(s);

    return tokens;
}

struct ViolationRecord {
    char carNumber[16];
    char name[32];
    char model[32];
    char data[32];
    char place[32];
    char article[32];
```

```
    short fine;

    ViolationRecord() = default;
    bool SetFieldsByStr(const std::string& s) {
        std::vector<std::string> tokens = Split(s, ";");

        if (tokens.size() != 7) {
            return false;
        }

        strcpy(carNumber, tokens[0].c_str());
        strcpy(name, tokens[1].c_str());
        strcpy(model, tokens[2].c_str());
        strcpy(data, tokens[3].c_str());
        strcpy(place, tokens[4].c_str());
        strcpy(article, tokens[5].c_str());
        fine = std::stoi(tokens[6]);

        return true;
    }

    std::string ToString() {
        std::string res;

        res = carNumber + std::string(";") + name + std::string(";") + model
+ std::string(";") +
              data + std::string(";") + place + std::string(";") + article +
std::string(";") +
              std::to_string(fine) + std::string(";");

        return res;
    }

    bool Validate() {
        return carNumber[0] != '\0' && name[0] != '\0' && model[0] != '\0' &&
data[0] != '\0' &&
               place[0] != '\0' && article[0] != '\0' && fine != 0;
    }
};

#endif //CODE_VIOLATIONRECORD_H
```

2.3.2 BinarySearchTree

BinarySearchTree.h:

```
#ifndef CODE_BINARYSEARCHTREE_H
#define CODE_BINARYSEARCHTREE_H

#include <cstddef>
#include <string>
#include <iostream>
#include <iomanip>

class BinarySearchTree {
private:
    struct Node {
        std::string key;
        size_t link_to_record = 0;
```

```
        Node* p_parent = nullptr;
        Node* p_left = nullptr;
        Node* p_right = nullptr;

        explicit Node(std::string s, size_t ind, Node* parent) :
key(std::move(s)),

link_to_record(ind),

p_parent(parent) {};

        ~Node() {
            if (!p_left) {
                delete p_left;
                p_left = nullptr;
            }

            if (!p_right) {
                delete p_right;
                p_right = nullptr;
            }

        }
    };

    Node* p_root = nullptr;

    void PrintHelper(Node* node, const std::string& prefix, bool isLeft, bool
isRoot = false);
    Node* FindMin(Node* node);
public:
    BinarySearchTree() = default;
    ~BinarySearchTree();

    void Insert(const std::string& key, size_t linkToRecord);
    bool Find(const std::string& key, size_t& found_ind) const;
    void Remove(const std::string& key);
    void PrintTree();
};


#endif //CODE_BINARYSEARCHTREE_H
```

BinarySearchTree.cpp:

```
#include "BinarySearchTree.h"

void BinarySearchTree::Insert(const std::string &key, size_t ind) {
    if (!p_root) {
        p_root = new Node(key, ind, nullptr);
        return;
    }

    Node* p_cur_node = p_root;
    Node* p_parent = p_cur_node;

    while (p_cur_node) {
        p_parent = p_cur_node;
        if (key < p_cur_node->key)
            p_cur_node = p_cur_node->p_left;
```

```
        else
            p_cur_node = p_cur_node->p_right;
    }


    if (key < p_parent->key)
        p_parent->p_left = new Node(key, ind, p_parent);
    else
        p_parent->p_right = new Node(key, ind, p_parent);
}


bool BinarySearchTree::Find(const std::string &key, size_t& found_ind) const
{
    Node* p_cur_node = p_root;

    while (p_cur_node && p_cur_node->key != key) {
        if (key < p_cur_node->key)
            p_cur_node = p_cur_node->p_left;
        else
            p_cur_node = p_cur_node->p_right;
    }

    if (p_cur_node)
        found_ind = p_cur_node->link_to_record;

    return p_cur_node;
}

BinarySearchTree::Node *BinarySearchTree::FindMin(BinarySearchTree::Node
*node) {
    while (node && node->p_left)
        node = node->p_left;

    return node;
}

void BinarySearchTree::Remove(const std::string &key) {
    Node* p_delete_node = p_root;

    if (p_root->key == key) {
        Node* p_min_node = FindMin(p_root->p_right);
        p_min_node->p_parent->p_left = nullptr;

        p_min_node->p_left = p_root->p_left;
        p_min_node->p_right = p_root->p_right;
        delete p_root;
        p_root = p_min_node;
        return;
    }

    while (p_delete_node && p_delete_node->key != key) {
        if (key < p_delete_node->key)
            p_delete_node = p_delete_node->p_left;
        else
            p_delete_node = p_delete_node->p_right;
    }

    if (!p_delete_node)
        return;
```

```
    if (!p_delete_node->p_left) {
        if (p_delete_node->p_parent->p_left == p_delete_node) {
            p_delete_node->p_parent->p_left = p_delete_node->p_right;
            p_delete_node->p_left->p_parent = p_delete_node->p_parent;
        } else {
            p_delete_node->p_parent->p_right = p_delete_node->p_right;
            p_delete_node->p_right->p_parent = p_delete_node->p_parent;
        }

        delete p_delete_node;
        return;
    }

    if (!p_delete_node->p_right) {
        if (p_delete_node->p_parent->p_left == p_delete_node) {
            p_delete_node->p_parent->p_left = p_delete_node->p_left;
            p_delete_node->p_left->p_parent = p_delete_node->p_parent;
        } else {
            p_delete_node->p_parent->p_right = p_delete_node->p_left;
            p_delete_node->p_left->p_parent = p_delete_node->p_parent;
        }

        delete p_delete_node;
        return;
    }

    Node* p_min_node = FindMin(p_delete_node->p_right);
    p_min_node->p_parent->p_left = nullptr;

    if (p_delete_node->p_parent->p_left == p_delete_node) {
        p_delete_node->p_parent->p_left = p_min_node;
    } else {
        p_delete_node->p_parent->p_right = p_min_node;
    }

    p_min_node->p_left = p_delete_node->p_left;
    p_delete_node->p_left->p_parent = p_min_node;

    p_min_node->p_right = p_delete_node->p_right;
    p_delete_node->p_right->p_parent = p_min_node;

    p_min_node->p_parent = p_delete_node->p_parent;
    delete p_delete_node;
}

void BinarySearchTree::PrintTree() {
    if (!this->p_root) {
        std::cout << "Empty tree\n";
        return;
    }

    PrintHelper(p_root, "", true, true);
}

void BinarySearchTree::PrintHelper(Node* node, const std::string& prefix,
bool isLeft, bool isRoot) {
    if (node != nullptr) {

        std::cout << prefix;
        if (isRoot)
            std::cout <<  "----";
```

```
        else
            std::cout << (isLeft ? "L---" : "|---");

        std::cout << node->key << std::endl;

        PrintHelper(node->p_right, prefix + (isLeft ? "    " : "|   "),
false);
        PrintHelper(node->p_left, prefix + (isLeft ? "    " : "|   "), true);
    }
}

BinarySearchTree::~BinarySearchTree() {
    delete p_root;
    p_root = nullptr;
}
```

2.3.3 BinFileWorker

BinFileWorker.h:

```
#ifndef CODE_BINFILEWORKER_H
#define CODE_BINFILEWORKER_H

#include <fstream>
#include <filesystem>
#include "ViolationRecord.h"
#include "BinarySearchTree.h"

namespace fs = std::filesystem;

class BinFileWorker {
private:
    fs::path bin_file_path;
    fs::path txt_file_path;

    BinFileWorker(const fs::path& txt_file_path, const fs::path&
bin_file_path);

public:
    BinarySearchTree* tree = nullptr;
    static BinFileWorker* CreateInstance(const fs::path& txt_file_path, const
fs::path& bin_file_path);
    static BinFileWorker* CreateInstance(const fs::path& txt_file_path);
    ~BinFileWorker();

    void TranslateToBin();
    void TranslateToTxt();
    void InitTreeByBin();
    void PrintTree();
    bool GetRecordByInd(size_t index, ViolationRecord& record);
    bool FindRecord(const std::string& car_number, ViolationRecord& record);
    void PrintBin();
    void RemoveInTree(const std::string& car_number) const;
    int FindInTree(const std::string& car_number) const;
    void AddToTree(std::string key, size_t ind);
};


#endif //CODE_BINFILEWORKER_H
```

BinFileWorker.cpp:

```
#include "BinFileWorker.h"


BinFileWorker *BinFileWorker::CreateInstance(const fs::path& txt_file_path,
const fs::path& bin_file_path) {
    if (!fs::exists(txt_file_path))
        return nullptr;

    if (!fs::exists(bin_file_path))
        return nullptr;

    return new BinFileWorker(txt_file_path, bin_file_path);
}

BinFileWorker *BinFileWorker::CreateInstance(const fs::path &txt_file_path) {
    if (!fs::exists(txt_file_path))
        return nullptr;

    fs::path bin_file = txt_file_path.parent_path() /
(txt_file_path.stem().string() + ".bin");
    std::ofstream file(bin_file);

    if (!file.is_open())
        return nullptr;
    file.close();

    return new BinFileWorker(txt_file_path, bin_file);
}

BinFileWorker::BinFileWorker(const fs::path &txt_file_path, const fs::path
&bin_file_path) {
    this->txt_file_path = txt_file_path;
    this->bin_file_path = bin_file_path;

    TranslateToBin();

    tree = new BinarySearchTree();
    InitTreeByBin();
}

BinFileWorker::~BinFileWorker() {
    delete tree;
}

void BinFileWorker::TranslateToBin() {
    std::fstream txt_file(txt_file_path, std::ios::in);
    std::fstream bin_file(bin_file_path, std::ios::out | std::ios::binary);

    if (!txt_file.is_open() || !bin_file.is_open())
        throw std::runtime_error("TranslateToBin-> Can't open file");

    std::string s;
    ViolationRecord violation{};

    while (std::getline(txt_file, s)) {
        violation.SetFieldsByStr(s);
```

```
        if (!violation.Validate())
            throw std::runtime_error("TranslateToBin-> Invalid record: " +
s);

        bin_file.write((char*)&violation, sizeof(ViolationRecord));
    }

    txt_file.close();
    bin_file.close();
}

void BinFileWorker::TranslateToTxt() {
    std::fstream txt_file(txt_file_path, std::ios::out);
    std::fstream bin_file(bin_file_path, std::ios::in | std::ios::binary);

    if (!txt_file.is_open() || !bin_file.is_open())
        throw std::runtime_error("TranslateToTxt-> Can't open file");

    ViolationRecord violation{};

    while (bin_file.read((char*)&violation, sizeof(ViolationRecord))) {
        txt_file << violation.ToString() << std::endl;
    }

    bin_file.close();
    txt_file.close();
}

void BinFileWorker::InitTreeByBin() {
    if (!tree)
        throw std::runtime_error("InitTreeByBin-> Tree is not initialized");

    std::fstream bin_file(bin_file_path, std::ios::in | std::ios::binary);
    if (!bin_file.is_open())
        throw std::runtime_error("InitTreeByBin-> Can't open file");

    ViolationRecord violation{};
    size_t ind = 0;

    while (bin_file.read((char*)&violation, sizeof(ViolationRecord))) {
        tree->Insert(violation.carNumber, ind++);
    }

    bin_file.close();
}

void BinFileWorker::PrintTree() {
    tree->PrintTree();
}

bool BinFileWorker::GetRecordByInd(size_t index, ViolationRecord &record) {
    std::fstream bin_file(bin_file_path, std::ios::binary | std::ios::in |
std::ios::out | std::ios::ate);

    if (!bin_file.is_open()) {
        return false;
    }

    size_t file_size = fs::file_size(bin_file_path);
    bin_file.seekg(0, std::ios::beg);
    int violations_count = file_size / sizeof(ViolationRecord);
```

```
    if (index > violations_count - 1) {
        return false;
    }

    std::streampos record_pos = index * sizeof(ViolationRecord);
    bin_file.seekg(record_pos, std::ios::beg);

    bin_file.read((char*)&record, sizeof(ViolationRecord));
    return true;
}

bool BinFileWorker::FindRecord(const std::string &car_number, ViolationRecord
&record) {
    if (!tree)
        throw std::runtime_error("FindRecord-> Tree is not initialized");

    size_t index;
    if (!tree->Find(car_number, index))
        return false;

    return GetRecordByInd(index, record);
}

void BinFileWorker::RemoveInTree(const std::string &car_number) const {
    if (!tree)
        throw std::runtime_error("RemoveInTree-> Tree is not initialized");

    tree->Remove(car_number);
}

int BinFileWorker::FindInTree(const std::string &car_number) const {
    if (!tree)
        throw std::runtime_error("FindInTree-> Tree is not initialized");

    size_t index;
    if (tree->Find(car_number, index))
        return index;

    return -1;
}

void BinFileWorker::AddToTree(std::string key, size_t ind) {
    if (!tree)
        throw std::runtime_error("AddToTree-> Tree is not initialized");

    tree->Insert(key, ind);

}
```

2.3.4 Основная программа

```
#include <iostream>
#include "BinFileWorker.h"


int main() {
    BinFileWorker *worker = BinFileWorker::CreateInstance("example.txt");
    int mode;
```

```
    std::cout << "1. Insert to tree\n"
                 "2. Find In Tree\n" <<
                 "3. Remove from tree\n" <<
                 "4. Print tree\n" <<
                 "5. Exit\n";
    std::cin >> mode;

    size_t ind;
    std::string key;
    while (mode != 5) {
        switch (mode) {
            case 1:
                std::cout << "Enter key and ind: ";
                std::cin >> key >> ind;
                worker->AddToTree(key, ind);
                break;
            case 2:
                std::cout << "Enter key: ";
                std::cin >> key;
                ViolationRecord r;
                if (worker->FindRecord(key, r))
                    std::cout << "Found: " << r.ToString() << std::endl;
                else
                    std::cout << "Not found" << std::endl;
                break;
            case 3:
                std::cout << "Enter key: ";
                std::cin >> key;
                worker->RemoveInTree(key);
            case 4:
                worker->PrintTree();
            default:
                std::cout << "Invalid mode" << std::endl;
                break;
        }

        std::cout << "1. Insert to tree\n"
                     "2. FindInTree\n" <<
                     "3. Remove from tree\n" <<
                     "4. Print tree\n" <<
                     "5. Exit\n";
        std::cin >> mode;
    }
}
```

2.4 Тестирование

На рисунке 1 показана структура дерева после инициализации данными, которые описаны вышек

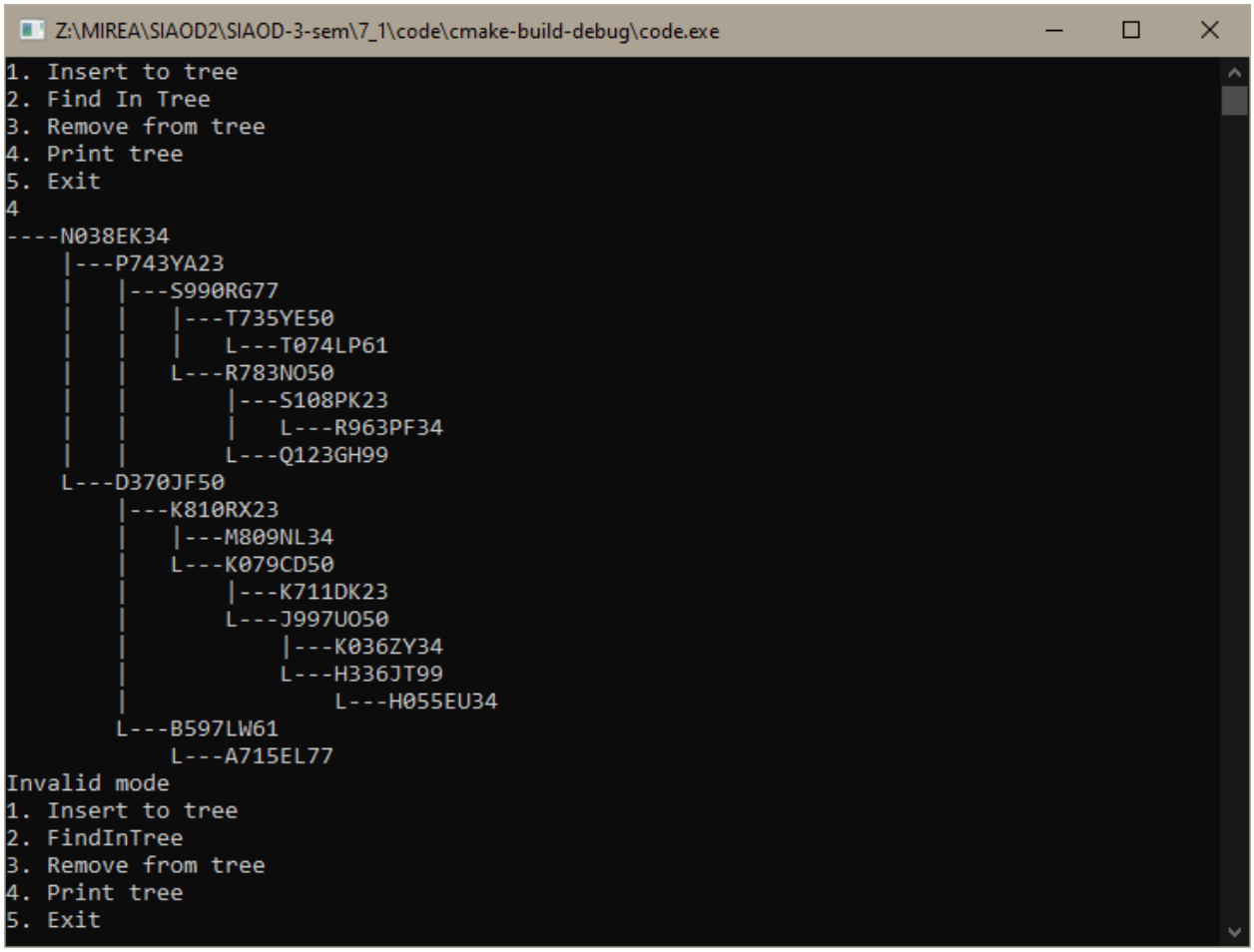

Рисунок 1 – Тестирование приложения №1

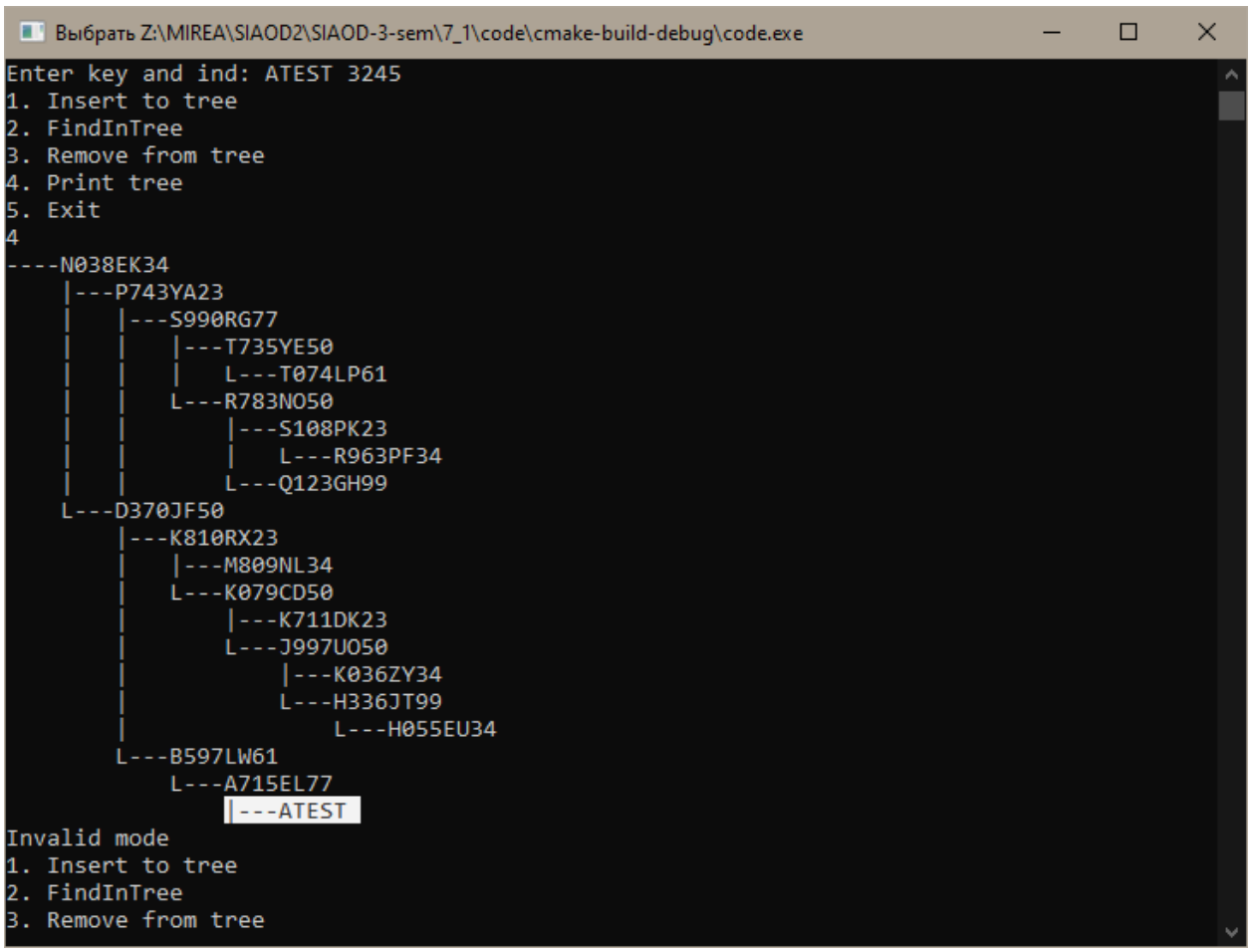

Рисунок 2 – Тестирование вставки нового узла

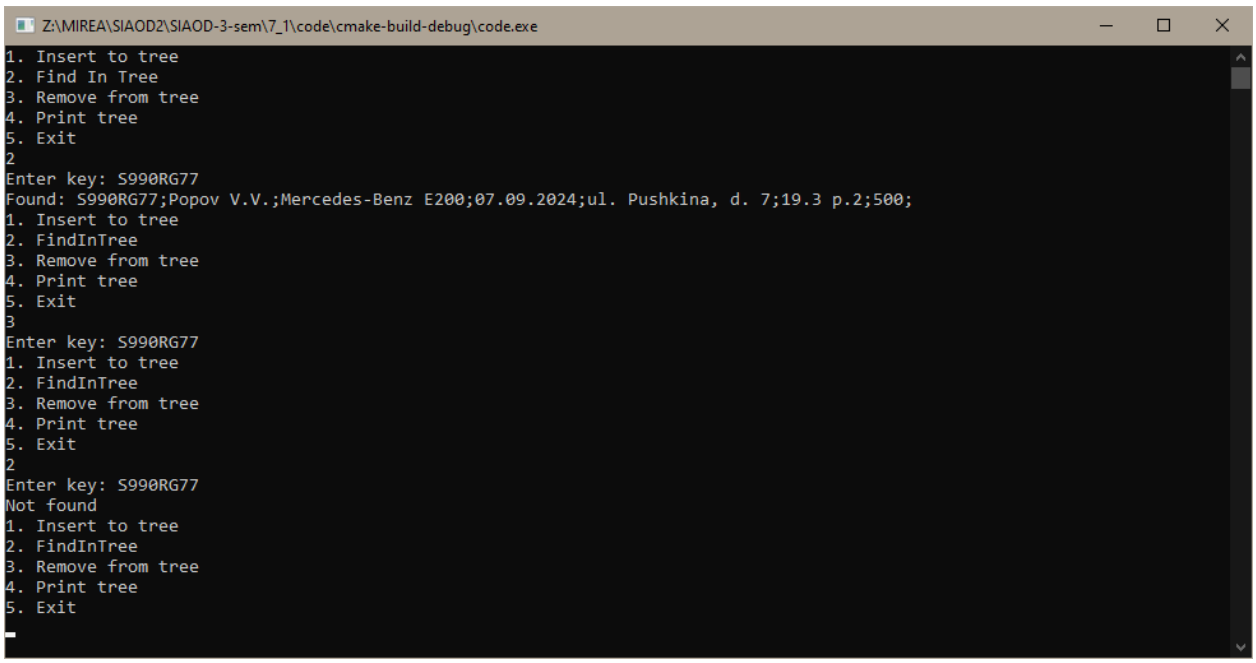

Рисунок 3 – Удаление и поиск элемента

3 ЗАДАНИЕ №2

3.1 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.

Таблица 2 – Индивидуальный вариант №28

Сбалансированное дерево поиска (СДП)	Структура элемента множества (ключ – подчеркнутое поле) остальные поля представляют данные элемента
Косое	Учет нарушений ПДД. Структура записи о нарушении ПДД: номер автомобиля, фамилия и инициалы владельца, модель, дата нарушения, место нарушения (текстом), статья (КоАП), наказание (сумма штрафа).

3.2 Ход решения

Для решения поставленной задачи воспользуемся классом BinFileWorker, который был немного изменен для непосредственной работы с

косым деревом, и структурой ViolationRecord, описанной выше. Кроме того, создадим класс для управления сбалансированным деревом поиска (Splay tree). Он будет содержать следующие методы:

1. Insert – метод добавления нового ключа
2. Find – метод поиска узла по ключу
3. Remove – метод удаления узла

   Осуществляется с помощью метода поиска и Splay, который поднимает узел в корень дерева, откуда его легко удалить, по принципу, описанному выше.
4. PrintTree – метод вывода дерева в консоль
5. Splay – основной метод данного дерева, который совершает пово
6. RightRotate – Правый поворот
7. LeftRotate – Левый поворот

Для подсчета количества поворотов будет использоваться публичное поле rotations.

Некоторые изменения в BinFileWorker:

1. Изменен вид дерева для работы с файлом (на SplayTree), следовательно, незначительно изменены существующие методы для работы с этим деревом, и добавлены новые методы:
   a. GetRotations – для получения количеств поворотов после инициализации дерева для задания;
   b. RemoveFromTree – удаляет узел из дерева по ключу
2.

3.3 Программная реализация

3.3.1 SplayTree

SplayTree.h:

```
#ifndef CODE_SPLAYTREE_H
#define CODE_SPLAYTREE_H

#include "Node.h"
```

```
class SplayTree {
private:
    std::shared_ptr<Node> p_root = nullptr;
    std::shared_ptr<Node> RightRotate(std::shared_ptr<Node> node);
    std::shared_ptr<Node> LeftRotate(std::shared_ptr<Node> node);
    void Splay(std::shared_ptr<Node> node);
    std::shared_ptr<Node> FindMin(std::shared_ptr<Node> node);
    void PrintHelper(std::shared_ptr<Node> node, const std::string& prefix,
bool isLeft, bool isRoot = false);

public:
    int rotations = 0; // for testing

    void Insert(const std::string &key, size_t ind);
    bool Find(const std::string &key, size_t& found_ind);
    void Remove(const std::string &key);
    void PrintTree();
};


#endif //CODE_SPLAYTREE_H
```

SplayTree.cpp:

```
#include "SplayTree.h"
#include <iostream>

void SplayTree::Splay(std::shared_ptr<Node> node) {
    while (node->p_parent) {
        if (!node->p_parent->p_parent) {
            if (node->IsLeft())
                RightRotate(node->p_parent);
            else
                LeftRotate(node->p_parent);
        } else {
            bool isLeftChild = node->IsLeft();
            bool parentIsLeftChild = node->p_parent->IsLeft();

            if (isLeftChild == parentIsLeftChild) {
                // Zig-Zig
                if (isLeftChild) {
                    RightRotate(node->p_parent->p_parent);
                    RightRotate(node->p_parent);
                } else {
                    LeftRotate(node->p_parent->p_parent);
                    LeftRotate(node->p_parent);
                }
            } else {
                // Zig-Zag
                if (isLeftChild) {
                    RightRotate(node->p_parent);
                    LeftRotate(node->p_parent);
                } else {
                    LeftRotate(node->p_parent);
                    RightRotate(node->p_parent);
                }
            }
        }
    }
}
```

```
std::shared_ptr<Node> SplayTree::RightRotate(std::shared_ptr<Node> node) {
    rotations++;

    auto new_root = node->p_left;
    if (!new_root) return node;

    node->p_left = new_root->p_right;
    if (new_root->p_right) {
        new_root->p_right->p_parent = node;
    }

    new_root->p_parent = node->p_parent;
    if (!node->p_parent) {
        p_root = new_root;
    } else if (node->IsLeft()) {
        node->p_parent->p_left = new_root;
    } else {
        node->p_parent->p_right = new_root;
    }

    new_root->p_right = node;
    node->p_parent = new_root;

    return new_root;
}

std::shared_ptr<Node> SplayTree::LeftRotate(std::shared_ptr<Node> node) {
    rotations++;

    auto new_root = node->p_right;
    if (!new_root) return node;

    node->p_right = new_root->p_left;
    if (new_root->p_left) {
        new_root->p_left->p_parent = node;
    }

    new_root->p_parent = node->p_parent;
    if (!node->p_parent) {
        p_root = new_root;
    } else if (node->IsLeft()) {
        node->p_parent->p_left = new_root;
    } else {
        node->p_parent->p_right = new_root;
    }

    new_root->p_left = node;
    node->p_parent = new_root;

    return new_root;
}

std::shared_ptr<Node> SplayTree::FindMin(std::shared_ptr<Node> node) {
    while (node && node->p_left)
        node = node->p_left;

    return node;
}

void SplayTree::Insert(const std::string &key, size_t ind) {
```

```
    if (!p_root) {
        p_root = std::make_shared<Node>(key, ind, nullptr);
        return;
    }

    std::shared_ptr<Node> p_cur_node = p_root;
    std::shared_ptr<Node> p_parent = nullptr;

    while (p_cur_node) {
        p_parent = p_cur_node;
        if (key < p_cur_node->key)
            p_cur_node = p_cur_node->p_left;
        else
            p_cur_node = p_cur_node->p_right;
    }

    if (key < p_parent->key) {
        p_parent->p_left = std::make_shared<Node>(key, ind, p_parent);
        Splay(p_parent->p_left);
    } else {
        p_parent->p_right = std::make_shared<Node>(key, ind, p_parent);
        Splay(p_parent->p_right);
    }
}

bool SplayTree::Find(const std::string &key, size_t &found_ind) {
    std::shared_ptr<Node> p_cur_node = p_root;

    while (p_cur_node && p_cur_node->key != key) {
        if (key < p_cur_node->key)
            p_cur_node = p_cur_node->p_left;
        else
            p_cur_node = p_cur_node->p_right;
    }

    if (p_cur_node) {
        found_ind = p_cur_node->link_to_record;
        Splay(p_cur_node);
    }

    return (bool)p_cur_node;
}

void SplayTree::Remove(const std::string &key) {
    size_t ind;
    if (!Find(key, ind))
        return;

    std::shared_ptr<Node> left_subtree = p_root->p_left;
    std::shared_ptr<Node> right_subtree = p_root->p_right;

    if (!right_subtree) {
        p_root = left_subtree;
        if (p_root) p_root->p_parent = nullptr;
    } else {
        std::shared_ptr<Node> min_node = FindMin(right_subtree);
        Splay(min_node);
        min_node->p_left = left_subtree;
        if (left_subtree) left_subtree->p_parent = min_node;
        p_root = min_node;
    }
```

```
}

void SplayTree::PrintTree() {
    PrintHelper(p_root, "", false, true);
}

void SplayTree::PrintHelper(std::shared_ptr<Node> node, const std::string
&prefix, bool isLeft, bool isRoot) {
    if (node != nullptr) {

        std::cout << prefix;
        if (isRoot)
            std::cout <<  "----";
        else
            std::cout << (isLeft ? "L---" : "|---");

        std::cout << node->key << std::endl;

        PrintHelper(node->p_right, prefix + (isLeft ? "    " : "|   "),
false);
        PrintHelper(node->p_left, prefix + (isLeft ? "    " : "|   "), true);
    }
}
```

3.3.2 BinFileWorker

BinFileWorker.h:

```
#ifndef CODE_BINFILEWORKER_H
#define CODE_BINFILEWORKER_H

#include <fstream>
#include <filesystem>
#include "ViolationRecord.h"
#include "SplayTree.h"

namespace fs = std::filesystem;

class BinFileWorker {
private:
    fs::path bin_file_path;
    fs::path txt_file_path;

    SplayTree* tree = nullptr;

    BinFileWorker(const fs::path& txt_file_path, const fs::path&
bin_file_path);
public:
    static BinFileWorker* CreateInstance(const fs::path& txt_file_path, const
fs::path& bin_file_path);
    static BinFileWorker* CreateInstance(const fs::path& txt_file_path);
    ~BinFileWorker();

    void RemoveFromTree(const std::string& key);
    void TranslateToBin();
    void TranslateToTxt();
    void InitTreeByBin();
    void PrintTree();
```

```
    bool GetRecordByInd(size_t index, ViolationRecord& record);
    bool FindRecord(const std::string& car_number, ViolationRecord& record);
    int GetRotations();
    void PrintBin();
};


#endif //CODE_BINFILEWORKER_H
```

BinFileWorker.cpp

```
#include "BinFileWorker.h"


BinFileWorker *BinFileWorker::CreateInstance(const fs::path& txt_file_path,
const fs::path& bin_file_path) {
    if (!fs::exists(txt_file_path))
        return nullptr;

    if (!fs::exists(bin_file_path))
        return nullptr;

    return new BinFileWorker(txt_file_path, bin_file_path);
}

BinFileWorker *BinFileWorker::CreateInstance(const fs::path &txt_file_path) {
    if (!fs::exists(txt_file_path))
        return nullptr;

    fs::path bin_file = txt_file_path.parent_path() /
(txt_file_path.stem().string() + ".bin");
    std::ofstream file(bin_file);

    if (!file.is_open())
        return nullptr;
    file.close();

    return new BinFileWorker(txt_file_path, bin_file);
}

BinFileWorker::BinFileWorker(const fs::path &txt_file_path, const fs::path
&bin_file_path) {
    this->txt_file_path = txt_file_path;
    this->bin_file_path = bin_file_path;

    TranslateToBin();

    tree = new SplayTree();
    InitTreeByBin();
}

BinFileWorker::~BinFileWorker() {
    delete tree;
}

void BinFileWorker::TranslateToBin() {
    std::fstream txt_file(txt_file_path, std::ios::in);
    std::fstream bin_file(bin_file_path, std::ios::out | std::ios::binary);

    if (!txt_file.is_open() || !bin_file.is_open())
```

```
        throw std::runtime_error("TranslateToBin-> Can't open file");

    std::string s;
    ViolationRecord violation{};

    while (std::getline(txt_file, s)) {
        violation.SetFieldsByStr(s);

        if (!violation.Validate())
            throw std::runtime_error("TranslateToBin-> Invalid record: " +
s);

        bin_file.write((char*)&violation, sizeof(ViolationRecord));
    }

    txt_file.close();
    bin_file.close();
}

void BinFileWorker::TranslateToTxt() {
    std::fstream txt_file(txt_file_path, std::ios::out);
    std::fstream bin_file(bin_file_path, std::ios::in | std::ios::binary);

    if (!txt_file.is_open() || !bin_file.is_open())
        throw std::runtime_error("TranslateToTxt-> Can't open file");

    ViolationRecord violation{};

    while (bin_file.read((char*)&violation, sizeof(ViolationRecord))) {
        txt_file << violation.ToString() << std::endl;
    }

    bin_file.close();
    txt_file.close();
}

void BinFileWorker::InitTreeByBin() {
    if (!tree)
        throw std::runtime_error("InitTreeByBin-> Tree is not initialized");

    std::fstream bin_file(bin_file_path, std::ios::in | std::ios::binary);
    if (!bin_file.is_open())
        throw std::runtime_error("InitTreeByBin-> Can't open file");

    ViolationRecord violation{};
    size_t ind = 0;

    while (bin_file.read((char*)&violation, sizeof(ViolationRecord))) {
        tree->Insert(violation.carNumber, ind++);
    }

    bin_file.close();
}

void BinFileWorker::PrintTree() {
    tree->PrintTree();
}

bool BinFileWorker::GetRecordByInd(size_t index, ViolationRecord &record) {
    std::fstream bin_file(bin_file_path, std::ios::binary | std::ios::in |
std::ios::out | std::ios::ate);
```

```
    if (!bin_file.is_open()) {
        return false;
    }

    size_t file_size = fs::file_size(bin_file_path);
    bin_file.seekg(0, std::ios::beg);
    int violations_count = file_size / sizeof(ViolationRecord);

    if (index > violations_count - 1) {
        return false;
    }

    std::streampos record_pos = index * sizeof(ViolationRecord);
    bin_file.seekg(record_pos, std::ios::beg);

    bin_file.read((char*)&record, sizeof(ViolationRecord));
    return true;
}

bool BinFileWorker::FindRecord(const std::string &car_number, ViolationRecord
&record) {
    if (!tree)
        throw std::runtime_error("FindRecord-> Tree is not initialized");

    size_t index;
    if (!tree->Find(car_number, index))
        return false;

    return GetRecordByInd(index, record);
}

void BinFileWorker::RemoveFromTree(const std::string &key) {
    if (!tree)
        throw std::runtime_error("RemoveFromTree-> Tree is not initialized");

    tree->Remove(key);
}

int BinFileWorker::GetRotations() {
    return tree->rotations;
}
```

3.3.3 Основная программа

```
#include "BinFileWorker.h"
#include <iostream>

int main() {
    BinFileWorker *worker = BinFileWorker::CreateInstance("example.txt");
    int mode;

    std::cout << "1. Find\n" <<
                 "2. Remove\n" <<
                 "3. Print\n" <<
                 "4. Get rotations\n" <<
                 "5. Exit\n";
    std::cin >> mode;

    size_t ind;
```

```
    std::string key;
    while (mode != 5) {
        switch (mode) {
            case 1:
                std::cout << "Enter key: ";
                std::cin >> key;
                ViolationRecord r;
                if (worker->FindRecord(key, r))
                    std::cout << "Found: " << r.ToString() << std::endl;
                else
                    std::cout << "Not found" << std::endl;
                break;
            case 2:
                std::cout << "Enter key: ";
                std::cin >> key;
                worker->RemoveFromTree(key);
                break;
            case 3:
                worker->PrintTree();
                break;
            case 4:
                std::cout << "Rotations: " << worker->GetRotations() <<
std::endl;
                break;
            default:
                std::cout << "Invalid mode" << std::endl;
                break;
        }

        std::cout << "1. Find\n" <<
                  "2. Remove\n" <<
                  "3. Print\n" <<
                  "4. Get rotations\n" <<
                  "5. Exit\n";
        std::cin >> mode;


    }
}
```

3.4 Тестирование

3.4.1 Инициализация и вывод дерева

Как и в прошлом задании файл содержит 20 записей:

```
N038EK34;Kuznetsov P.P.;Nissan Teana;03.09.2024;ul. Lenina, d. 12;20.1
p.1;1200;
P743YA23;Kuznetsov K.K.;Honda Accord;28.09.2024;ul. Lenina, d. 12;12.5
p.1;1200;
S990RG77;Popov V.V.;Mercedes-Benz E200;07.09.2024;ul. Pushkina, d. 7;19.3
p.2;500;
D370JF50;Popov I.I.;BMW X5;28.09.2024;ul. Sovetskaya, d. 15;19.3 p.2;700;
K810RX23;Sidorov K.K.;Nissan Teana;13.09.2024;ul. Sovetskaya, d. 15;20.1
p.1;300;
K079CD50;Petrov V.V.;Honda Accord;06.09.2024;pr. Mira, d. 5;12.5 p.1;1200;
J997UO50;Ivanov K.K.;Nissan Teana;10.09.2024;pr. Mira, d. 5;19.3 p.2;1000;
K711DK23;Mikhailov A.A.;Nissan Teana;11.09.2024;ul. Pushkina, d. 7;19.3
p.2;300;
T735YE50;Fedorov S.S.;Honda Accord;22.09.2024;ul. Lenina, d. 12;12.9 p.1;500;
```

```
B597LW61;Popov V.V.;Nissan Teana;15.09.2024;ul. Sovetskaya, d. 15;12.9
p.1;1200;
R783NO50;Fedorov M.M.;Mercedes-Benz E200;17.09.2024;ul. Lenina, d. 12;12.5
p.1;300;
S108PK23;Smirnov S.S.;Lada Granta;19.09.2024;ul. Sovetskaya, d. 15;12.9
p.1;1200;
R963PF34;Mikhailov V.V.;Toyota Camry;20.09.2024;pr. Mira, d. 5;12.5 p.1;500;
T074LP61;Smirnov S.S.;BMW X5;06.09.2024;ul. Lenina, d. 12;12.8 p.2;300;
Q123GH99;Fedorov V.V.;Nissan Teana;19.09.2024;ul. Lenina, d. 12;12.5 p.1;300;
M809NL34;Sokolov A.A.;Mercedes-Benz E200;17.09.2024;ul. Pushkina, d. 7;20.1
p.1;500;
H336JT99;Sokolov V.V.;Honda Accord;04.09.2024;ul. Gagarina, d. 22;12.9
p.1;700;
K036ZY34;Mikhailov V.V.;Honda Accord;13.09.2024;pr. Mira, d. 5;12.9 p.1;700;
H055EU34;Petrov A.A.;Nissan Teana;12.09.2024;ul. Sovetskaya, d. 15;20.1
p.1;1200;
A715EL77;Kuznetsov I.I.;Nissan Teana;12.09.2024;ul. Lenina, d. 12;12.9
p.1;500;
```

Результат инициализации дерева представлен на рис. 1.

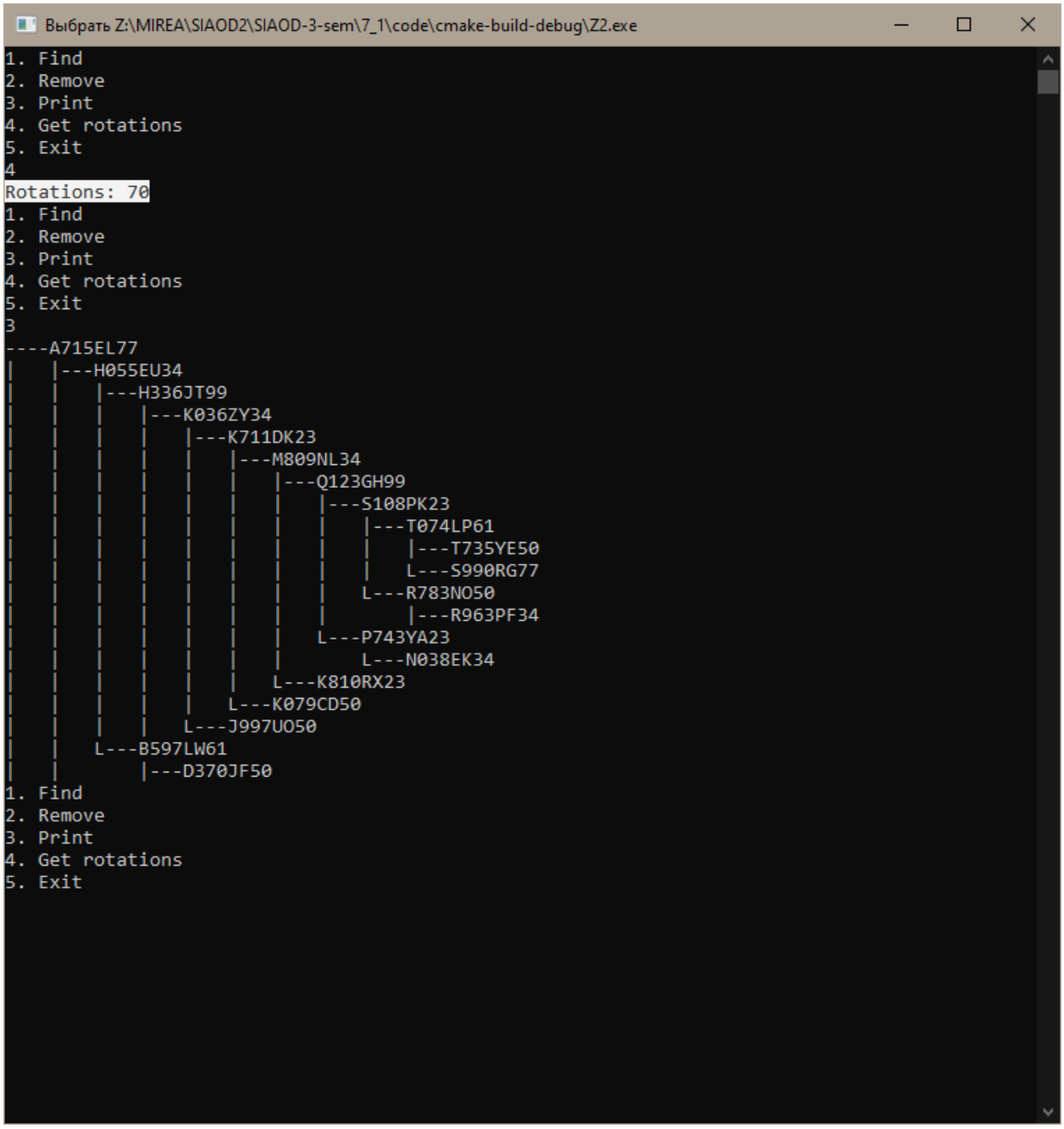

Рисунок 1 – Тестирование приложения №2

Как видно из рис. 1, количество поворотов при вставке 20 ключа совпадает с числом 70, откуда **среднее число поворотов** для одного ключа – 3,5.

3.4.2 Поиск по ключу

Для демонстрации работы попробуем отыскать ключ M809NL34, после чего он должен подняться в корень дерево, в соответствии с идеей косого дерева (рис. 2).

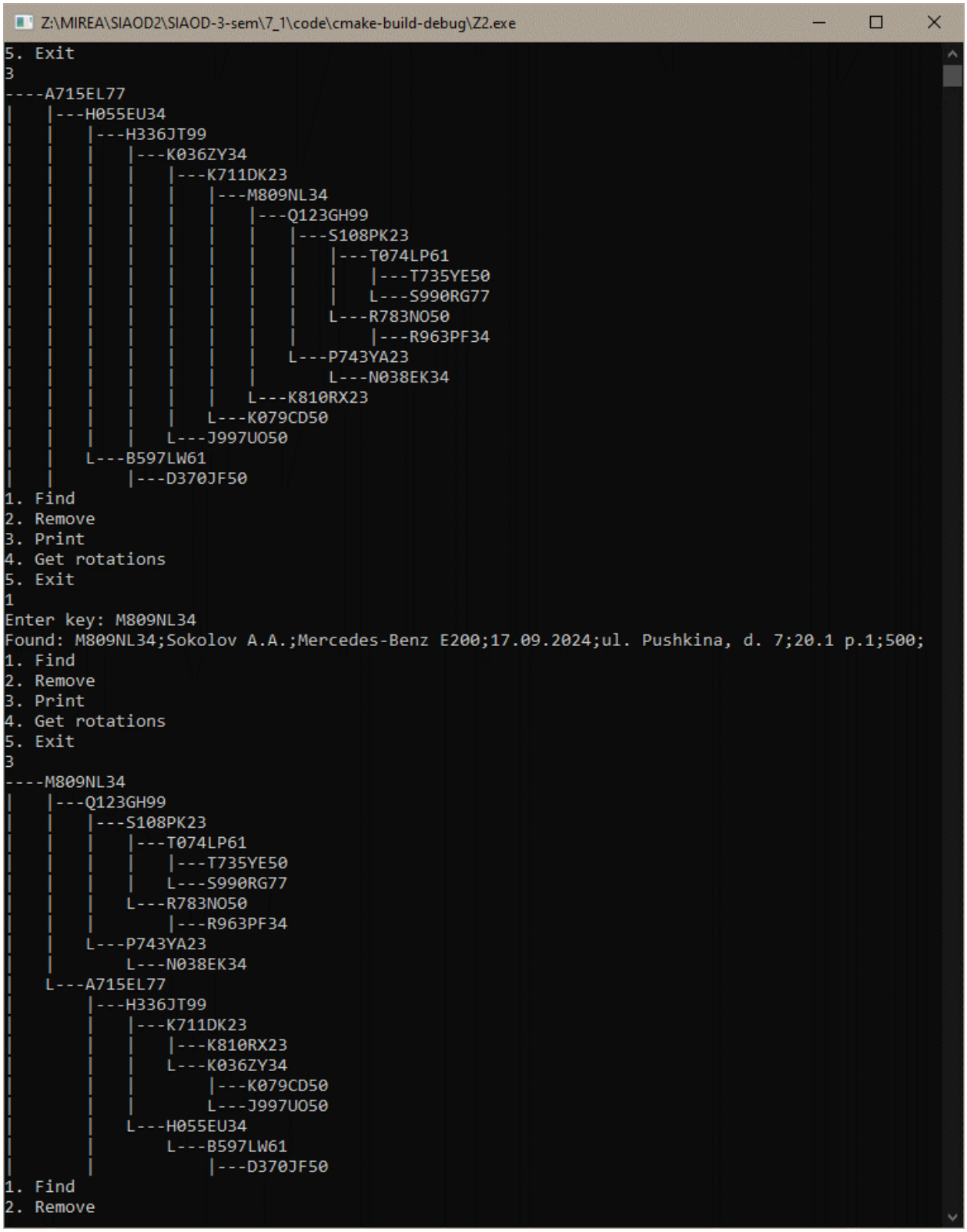

Рисунок 2 – Тестирование приложения №2

3.4.3 Удаление элемента

Для примера удалим элемент с ключом zks (рис. 3).

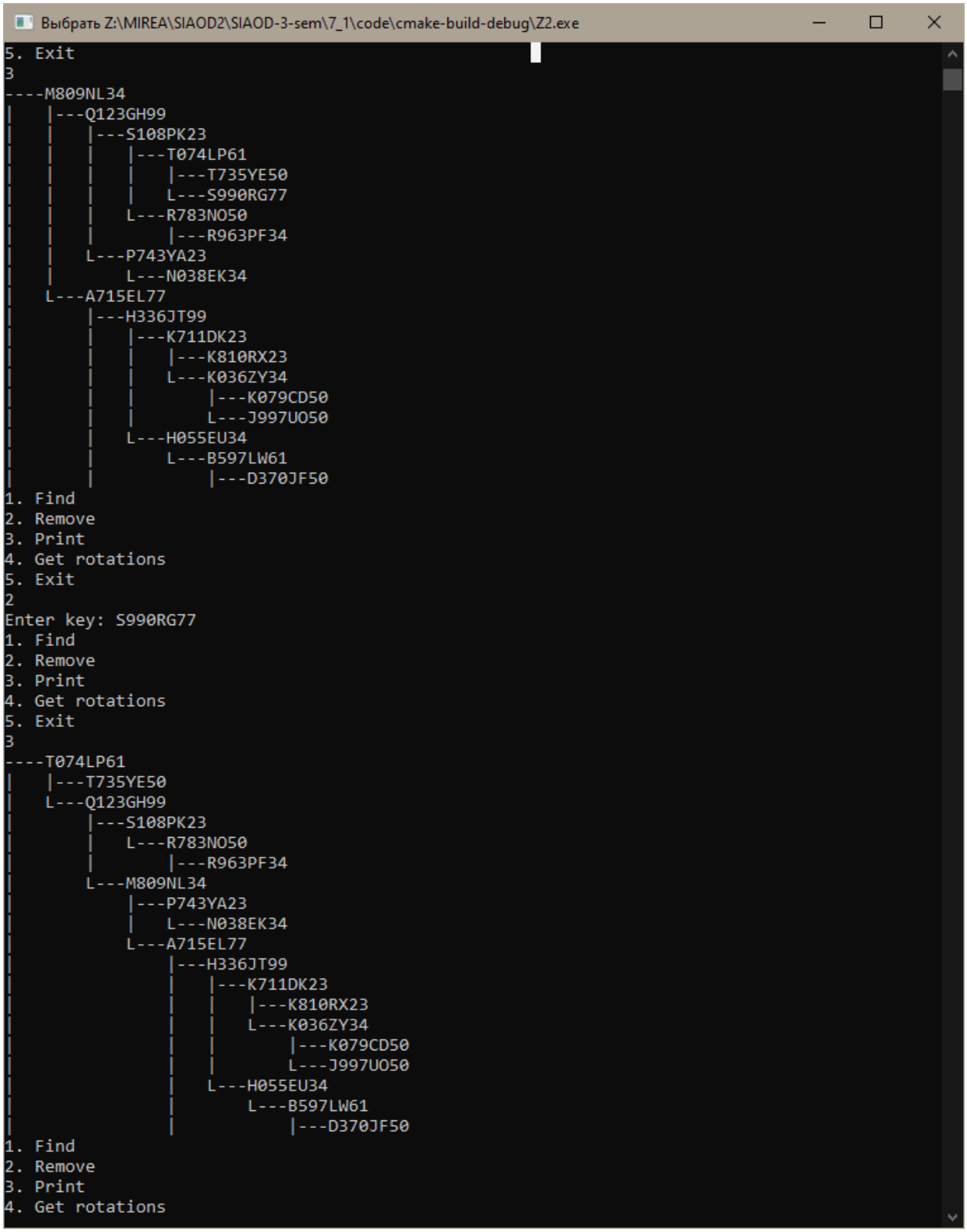

Рисунок 3 – Тестирование приложения №2

4 ЗАДАНИЕ 3

4.1 Постановка задачи

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

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

Требования по выполнению задания

1. Протестировать на числовой последовательности: а) небольшого объема; 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 по определенной форме

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

4.2 Ход решения

Для решения поставленной задачи понадобятся структуры, реализованные раннее, а именно HashTable, BinarySearchTree, SplayTree. Отличие состоит в том, что ключи в этот раз представляют из себя целые числа, а не строки, и реализованные вспомогательные методы для получения некоторых параметров структур (занимаемое место и кол-во сравнений при поиске).

Входные данные – целые числа, далее приведет пример этих чисел:

```
7147054
1352289
2753885
471983
1406567
7845183
5508115
5565329
53228
996482
```

```
6720130
7539022
2534267
9526178
6823508
3114186
5655118
5168373
9926594
7382459
7693748
8264771
67711
2716271
4855173
```

В теле основной программы будет происходить инициализация всех этих структур данными из файла nums (числами в определенном формате, продемонстрированном выше). Кроме того, будет осуществляться замер времени поиска случайного числа в каждой из структур.

4.3 Программная реализация

main.cpp:

```cpp
#include <fstream>
#include "HashTable.h"
#include "BinarySearchTree.h"
#include "SplayTree.h"
#include <chrono>
#include <cassert>

std::string path = "../Z3/nums";

void InitHashTable(HashTable& hashTable, int count = 0) {
    std::ifstream file(path);
    int num;

    if (count > 0) {
        for (int i = 0; i < count; i++) {
            file >> num;
            InsertElem(hashTable, num);
        }
    } else {
        while (file >> num) {
            InsertElem(hashTable, num);
        }
    }

    file.close();
}

void InitBinarySearchTree(BinarySearchTree& tree, int count = 0) {
    std::ifstream file(path);
    int num;

    if (count > 0) {
```

```
        for (int i = 0; i < count; i++) {
            file >> num;
            tree.Insert(num);
        }
    } else {
        while (file >> num) {
            tree.Insert(num);
        }
    }

    file.close();
}

void InitSplayTree(SplayTree& tree, int count = 0) {
    std::ifstream file(path);
    int num;

    if (count > 0) {
        for (int i = 0; i < count; i++) {
            file >> num;
            tree.Insert(num);
        }
    } else {
        while (file >> num) {
            tree.Insert(num);
        }
    }

    file.close();

}

int main () {
    HashTable hashTable;
    BinarySearchTree tree;
    SplayTree splayTree;

    int count = 0;
    int toFind = 9735410;

    InitHashTable(hashTable, count);
    InitBinarySearchTree(tree, count);
    InitSplayTree(splayTree, count);

    int k1;
    int k2;
    int k3;
    int k4;

    auto start = std::chrono::high_resolution_clock::now();
    k1 = FindElem(hashTable, toFind);
    auto end = std::chrono::high_resolution_clock::now();

    auto hashTableTime = std::chrono::duration<double>(end - start).count();;

    start = std::chrono::high_resolution_clock::now();
    k2 = tree.Find(toFind);
    end = std::chrono::high_resolution_clock::now();

    auto binaryTreeTime = std::chrono::duration<double>(end - start).count();
```

```
    start = std::chrono::high_resolution_clock::now();
    k3 = splayTree.Find(toFind);
    end = std::chrono::high_resolution_clock::now();

    auto splayTreeTime = std::chrono::duration<double>(end - start).count();


    start = std::chrono::high_resolution_clock::now();
    k4 = splayTree.Find(toFind);
    end = std::chrono::high_resolution_clock::now();

    auto repeatSplayTreeTime = std::chrono::duration<double>(end -
start).count();

    std::cout << std::fixed << std::setprecision(9) << "HashTable time:
" << hashTableTime << " s | " << MemoryUsage(hashTable) << " byte | " << k1
<< std::endl;
    std::cout << std::fixed << std::setprecision(9) << "BinarySearchTree
time: " << binaryTreeTime << " s | " << tree.MemoryUsage() << " byte | " <<
k2 << std::endl;
    std::cout << std::fixed << std::setprecision(9) << "SplayTree time:
" << splayTreeTime << " s | " << splayTree.MemoryUsage() << " byte | " << k3
<< std::endl;
    std::cout << std::fixed << std::setprecision(9) << "Repeat SplayTree
time: " << repeatSplayTreeTime << " s | " << splayTree.MemoryUsage() << "
byte | " << k4  << std::endl;
}
```

4.4 Тестирование

Для 100 чисел (рис. 4):

```
Z:\MIREA\SIAOD2\SIAOD-3-sem\7_1\code\cmake-build-debug\Z3.exe
HashTable time:         0.000000500 s | 1600 byte | 2
BinarySearchTree time: 0.000000200 s | 4008 byte | 12
SplayTree time:         0.000002500 s | 6416 byte | 21
Repeat SplayTree time: 0.000000100 s | 6416 byte | 1

Process finished with exit code 0
```

Рисунок 5 – Тестирование для 100 чисел

Для 10000 чисел (рис. 5):

```
Z:\MIREA\SIAOD2\SIAOD-3-sem\7_1\code\cmake-build-debug\Z3.exe
HashTable time:         0.000000300 s | 205760 byte | 2
BinarySearchTree time: 0.000000800 s | 400008 byte | 12
SplayTree time:         0.000008600 s | 640016 byte | 87
Repeat SplayTree time: 0.000000200 s | 640016 byte | 1

Process finished with exit code 0
```

Рисунок 6 – Тестирование для 10000 чисел

4.5 Результаты измерений

Результаты измерений времени поиска элемента в разных структурах данных представлены в табл. 3.

Вид структуры	Кол-во элементов	Емкостная сложность, байт	Кол-во сравнений, Время поиска
Хеш-таблица	100	1600	2
Хеш-таблица	10000	205760	2
Бинарное дерево	100	4008	12
Бинарное дерево	10000	400008	12
Косое дерево	100	6416	21
Косое дерево	10000	640016	87

Анализируя результаты измерений, хорошо видны отличительные особенности каждой структуры данных. Вот некоторые из них:

- В бинарном и косом дереве объем занимаемой памяти растет пропорционально кол-ву элементов, а в хеш-таблице размер таблицы определяется простым числом, большим кол-ва элементов.
- В косом дереве повторный поиск одного и того же элемента происходит моментально, потому что он находится в корне дерева.
- Поиск в бинарном дереве несильно зависит от общего количества элементов.

Каждая структура имеет свои специфические особенности, и использование той или иной структуры предпочтительнее в разных обстоятельствах.

ВЫВОД

Получили навыки в разработки и реализации алгоритмов управления бинарным деревом поиска и сбалансированными бинарными деревьями поиска (АВЛ – деревьями);

Получить навыки в применении файловых потоков прямого доступа к данным файла;

Получить навыки в применении сбалансированного дерева поиска для прямого доступа к записям файла.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Рысин, М. Л. Введение в структуры и алгоритмы обработки данных : учебное пособие / М. Л. Рысин, М. В. Сартаков, М. Б. Туманова. — Москва : РТУ МИРЭА, 2022 — Часть 2 : Поиск в тексте. Нелинейные структуры данных. Кодирование информации. Алгоритмические стратегии — 2022. — 111 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/310826 (дата обращения: 10.09.2024).
2. ГОСТ 19.701-90. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения : межгосударственный стандарт : дата введения 1992-01- 01 / Федеральное агентство по техническому регулированию. – Изд. официальное. – Москва : Стандартинформ, 2010. – 23 с