

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 4](#_Toc184758891)

[2 ЗАДАНИЕ №1 5](#_Toc184758892)

[2.1 Постановка задачи 5](#_Toc184758893)

[2.2 Ход решения 5](#_Toc184758894)

[2.3 Программная реализация 7](#_Toc184758895)

[2.3.1 Структура ViolationRecord 7](#_Toc184758896)

[2.3.2 BinarySearchTree 8](#_Toc184758897)

[2.3.3 BinFileWorker 12](#_Toc184758898)

[2.3.4 Основная программа 15](#_Toc184758899)

[2.4 Тестирование 16](#_Toc184758900)

[3 ЗАДАНИЕ №2 19](#_Toc184758901)

[3.1 Постановка задачи 19](#_Toc184758902)

[3.2 Ход решения 19](#_Toc184758903)

[3.3 Программная реализация 20](#_Toc184758904)

[3.3.1 SplayTree 20](#_Toc184758905)

[3.3.2 BinFileWorker 24](#_Toc184758906)

[3.3.3 Основная программа 27](#_Toc184758907)

[3.4 Тестирование 28](#_Toc184758908)

[3.4.1 Инициализация и вывод дерева 28](#_Toc184758909)

[3.4.2 Поиск по ключу 30](#_Toc184758910)

[3.4.3 Удаление элемента 31](#_Toc184758911)

[4 ЗАДАНИЕ 3 33](#_Toc184758912)

[4.1 Постановка задачи 33](#_Toc184758913)

[4.2 Ход решения 33](#_Toc184758914)

[4.3 Программная реализация 34](#_Toc184758915)

[4.4 Тестирование 36](#_Toc184758916)

[4.5 Результаты измерений 37](#_Toc184758917)

[ВЫВОД 38](#_Toc184758918)

[СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ 39](#_Toc184758919)

# 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;
   1. 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

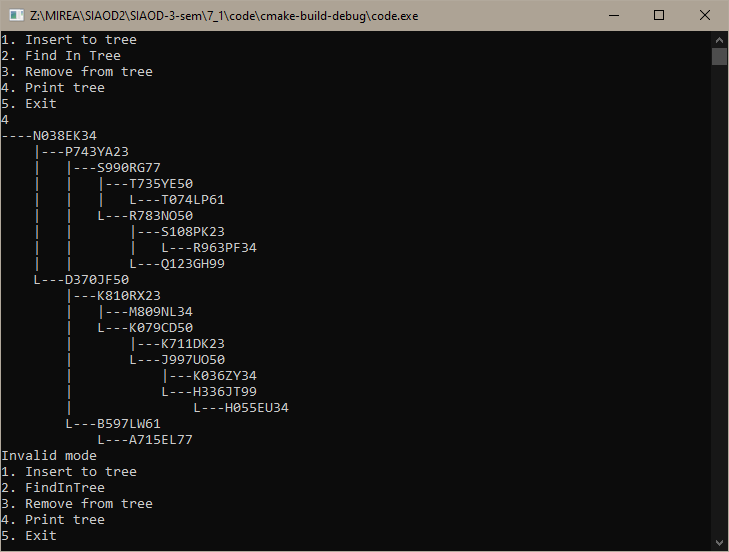


Рисунок 1 – Тестирование приложения №1

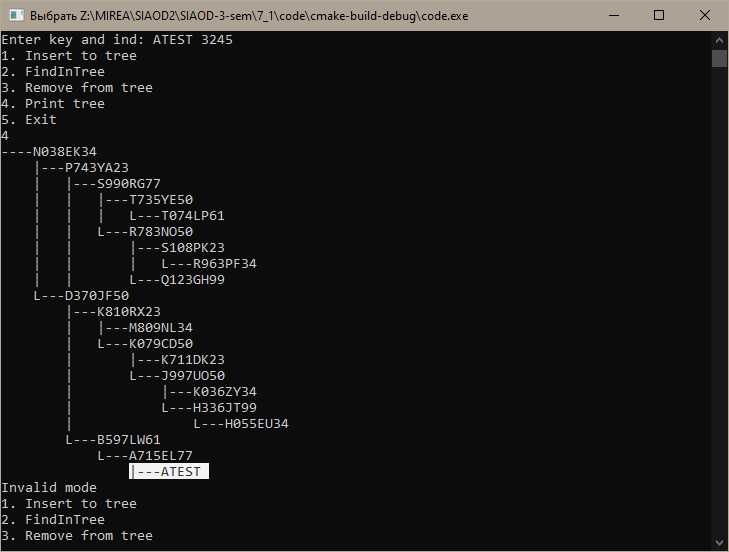


Рисунок 2 – Тестирование вставки нового узла

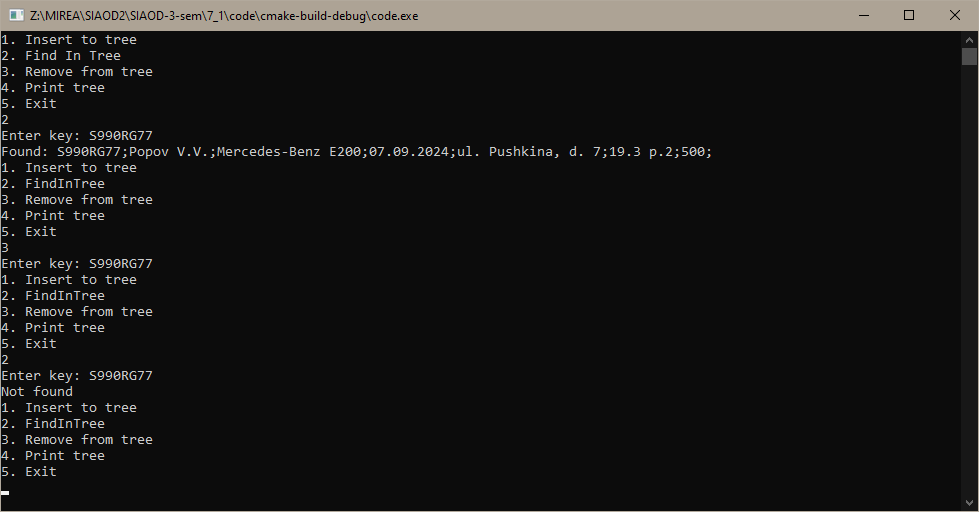


Рисунок 3 – Удаление и поиск элемента

# 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.

1. PrintTree – метод вывода дерева в консоль
2. Splay – основной метод данного дерева, который совершает пово
3. RightRotate – Правый поворот
4. 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:
   1. GetRotations – для получения количеств поворотов после инициализации дерева для задания;
   2. RemoveFromTree – удаляет узел из дерева по ключу

## 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.

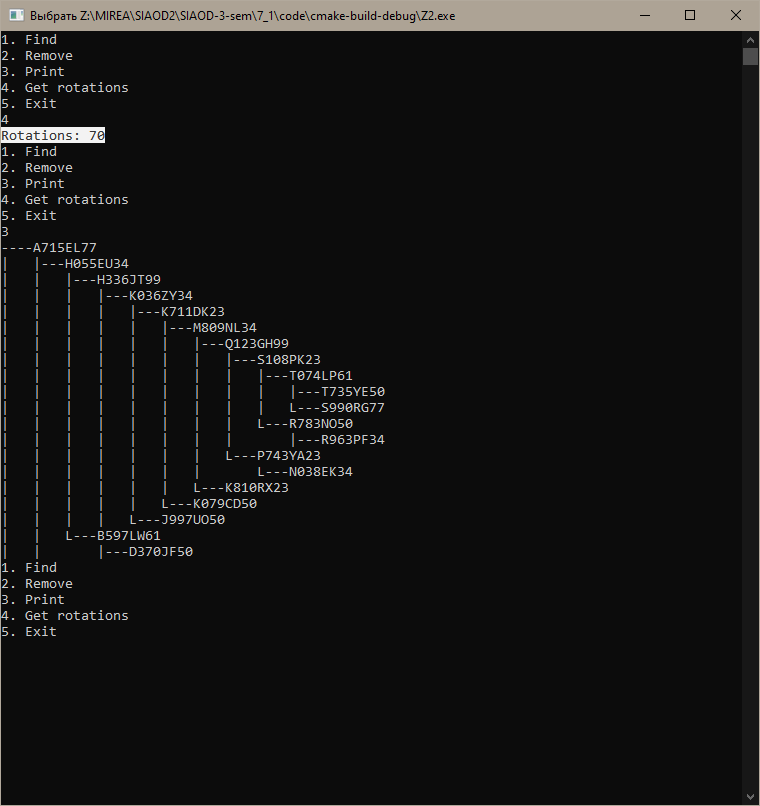


Рисунок 1 – Тестирование приложения №2

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).

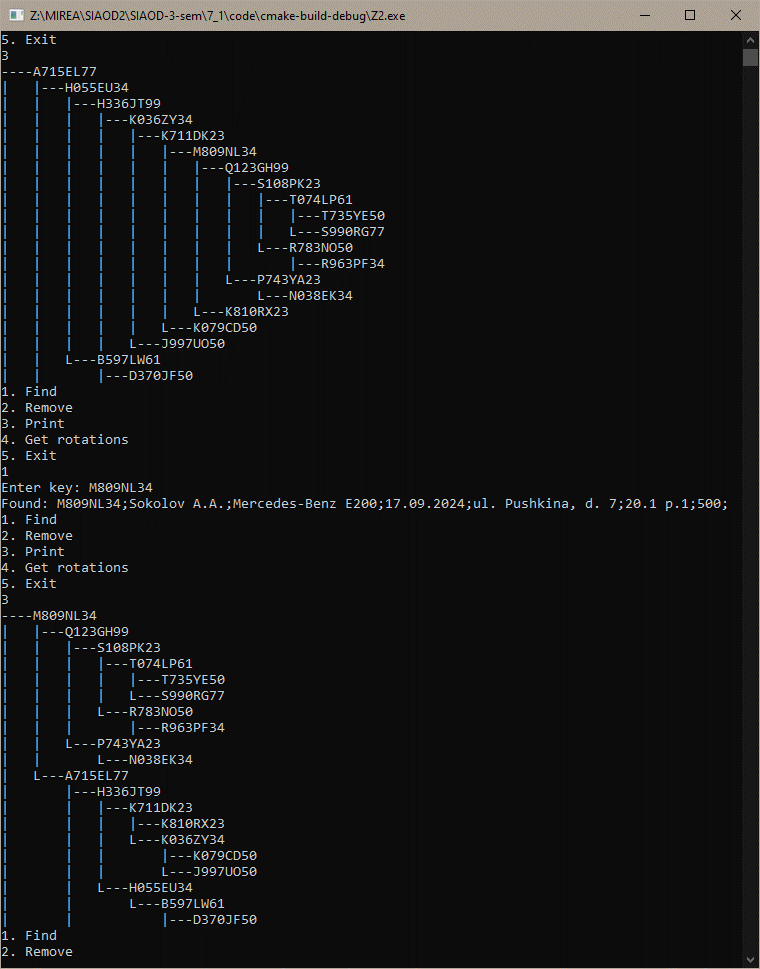


Рисунок 2 – Тестирование приложения №2

### 3.4.3 Удаление элемента

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

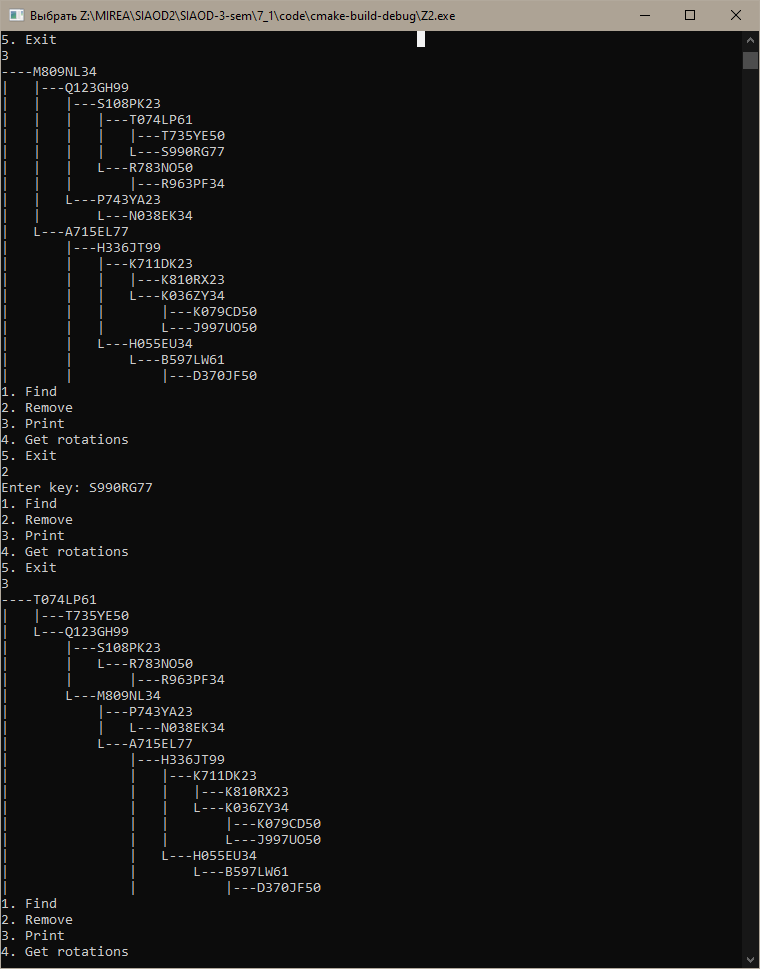


Рисунок 3 – Тестирование приложения №2

# 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:

*#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):

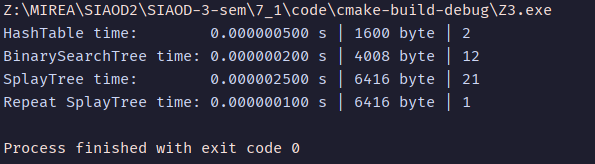


Рисунок 5 – Тестирование для 100 чисел

Для 10000 чисел (рис. 5):

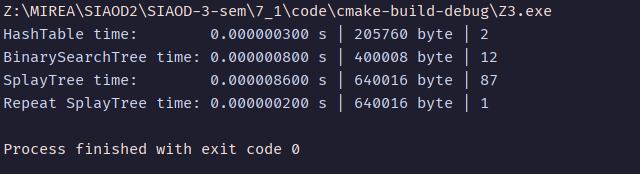


Рисунок 6 – Тестирование для 10000 чисел

## 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