

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 №5**

**по дисциплине**

**«Структуры и алгоритмы обработки данных»**

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

|  |  |  |
| --- | --- | --- |
| Выполнил студент группы ИКБО-04-22 |  | Кликушин В.И. |
| Принял старший преподаватель |  | Скворцова Л.А. |

Москва 2023

ОГЛАВЛЕНИЕ

[1 ЗАДАНИЕ №1 4](#_Toc151655960)

[1.1 УСЛОВИЕ ЗАДАЧИ И ЗАДАНИЕ ВАРИАНТА 4](#_Toc151655961)

[1.2 ПОДХОД К РЕШЕНИЮ 4](#_Toc151655962)

[1.3 ДВОИЧНЫЙ ФАЙЛ ИЗ ЗАПИСЕЙ ФИКСИРОВАННОЙ ДЛИНЫ 5](#_Toc151655963)

[1.4 ОПЕРАЦИИ ПО УПРАВЛЕНИЮ БДП 5](#_Toc151655964)

[1.5 АЛГОРИТМЫ ОПЕРАЦИЙ НА ПСЕВДОКОДЕ 6](#_Toc151655965)

[1.6 КОД ПРИЛОЖЕНИЯ 9](#_Toc151655966)

[1.6.1 Код файла BinaryTree.h 9](#_Toc151655967)

[1.6.2 Код файла BinaryTree.cpp 10](#_Toc151655968)

[1.6.3 Код файла BinFile.h 14](#_Toc151655969)

[1.6.4 Код файла BinFile.cpp 15](#_Toc151655970)

[1.6.5 Код файла TreeFile.h 17](#_Toc151655971)

[1.6.6 Код файла TreeFile.cpp 17](#_Toc151655972)

[1.6.7 Код файла main.cpp 18](#_Toc151655973)

[1.7 СОДЕРЖАНИЕ ФАЙЛА 24](#_Toc151655974)

[1.8 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ 25](#_Toc151655975)

[2 ЗАДАНИЕ №2 30](#_Toc151655976)

[2.1 УСЛОВИЕ ЗАДАЧИ И ЗАДАНИЕ ВАРИАНТА 30](#_Toc151655977)

[2.2 ПОДХОД К РЕШЕНИЮ 30](#_Toc151655978)

[2.3 ОПЕРАЦИИ ПО УПРАВЛЕНИЮ КОСЫМ ДЕРЕВОМ 30](#_Toc151655979)

[2.4 АЛГОРИТМЫ ОПЕРАЦИЙ НА ПСЕВДОКОДЕ 32](#_Toc151655980)

[2.5 КОД ПРИЛОЖЕНИЯ 33](#_Toc151655981)

[2.5.1 Код файла SplayTree.h 33](#_Toc151655982)

[2.5.2 Код файла SplayTree.cpp 34](#_Toc151655983)

[2.5.3 Код файла BinFile.h 38](#_Toc151655984)

[2.5.4 Код файла BinFile.cpp 38](#_Toc151655985)

[2.5.5 Код файла TreeFile.h 38](#_Toc151655986)

[2.5.6 Код файла TreeFile.cpp 38](#_Toc151655987)

[2.5.7 Код файла main.cpp 39](#_Toc151655988)

[2.6 СОДЕРЖАНИЕ ФАЙЛА 44](#_Toc151655989)

[2.7 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ 45](#_Toc151655990)

[3 ЗАДАНИЕ№3 51](#_Toc151655991)

[4 ВЫВОДЫ 52](#_Toc151655992)

[5 ИСПОЛЬЗУЕМЫЕ ИСТОЧНИКИ 53](#_Toc151655993)

# 1 ЗАДАНИЕ №1

## 1.1 УСЛОВИЕ ЗАДАЧИ И ЗАДАНИЕ ВАРИАНТА

Разработайте приложение, которое использует БДП для организации прямого доступа к записям файла, структура записи которого приведена в варианте (отобразить задачу варианта).

Структура записи файла: Cтудент: номер зачетной книжки, номер группы ФИО.

## 1.2 ПОДХОД К РЕШЕНИЮ

Для решения поставленной задачи были реализованы несколько библиотек функций: библиотека бинарного дерева поиска, библиотека для работы с двоичными файлами и библиотека для управления бинарным файлом посредством бинарного дерева поиска.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 1.1 – Структура элемента дерева и структура самого дерева

В структуру элемента входят: переменная строкового типа – ключ записи (номер зачетной книги); позиция элемента в файле; два конструктора.

В структуру бинарного дерева поиска входят объект структуры element, указатель на левое и правое дерево, конструктор.

## 1.3 ДВОИЧНЫЙ ФАЙЛ ИЗ ЗАПИСЕЙ ФИКСИРОВАННОЙ ДЛИНЫ

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Рисунок 1.2 – Структура записи файла

Произведем расчет размера одной структуры в байтах: 40 \* 1 + 40 \* 1 + 80 \* 1 = 160 байт.

Системный расчет размера одной структуры в байтах представлен на рисунке № 2.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 1.3 - Системный расчет размера одной структуры

Операции по управлению бинарным файлом: создать бинарный файл, вывести его в консоль, найти запись в файле по ключу, удалить запись в файле по ключу.

## 1.4 ОПЕРАЦИИ ПО УПРАВЛЕНИЮ БДП

Операции по управлению БДП: включение элемента в дерево, поиск ключа в дереве, удаление ключа из дерева, отображение дерева.

Прототипы функций:

1. Включение элемента в дерево

Предусловие. Ссылка на указатель на объект класса BinaryTree – корень дерева, объект структуры element, который необходимо добавить.

Постусловие. Нет возвращаемого значения.

**void add\_element(BinaryTree\*& root, element elem);**

1. Поиск ключа в дереве

Предусловие. Указатель на объект класса BinaryTree – корень дерева, строковое значение key – ключ записи.

Постусловие. Возвращает позицию элемента в файле.

**int find\_element(BinaryTree\* root, string key);**

1. Отображение дерева

Предусловие. Указатель на объект класса BinaryTree – корень дерева, целочисленная переменная level для вывода отступов.

Постусловие. Нет возвращаемого значения.

**void print\_tree(BinaryTree\* node, int level = 0);**

1. Удаление ключа из дерева

Предусловие. Указатель на объект класса BinaryTree – корень дерева, строковое значение key – ключ записи, Указатель на объект класса BinaryTree – родитель удаляемого узла.

Постусловие. Позиция записи в файле.

**int delete\_element(BinaryTree\*& root, string key, BinaryTree\*& parent);**

## 1.5 АЛГОРИТМЫ ОПЕРАЦИЙ НА ПСЕВДОКОДЕ

Алгоритм вставки ключа:

//предусловие: root – корень дерева (указатель на узел в вершине), key – ключ.

//постусловие: вставляет узел root в дерево root

add\_element(root, key)

{

If (root = NULL) root = new BinaryTree(key);

Else

If (key < root.data.key) add\_element(root.left, key);

Else If (key > root.data.key) add\_element(root.right, key);

endIf

endIf

}

Алгоритм поиска записи по ключу:

*//*предусловие:root – ссылка на указатель на корень дерева, key – ключ.

//постусловие: возвращает позицию элемента в файле или INT\_MIN, если записи с ключом нет в дереве.

find\_element(root, key)

{

If (root = NULL) return INT\_MIN;

If (key < root.data.key) return find\_element(root.left, key);

Else If (key > root.data.key) return find\_element(root.right, key);

Else return root.data.position;

endIf

}

Алгоритм удаления записи с данным ключом:

//предусловие: root – ссылка на указатель на корень дерева, key – ключ поиска, parent – ссылка на указатель на родителя удаляемого узла.

//постусловие: возвращает позицию удалённого элемента в файле или INT\_MIN, если записи с ключом нет в дереве.

delete\_element(root, key, parent)

{

If (root = NULL) return INT\_MIN;

If (key < root.data.key) then

parent <- root;

return delete\_element(root.left, x, parent);

Else If (x > root.data.key) then

Parent <- root;

return deleteElement(root.right, x, parent);

Else

position = root.data.position;

If (root.left = NULL & root.right = NULL) then

If (parent.left = root) then

parent.left = NULL;

delete root;

Else If (parent.right = root) then

parent.right = NULL;

delete root;

endIf

Else If (root.left = NULL & root.right != NULL) then

replace <- root.right;

If (parent.left = root) then

parent.left = replace;

delete root;

Else If (parent.right = root) then

parent.right = replace;

delete root;

endIf

Else If (root.left != NULL & root.right = NULL) then

replace <- root.left;

If (parent.left = root) then

parent.left = replace;

delete root;

Else If (parent.right = root) then

parent.right = replace;

delete root;

endIf

Else

replace <- root.left;

If (replace.right = NULL) replace.right = root.right;

Else

parentReplace <- root.left;

replace <- replace.right;

findReplace(replace, parentReplace);

replace.left = root.left;

replace.right = root.right;

If (parent.left = root) then

parent.left = replace;

delete root;

Else If (parent.right = root) then

parent.right = replace;

delete root;

endIf

endIf

endIf

return position;

endIf

}

## 1.6 КОД ПРИЛОЖЕНИЯ

### 1.6.1 Код файла BinaryTree.h

#pragma once

#include "BinFile.h"

struct element

{

string key;

int position = INT\_MIN;

element() {}

element(string key, int position) : key(key), position(position) {}

};

struct BinaryTree

{

element data;

BinaryTree\* left = nullptr;

BinaryTree\* right = nullptr;

BinaryTree(element elem) : data(elem) {}

};

int& find\_element(BinaryTree\* root, string key);

void add\_element(BinaryTree\*& root, element elem);

int delete\_element(BinaryTree\*& root, string key);

int delete\_element(BinaryTree\*& root, string key, BinaryTree\*& parent);

void find\_replace(BinaryTree\*& replace, BinaryTree\*& parentReplace);

void print\_tree(BinaryTree\* node, int level = 0);

### 1.6.2 Код файла BinaryTree.cpp

#include "BinaryTree.h"

void add\_element(BinaryTree\*& root, element elem)

{

if (root == nullptr)

{

root = new BinaryTree(elem);

}

else

{

if (elem.key.compare(root->data.key)<0)

{

add\_element(root->left, elem);

}

else if (elem.key.compare(root->data.key) >0)

{

add\_element(root->right, elem);

}

}

}

void print\_tree(BinaryTree\* tree, int level)

{

if (tree != nullptr)

{

level += 12;

print\_tree(tree->right, level);

cout << endl;

for (int i = 0; i < level; i++)

cout << " ";

cout << tree->data.key << "\n";

print\_tree(tree->left, level);

}

}

int& find\_element(BinaryTree\* root, string key)

{

if (root == nullptr)

{

int res = INT\_MIN;

return res;

}

if (root->data.key == key)

{

return root->data.position;

}

else if (key.compare(root->data.key)<0)

{

return find\_element(root->left, key);

}

else

{

return find\_element(root->right, key);

}

}

int delete\_element(BinaryTree\*& root, string key)

{

BinaryTree\* parent = nullptr;

int res = delete\_element(root, key, parent);

return res;

}

int delete\_element(BinaryTree\*& root, string key, BinaryTree\*& parent)

{

if (root == nullptr)

{

return INT\_MIN;

}

if (key.compare(root->data.key)<0)

{

parent = root;

int res = delete\_element(root->left, key, parent);

return res;

}

else if (key.compare(root->data.key) > 0)

{

parent = root;

int res = delete\_element(root->right, key, parent);

return res;

}

else

{

int position = root->data.position;

BinaryTree\* deleted = root, \* replace = nullptr, \* parentReplace = nullptr;

if (root->left == nullptr && root->right == nullptr)

{

if (parent == nullptr)

{

root = replace;

}

else if (parent->left == deleted)

{

parent->left = nullptr;

delete deleted;

}

else if (parent->right == deleted)

{

parent->right = nullptr;

delete deleted;

}

}

else if (root->left != nullptr && root->right == nullptr)

{

replace = deleted->left;

if (parent == nullptr)

{

root = replace;

}

else if (parent->left == deleted)

{

parent->left = replace;

delete deleted;

}

else if (parent->right == deleted)

{

parent->right = replace;

delete deleted;

}

}

else if (root->right != nullptr && root->left == nullptr)

{

replace = deleted->right;

if (parent == nullptr)

{

root = replace;

}

else if (parent->left == deleted)

{

parent->left = replace;

}

else if (parent->right == deleted)

{

parent->right = replace;

}

delete deleted;

}

else

{

replace = deleted->left;

if (replace->right == nullptr)

{

replace->right = deleted->right;

}

else

{

parentReplace = deleted->left;

replace = replace->right;

find\_replace(replace, parentReplace);

replace->left = deleted->left;

replace->right = deleted->right;

}

if (parent == nullptr)

{

root = replace;

}

else if (parent->left == deleted)

{

parent->left = replace;

}

else if (parent->right == deleted)

{

parent->right = replace;

}

delete deleted;

}

return position;

}

}

void find\_replace(BinaryTree\*& replace, BinaryTree\*& parentReplace)

{

if (replace->right != nullptr)

{

parentReplace = parentReplace->right;

replace = replace->right;

find\_replace(replace, parentReplace);

}

else

{

parentReplace->right = replace->left;

replace->left = nullptr;

}

}

### 1.6.3 Код файла BinFile.h

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <string>

#include <iostream>

#include <fstream>

#include <chrono>

#include <fcntl.h>

#include <io.h>

using namespace std;

struct Student

{

char studentID[40];

char group[40];

char full\_name[80];

Student();

~Student();

};

void create\_bin\_file(ifstream& output, ofstream& input);

void output\_bin(ifstream& output);

Student get\_element(ifstream& output, int n);

Student\* delete\_element(fstream& f, long long int n, string bin\_file);

### 1.6.4 Код файла BinFile.cpp

#include "BinFile.h"

Student::Student()

{

this->studentID[0] = '\0';

this->group[0] = '\0';

this->full\_name[0] = '\0';

}

Student::~Student()

{

this->studentID[0] = '\0';

this->group[0] = '\0';

this->full\_name[0] = '\0';

}

void create\_bin\_file(ifstream& output, ofstream& input)

{

while (!output.eof())

{

Student Vlad;

output.getline(Vlad.studentID, 40);

output.getline(Vlad.group, 40);

output.getline(Vlad.full\_name, 80);

input.write((char\*)&Vlad, sizeof(Student));

}

}

void output\_bin(ifstream& output)

{

Student Vlad;

output.read((char\*)&Vlad, sizeof(Student));

while (!output.eof())

{

cout << Vlad.studentID << " | " << Vlad.group << " | " << Vlad.full\_name << "\n";

output.read((char\*)&Vlad, sizeof(Vlad));

}

}

Student get\_element(ifstream& output, int n)

{

Student Vlad;

output.seekg(0, ios::end);

int size = output.tellg();

if (n \* sizeof(Student) >= size || n < 0)

{

return Student();

}

output.seekg(n \* sizeof(Student), ios::beg);

output.read((char\*)&Vlad, sizeof(Student));

return Vlad;

}

Student\* delete\_element(fstream& f, long long int n, string bin\_file)

{

f.seekg(0, ios::end);

int size = f.tellg();

if (n \* sizeof(Student) >= size && n >= 0)

{

return nullptr;

}

Student\* lastRecord = new Student(), \* record = new Student();

f.seekg((-1) \* sizeof(Student), ios::end);

f.read((char\*)lastRecord, sizeof(Student));

f.seekg(n \* sizeof(Student), ios::beg);

f.read((char\*)record, sizeof(Student));

record = lastRecord;

f.seekg((-1) \* sizeof(Student), ios::cur);

f.write((char\*)record, sizeof(Student));

f.close();

int desc = \_open(bin\_file.c\_str(), O\_RDWR);

\_chsize\_s(desc, size - sizeof(Student));

\_close(desc);

f.open(bin\_file, ios::binary);

return lastRecord;

}

### 1.6.5 Код файла TreeFile.h

#pragma once

#include "BinaryTree.h"

#include "BinFile.h"

void create\_binary\_tree(BinaryTree\*& tree, ifstream& output);

Student find\_el(BinaryTree\*& tree, ifstream& output, string key);

void delete\_el(BinaryTree\*& tree, fstream& file, string key, string filename);

### 1.6.6 Код файла TreeFile.cpp

#include "TreeFile.h"

void create\_binary\_tree(BinaryTree\*& tree, ifstream& file)

{

Student Vlad;

element elem;

int pos = 0;

file.read((char\*)&Vlad, sizeof(Student));

while (!file.eof())

{

elem.position = pos;

elem.key = Vlad.studentID;

add\_element(tree, elem);

pos++;

file.read((char\*)&Vlad, sizeof(Student));

}

}

Student find\_el(BinaryTree\*& tree, ifstream& file, string key)

{

long long int n = find\_element(tree, key);

if (n == INT\_MIN)

return Student();

return get\_element(file, n);

}

void delete\_el(BinaryTree\*& tree, fstream& file, string key, string filename)

{

long long int n = delete\_element(tree, key);

Student\* Vlad = delete\_element(file, n, filename);

if (Vlad == nullptr)

{

return;

}

if (Vlad->studentID != key)

{

element elem(Vlad->studentID, n);

delete\_element(tree,elem.key);

add\_element(tree, elem);

}

}

### 1.6.7 Код файла main.cpp

#include "BinFile.h"

#include "BinaryTree.h"

#include "TreeFile.h"

int main()

{

system("chcp 1251");

BinaryTree\* tree = nullptr;

int num;

ifstream output;

ofstream input;

string first\_file, second\_file;

fstream f;

string key;

while (true)

{

cout << "-----------------------------------------------------------------------------------------------------------" << endl;

cout << " Меню\n";

cout << " 1. Преобразовать текстовый файл в двоичный\n";

cout << " 2. Перевод двоичного файла в бинарное дерево поиска\n";

cout << " 3. Поиск записи\n";

cout << " 4. Удаление записи\n";

cout << " 5. Вывести бинарное дерево поиска\n";

cout << " 6. Вывести содержимое двоичного файла в консоль\n";

cout << " 7. Завершить выполнение программы\n";

cin >> num;

switch (num)

{

case 1:

{

cout << " Введите название текстового файла: ";

cin >> first\_file;

cout << " Введите название двоичного файла: ";

cin >> second\_file;

output.open(first\_file);

if (!output.is\_open())

{

cout << " Текстовый файл не открыт\n";

}

else

{

input.open(second\_file, ios::binary);

create\_bin\_file(output, input);

if (!input.good())

{

cout << " Ошибка записи в двоичный файл\n";

return -1;

}

else

{

output.close();

input.close();

}

}

break;

}

case 2:

{

cout << " Введите название двоичного файла: ";

cin >> first\_file;

output.open(first\_file, ios::binary);

if (output.is\_open())

{

create\_binary\_tree(tree, output);

if (output.bad())

{

cout << " Возникла ошибка создания дерева поиска\n";

return 1;

}

else

{

cout << " Создание дерева поиска произошло успешно\n";

output.close();

}

}

else

{

cout << " Двоичного файла не существует\n";

}

break;

}

case 3:

{

cout << " Введите название двоичного файла: ";

cin >> first\_file;

output.open(first\_file, ios::binary);

if (output.is\_open())

{

cout << "Введите номер зачетной книжки: ";

cin.get();

getline(cin, key);

auto start = chrono::high\_resolution\_clock::now();

Student stud = find\_el(tree, output, key);

auto end = chrono::high\_resolution\_clock::now();

cout << " Время поиска: " << chrono::duration\_cast<chrono::nanoseconds>(end - start).count() / 1e6 << " мс\n";

if (output.bad())

{

cout << " Возникла ошибка чтения\n";

return 1;

}

else

{

if (stud.studentID[0] != '\0')

cout << stud.studentID << " | " << stud.group << " | " << stud.full\_name << "\n";

else

cout << " Запись не найдена\n";

output.close();

}

}

else

{

cout << " Двоичный файл не существует\n";

}

break;

}

case 4:

{

cout << "Введите название двоичного файла: ";

cin >> first\_file;

f.open(first\_file, ios::in | ios::out | ios::binary);

if (f.is\_open())

{

string key;

cout << " Введите номер зачетной книжки: ";

cin.get();

getline(cin, key);

delete\_el(tree, f, key, first\_file);

if (output.bad())

{

cout << " Возникла ошибка при удалении\n";

return 1;

}

else

{

cout << " Запись успешно удалена!\n";

f.close();

}

}

else

{

cout << " Двоичный файл не существует\n";

}

break;

}

case 5:

{

print\_tree(tree,0);

break;

}

case 6:

{

cout << " Введите название двоичного файла: ";

cin >> first\_file;

output.open(first\_file, ios::binary);

if (!output.is\_open())

{

cout << " Двоичный файл не открыт\n";

}

else

{

output\_bin(output);

if (output.bad())

{

cout << " Ошибка при чтении из двоичного файла\n";

return 1;

}

else

{

output.close();

}

}

break;

}

case 7:

return 0;

default:

{

cout << " Нет такого пункта\n";

}

}

}

}

## 1.7 СОДЕРЖАНИЕ ФАЙЛА

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 1.4 – Содержание текстового файла на 12 записей

## 1.8 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 1.5 – Тестирование создания бинарного файла на основе текстового

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 1.6 – Создание и вывод бинарного дерева поиска на основе двоичного файла

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 1.7 – Поиск записи (узел – корень)

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 1.8 – Поиск записи (узел – лист)

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 1.9 – Поиск несуществующей записи

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 1.10 – Удаление записи (лист)

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 1.11 – Удаление записи (есть оба сына)

# 2 ЗАДАНИЕ №2

## 2.1 УСЛОВИЕ ЗАДАЧИ И ЗАДАНИЕ ВАРИАНТА

Разработать приложение, которое использует сбалансированное дерево поиска – косое дерево для доступа к записям файла.

Структура записи файла: Cтудент: номер зачетной книжки, номер группы ФИО.

## 2.2 ПОДХОД К РЕШЕНИЮ

Для решения поставленной задачи были реализованы несколько библиотек функций: библиотека косого дерева, библиотека для работы с двоичными файлами и библиотека для управления бинарным файлом посредством косого дерева.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2.1 – Структура элемента дерева и структура самого дерева

В структуру элемента входят: переменная строкового типа – ключ записи (номер зачетной книги); позиция элемента в файле; два конструктора.

В структуру косого дерева входят объект структуры element, указатель на левое и правое дерево, конструкторы.

## 2.3 ОПЕРАЦИИ ПО УПРАВЛЕНИЮ КОСЫМ ДЕРЕВОМ

Операции по управлению косым деревом: включение элемента в дерево, поиск ключа в дереве, удаление ключа из дерева, отображение дерева, левый и правый повороты, splay – перемещение узла в корень.

Прототипы функций:

1. Включение элемента в дерево

Предусловие. Указатель на объект класса SplayTree – корень дерева, объект структуры element, который необходимо добавить.

Постусловие. Указатель на объект структуры SplayTree.

**SplayTree\* insertToSplay(SplayTree\* root, element elem);**

1. Поиск ключа в дереве

Предусловие. Ссылка на указатель на объект класса SplayTree – корень дерева, строковое значение key – ключ записи.

Постусловие. Возвращает позицию элемента в файле.

**int& find\_element(SplayTree\*& root, string key);**

1. Отображение дерева

Предусловие. Указатель на объект класса SplayTree – корень дерева, целочисленная переменная level для вывода отступов.

Постусловие. Нет возвращаемого значения.

**void print\_tree(SplayTree\* node, int level = 0);**

1. Удаление ключа из дерева

Предусловие. Ссылка на указатель на объект класса SplayTree – корень дерева, строковое значение key – ключ записи.

Постусловие. Позиция записи в файле.

**int delFromSplayTree(SplayTree\*& root, string key);**

1. Левый поворот

Предусловие. Указатель на объект класса SplayTree – родитель текущего узла дерева.

Постусловие. Указатель на объект класса SplayTree.

**SplayTree\* LeftRotation(SplayTree\* root);**

1. Правый поворот

Предусловие. Указатель на объект класса SplayTree – родитель текущего узла дерева.

Постусловие. Указатель на объект класса SplayTree.

**SplayTree\* RightRotation(SplayTree\* root);**

1. Перемещение узла в корень

Предусловие. Указатель на объект класса SplayTree – корень дерева, ключ записи.

Постусловие. Указатель на объект класса SplayTree.

**SplayTree\* splay(SplayTree\* root, string key);**

## 2.4 АЛГОРИТМЫ ОПЕРАЦИЙ НА ПСЕВДОКОДЕ

Алгоритм вставки ключа:

//предусловие: root – корень дерева (указатель на узел в вершине), elem – элемент (ключ).

//постусловие: вставляет узел elem в дерево root

insertToSplay(root, elem)

{

If root = nullptr then

root = new SplayTree;

root->data.key = elem.key;

root->data.position = elem.position;

return root;

endIF

root = splay(root, elem.key);

If (root->data.key = elem.key) return root;

SplayTree\* newnode = new SplayTree;

newnode->data.key = elem.key;

newnode->data.position = elem.position;

If root->data.key.compare> elem.key then

newnode->right = root;

newnode->left = root->left;

root->left = nullptr;

endIF

Else

newnode->left = root;

newnode->right = root->right;

root->right = nullptr;

return newnode;

}

Алгоритм поиска записи по ключу:

*//*предусловие:root – ссылка на указатель на корень дерева, key – ключ.

//постусловие: возвращает позицию элемента в файле или INT\_MIN, если записи с ключом нет в дереве.

find\_element(root, key)

{

root = splay(root, key);

If root->data.key = key then

return root->data.position;

endIF

res = INT\_MIN;

return res;

}

Алгоритм удаления записи с данным ключом:

//предусловие: root – ссылка на указатель на корень дерева, key – ключ поиска.

//постусловие: возвращает позицию удалённого элемента в файле или INT\_MIN, если записи с ключом нет в дереве.

delete\_element(root, key)

{

}

## 2.5 КОД ПРИЛОЖЕНИЯ

### 2.5.1 Код файла SplayTree.h

#include "BinFile.h"

struct element

{

string key;

int position = INT\_MIN;

element() {}

element(string key, int position) : key(key), position(position) {}

};

struct SplayTree

{

element data;

SplayTree\* left = nullptr;

SplayTree\* right = nullptr;

SplayTree(element elem) : data(elem) {}

SplayTree() {};

};

SplayTree\* LeftRotation(SplayTree\* root);

SplayTree\* RightRotation(SplayTree\* root);

SplayTree\* splay(SplayTree\* root, string key);

SplayTree\* insertToSplay(SplayTree\* root, element elem);

int delFromSplayTree(SplayTree\*& root, string key);

void print\_tree(SplayTree\* tree, int level);

int& find\_element(SplayTree\*& root, string key);

### 2.5.2 Код файла SplayTree.cpp

#include "SplayTree.h"

SplayTree\* LeftRotation(SplayTree\* root)

{

SplayTree\* X = root->right;

root->right = X->left;

X->left = root;

return X;

}

SplayTree\* RightRotation(SplayTree\* root)

{

SplayTree\* X = root->left;

root->left = X->right;

X->right = root;

return X;

}

SplayTree\* splay(SplayTree\* root, string key)

{

// Базовые случаи: root равен NULL или

// ключ находится в корне

if (root == nullptr || root->data.key == key)

return root;

// Ключ может находиться в левом поддереве

if (root->data.key.compare(key) > 0)

{

// Ключа нет в дереве, завершение

if (root->left == nullptr) return root;

// Zig-Zig (Левый-левый)

if (root->left->data.key.compare(key) > 0)

{

// Сначала рекурсивно поднимем ключ в качестве корня left-left

root->left->left = splay(root->left->left, key);

// Первый поворот для root, второй поворот выполняется после else

root = RightRotation(root);

}

else if (root->left->data.key.compare(key) < 0) // Zig-Zag (Left-Right)

{

// Сначала рекурсивно поднимаем ключ в качестве корня left-right

root->left->right = splay(root->left->right, key);

// Выполняем первый поворот для root->left

if (root->left->right != nullptr)

root->left = LeftRotation(root->left);

}

// Выполняем второй поворот для корня

return (root->left == nullptr) ? root : RightRotation(root);

}

else // Ключ может находиться в правом поддереве

{

// Ключа нет в дереве, завершение

if (root->right == nullptr) return root;

if (root->right->data.key.compare(key) > 0) // Zag-Zig (Left-Right)

{

// Поднять ключ в качестве корня right-left

root->right->left = splay(root->right->left, key);

// Выполняем первый поворот для root->right

if (root->right->left != nullptr)

root->right = RightRotation(root->right);

}

else if (root->right->data.key.compare(key) < 0)// Zag-Zag (Правый-правый)

{

// Поднимаем ключ в качестве корня right-right

root->right->right = splay(root->right->right, key);

//Выполняем первый поворот

root = LeftRotation(root);

}

// Выполняем второй поворот для root

return (root->right == nullptr) ? root : LeftRotation(root);

}

}

SplayTree\* insertToSplay(SplayTree\* root, element elem)

{

// случай 1: дерево пусто, создаем первый узел с ключом key

if (root == nullptr)

{

root = new SplayTree();

root->data.key = elem.key;

root->data.position = elem.position;

return root;

}

// случай 2. Дерево не пусто. Перемещаем ближайший узел-лист в корень

root = splay(root, elem.key);

//Если в корне ключ равный искомому (найден), то возвращаем дерево с этим

//ключом в корне

if (root->data.key == elem.key) return root;

// Если ключа в дереве нет (ключ в корне не равен искомому), то создаем

//новый узел

SplayTree\* newnode = new SplayTree;

newnode->data.key = elem.key;

newnode->data.position = elem.position;

//Так как этот узел будет корнем, то надо определить куда перевесить

//существующие поддеревья

// Если ключ в корне больше искомого, делаем бывший корень правым

//дочерним элементом нового узла,

//делаем левый дочерний элемент бывшего корня

//левым дочернего элемента нового узла

if (root->data.key.compare(elem.key) > 0)

{

newnode->right = root;

newnode->left = root->left;

root->left = nullptr;

}

// Если ключ в корне меньше, вставляемого узла, то бывший корень дерева

// становится левым дочерним элементом нового узла, а правый дочерний

// элемент, бывшего корня правым дочерним элементом нового узла

else

{

newnode->left = root;

newnode->right = root->right;

root->right = nullptr;

}

return newnode; // новый узел становится новым корнем

}

int delFromSplayTree(SplayTree\*& root, string key)

{

int res = INT\_MIN;

if (root == nullptr) return res; // дерево пусто

root = splay(root, key);

if (root->data.key == key)

res = root->data.position;

int cmp = (key == root->data.key) ? 0 : ((key.compare(root->data.key) < 0) ? -1 : 1);

if (cmp == 0)

{//ключ в корне

if (root->left == nullptr)

{

root = root->right;

}

else

{

SplayTree\* x = root->right;

root = root->left;

root = splay(root, key);

root->right = x;

}

}

return res;

}

void print\_tree(SplayTree\* tree, int level)

{

if (tree != nullptr)

{

level += 12;

print\_tree(tree->right, level);

cout << endl;

for (int i = 0; i < level; i++)

cout << " ";

cout << tree->data.key << "\n";

print\_tree(tree->left, level);

}

}

int& find\_element(SplayTree\*& root, string key)

{

root = splay(root, key);

if (root->data.key == key)

return root->data.position;

int res = INT\_MIN;

return res;

}

### 2.5.3 Код файла BinFile.h

Аналогичен коду BinFile.h из задания №1

### 2.5.4 Код файла BinFile.cpp

Аналогичен коду BinFile.cpp из задания №1

### 2.5.5 Код файла TreeFile.h

#pragma once

#include "SplayTree.h"

#include "BinFile.h"

void create\_splay\_tree(SplayTree\*& tree, ifstream& output);

Student find\_el(SplayTree\*& tree, ifstream& output, string key);

void delete\_el(SplayTree\*& tree, fstream& file, string key, string filename);

### 2.5.6 Код файла TreeFile.cpp

#include "TreeFile.h"

void create\_splay\_tree(SplayTree\*& tree, ifstream& file)

{

Student Vlad;

element elem;

int pos = 0;

file.read((char\*)&Vlad, sizeof(Student));

while (!file.eof())

{

elem.position = pos;

elem.key = Vlad.studentID;

tree = insertToSplay(tree, elem);

pos++;

file.read((char\*)&Vlad, sizeof(Student));

}

}

Student find\_el(SplayTree\*& tree, ifstream& file, string key)

{

long long int n = find\_element(tree, key);

if (n == INT\_MIN)

return Student();

return get\_element(file, n);

}

void delete\_el(SplayTree\*& tree, fstream& file, string key, string filename)

{

long long int n = delFromSplayTree(tree, key);

Student\* Vlad = delete\_element(file, n, filename);

if (Vlad == nullptr)

{

return;

}

if (Vlad->studentID != key)

{

element elem(Vlad->studentID, n);

delFromSplayTree(tree, elem.key);

insertToSplay(tree, elem);

}

}

### 2.5.7 Код файла main.cpp

#include "BinFile.h"

#include "TreeFile.h"

int main()

{

system("chcp 1251");

SplayTree\* tree = nullptr;

int num;

ifstream output;

ofstream input;

string first\_file, second\_file;

fstream f;

string key;

while (true)

{

cout << "-----------------------------------------------------------------------------------------------------------" << endl;

cout << " Меню\n";

cout << " 1. Преобразовать текстовый файл в двоичный\n";

cout << " 2. Перевод двоичного файла в косое дерево\n";

cout << " 3. Поиск записи\n";

cout << " 4. Удаление записи\n";

cout << " 5. Вывести косое дерево\n";

cout << " 6. Вывести содержимое двоичного файла в консоль\n";

cout << " 7. Завершить выполнение программы\n";

cin >> num;

switch (num)

{

case 1:

{

cout << " Введите название текстового файла: ";

cin >> first\_file;

cout << " Введите название двоичного файла: ";

cin >> second\_file;

output.open(first\_file);

if (!output.is\_open())

{

cout << " Текстовый файл не открыт\n";

}

else

{

input.open(second\_file, ios::binary);

create\_bin\_file(output, input);

if (!input.good())

{

cout << " Ошибка записи в двоичный файл\n";

return -1;

}

else

{

output.close();

input.close();

}

}

break;

}

case 2:

{

cout << " Введите название двоичного файла: ";

cin >> first\_file;

output.open(first\_file, ios::binary);

if (output.is\_open())

{

create\_splay\_tree(tree, output);

if (output.bad())

{

cout << " Возникла ошибка создания дерева поиска\n";

return 1;

}

else

{

cout << " Создание дерева поиска произошло успешно\n";

output.close();

}

}

else

{

cout << " Двоичного файла не существует\n";

}

break;

}

case 3:

{

cout << " Введите название двоичного файла: ";

cin >> first\_file;

output.open(first\_file, ios::binary);

if (output.is\_open())

{

cout << "Введите номер зачетной книжки: ";

cin.get();

getline(cin, key);

auto start = chrono::high\_resolution\_clock::now();

Student stud = find\_el(tree, output, key);

auto end = chrono::high\_resolution\_clock::now();

//cout << " Время поиска: " << chrono::duration\_cast<chrono::nanoseconds>(end - start).count() / 1e6 << " мс\n";

if (output.bad())

{

cout << " Возникла ошибка чтения\n";

return 1;

}

else

{

if (stud.studentID[0] != '\0')

cout << stud.studentID << " | " << stud.group << " | " << stud.full\_name << "\n";

else

cout << " Запись не найдена\n";

output.close();

}

}

else

{

cout << " Двоичный файл не существует\n";

}

break;

}

case 4:

{

cout << "Введите название двоичного файла: ";

cin >> first\_file;

f.open(first\_file, ios::in | ios::out | ios::binary);

if (f.is\_open())

{

string key;

cout << " Введите номер зачетной книжки: ";

cin.get();

getline(cin, key);

delete\_el(tree, f, key, first\_file);

if (output.bad())

{

cout << " Возникла ошибка при удалении\n";

return 1;

}

else

{

cout << " Запись успешно удалена!\n";

f.close();

}

}

else

{

cout << " Двоичный файл не существует\n";

}

break;

}

case 5:

{

print\_tree(tree, 0);

break;

}

case 6:

{

cout << " Введите название двоичного файла: ";

cin >> first\_file;

output.open(first\_file, ios::binary);

if (!output.is\_open())

{

cout << " Двоичный файл не открыт\n";

}

else

{

output\_bin(output);

if (output.bad())

{

cout << " Ошибка при чтении из двоичного файла\n";

return 1;

}

else

{

output.close();

}

}

break;

}

case 7:

return 0;

default:

{

cout << " Нет такого пункта\n";

}

}

}

}

## 2.6 СОДЕРЖАНИЕ ФАЙЛА

Аналогично содержанию файла в задании №1

## 2.7 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2.3 – Количество поворотов на вставку 1000 элементов

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 2.4 – Построение и вывод косого дерева

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 2.5 – Поиск записи (корень)

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 2.6 – Поиск записи (поддерево)

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 2.7 – Поиск записи (лист)

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 2.8 – Поиск несуществующей записи

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 2.9 – Удаление записи

# 3 ЗАДАНИЕ№3

|  |  |  |  |
| --- | --- | --- | --- |
| Вид поисковой структуры | Количество элементов, загруженных в структуру в момент выполнения поиска | Емкостная сложность: объем памяти для структуры | Количество выполненных сравнений, время на поиск ключа в структуре |
| Хэш-таблица | 1000 | O(n) | 2, 0.007 мс |
| БДП | 1000 | O(n) | 20, 0.0733 мс |
| Косое дерево | 1000 | O(n) | 29, 0.0167 мс |
| Хэш-таблица | 1000000 | O(n) | 2, 0.009 мс |
| БДП | 1000000 | O(n) | 18, 0.0829 мс |
| Косое дерево | 1000000 | O(n) | 64, 0.0736 мс |

По таблице видно, что хеш таблица является наиболее эффективной структурой для поиска записей, а косое дерево в свою очередь обгоняет БДП.

# 4 ВЫВОДЫ

В ходе выполнения практической работы были получены навыки работы со splay деревом и бинарным деревом поиска, проведены тестирования на выявления наиболее эффективной структуры для хранения и поиска записей.

# 5 ИСПОЛЬЗУЕМЫЕ ИСТОЧНИКИ

1. Учебно-методическое пособие СиАОД (часть 2)
2. Приложение к практическим работам – СДО (online-edu.mirea.ru)