**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №5**

**по дисциплине «Программирование»**

Тема: Сбалансированные деревья

|  |  |  |
| --- | --- | --- |
| Студент гр. 9894 |  | Леушкин С.М. |
| Преподаватель |  | Глущенко А.Г. |

Санкт-Петербург

2020

# **Цель работы.**

Изучение свойств и организации сбалансированных деревьев. Получение практических навыков в работе с АВЛ-деревьями. Определение преимуществ и недостатков подобных структур данных. Проведение сравнительной характеристики скорости вставки, удаления и поиска элемента в АВЛ-деревьях.

# **Основные теоретические положения.**

Сбалансированным называется такое двоичное дерево поиска, в котором высота каждого из поддеревьев, имеющих общий корень, отличается не более чем на некоторую константу, и при этом выполняется условия, характерные для двоичного дерева [1].

АВЛ-дерево – сбалансированное двоичное дерево поичка с константой отличия, равной 1. Для узлов АВЛ-дерева определен коэффициент сбалансированности. Это разность высот правого и левого поддеревьев, принимающая одно значения из множества {-1, 0, 1}. Графическое представление АВЛ-дерева представлено на рисунке 1.

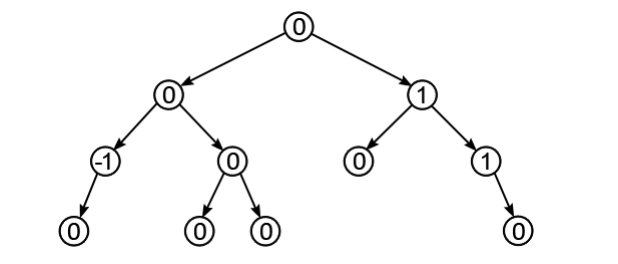


Рисунок 1 – Графическое представление бинарного дерева

Отличительной особенностью сбалансированного дерева является эфективность в обработке, потому как максимальное количество шагов, которое необходимо для обнаружения нужного узла, равно количеству уровней самого бинарного дерева поиска. Вследствии симметричности поддервьев, высота дерева сводится к оптимальному минимуму. Это позволяет ускорить поиск, что сказывается на общей производительности.

Основные операции над стеками:

* добавление узла в дерево;
* поиск узла по дереву;
* удаление узла;
* балансировка дерева.

Структура узла будет состоять из адресной и информационной частей. Информационная часть представляет из себя целочисленные данные узла и высоту поддерева. Адресная, в свою очередь, в качестве аргументов принимает ссылки на своих потомков. Общий вид подобной структуры представлен на рисунке 2.

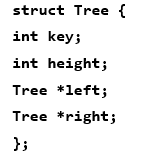


Рисунок 2 – Общий вид структуры данных АВЛ-дерева

В процессе обработки АВЛ-дерева, балансировка может нарушиться, тогда потребуется осуществить операцию балансировки.

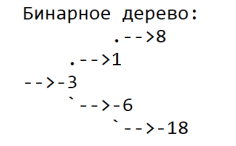
Операция балансировки осуществляется путем поворота узлов (изменения связей в поддереве). Вращения не меняют свойств бинарного дерева поиска. Выделяют несколько типов вращения:

1. малый правый поворот;
2. малый левый поворот;
3. большой правый поворот;
4. большой левый поворот.

# **Постановка задачи.**

Необходимо реализовать программу, которая выполняет следующие действия:

1. Формирование АВЛ-дерева из *N* элементов, заполнение которого можно реализовать автоматически (пользователь вводит количество элементов в дереве, который будет заполняться автоматически случайными числами в диапазоне от -99 до 99), в «ручном» режиме (пользователь вводит в консоль элементы дерева, *N* определяется автоматически по количеству введенных элементов) и с помощью текстового файла (дерево считывается с файла, *N* определяется как количество узлов дерева в текстовом файле).
2. Вывод в консоль и файл tree бинарного дерева. Бинарное дерево должно иметь подобный вид:



1. Определение скорости вставки, удаления и получения элемента дерева.
2. Определение скорости проверки на сбалансированность.

Также необходимо реализовать индивидуальное задание – удалить все отрицательные числа. Каждый нечетный элемент необходимо умножить на случайную величину в диапазоне от -2 до 2.

При выполнении пунктов не должны нарушаться правила построения бинарного дерева. Также должна присутствовать возможность запуска каждого пункта многократно, если есть возможность (если в дереве нет элементов, то нельзя ничего удалить и об этом нужно сообщить пользователю).

# **Выполнение работы.**

Листинг программы представлен в приложении А.

Доступ к функциям программы предоставляется с помощью меню, в котором необходимо выбрать один из пунктов. Меню реализовано в функции menu(). Выбор пунктов меню реализовано в функциях count\_val() и menu\_choice(). Дополнительные подменю также реализованы для выбора способа ввода данных в дерево.

Заполнение дерева выполнено тремя способами:

1. Автоматически – пользователю необходимо ввести количество элементов дерева, а заполнение будет производится случайными значениями в диапазоне от -99 до 99. Способ реализован в функции fill\_root\_random().
2. «Ручным» способом – пользователь вводит самостоятельно элементы дерева. Для того, чтобы остановить ввод элементов, необходимо ввести любой нечисловой символ. Способ реализован в функции fill\_root\_self().
3. Заполнение с помощью текстового файла. Способ реализован в функции fill\_root\_from\_file().

АВЛ-дерево реализовано и имеет структуру Tree.

Требования к текстовому файлу: между символами ставится пробел, постронних знаков быть не должно.

Вывод бинарного дерева в консоль реализован в функции printTree(). Вывод АВЛ-дерева в текстовый файл реализован в фунции printTree\_file(). Обе функции реализованы на рекурсивных алгоритмах.

Управление удалением элемента реализовано в функции delete\_node(), а само удаление реализовано в функции Delete(). На вход функция принимает дерево и значение, которое необходимо удалить. После удаления функция проводит балансировку и возвращает сбалансированное дерево.

Управление вставкой элемента реализовано в функции insert\_node(), а сама вставка реализована в функции Insert(). На вход функция принимает дерево и значение, которое необходимо вставить. После вставки функция проводит балансировку и возвращает сбалансированное дерево.

Управление поиском элемента реализовано в функции find\_node(), а сам поиск реализован в фунции getTreebyValue(). На вход функция принимает дерево и значение, которое необходимо найти. Возвращает искомый узел. Если узел найден не был возвращает NULL.

Вычисление скорости проверки на сбалансированность дерева реализовано в функции correctBalance().

Индивидуальное задание реализовано в функции individual(). Само изменение элементов происходит в функции individual\_ex(). После изменения данные проверяются на повторения, и, при необходимости, повторяющиеся значения удаляются и дерево формируется заново. Данная проверка реализована в функции getTreeNoRepeatValues().

## Тестирование программы.

На рисунках 3 – 8 представлены скриншоты тестирования программы.

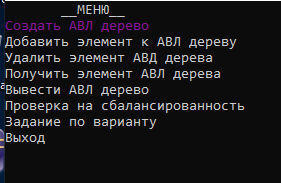


Рисунок 3 – Скриншот меню программы

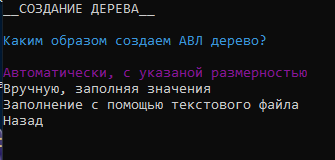


Рисунок 4 – Скриншот подменю создания дерева

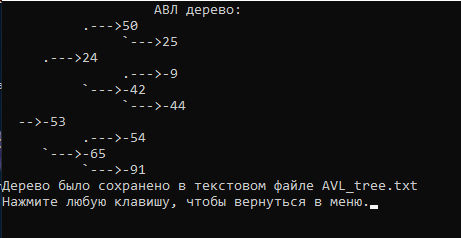


Рисунок 5 – Скриншот выбора пункта «Вывести АВЛ дерево»

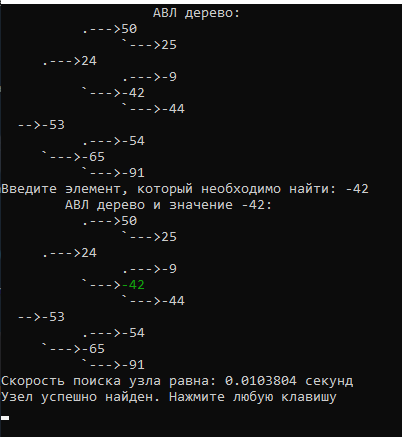


Рисунок 6 – Скриншот поиска элемента в АВЛ-дереве

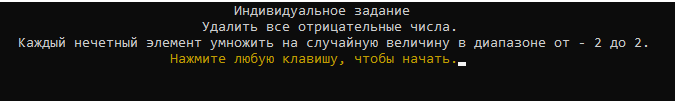


Рисунок 7 – Скриншот пояснения к индвидуальному заданию

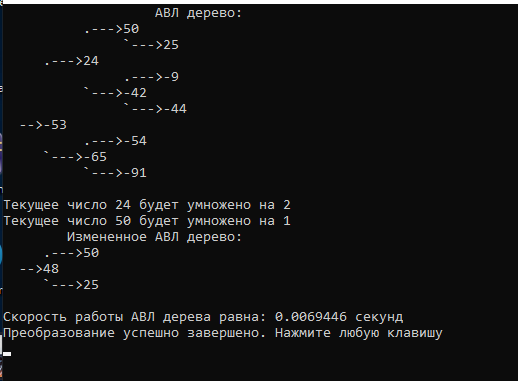


Рисунок 8 – Скриншот выполнения индвидиуального задания

# **Выводы.**

Были освоены теоретические знания и закреплены практические навыки по работе со сбалансированным бинарным деревом поиска. Было проведено исследование по практическому применению бинарных деревьев. Несомненное преимущество АВЛ-деревьев перед простыми бинарными деревьями поиска состоит в оптимальной минимальной высоте таких деревьев, что положительно сказывается на быстродействии.

# **Список использованных источников.**

1. Основы алгоритмизации и программирования на языке C++. Методические указания к практическим работам / сост.: А.Г. Глущенко. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2020. 126с.

Приложение А

ЛИСТИНГ ПРОГРАММЫ

////////////// Лабораторная работа № 5 – АВЛ-дерево

#include <iostream>

#include <vector>

#include <set>

#include <chrono>

#include <windows.h>

#include <conio.h>

#include <iomanip>

#include <fstream>

using namespace std;

int m\_count = 0;

int subm\_crT\_count = 0;

int size\_count = 0;

bool fl\_tree = FALSE;

void SetColor(int text, int bg) {

HANDLE hStdOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hStdOut, (WORD)((bg << 4) | text));

}

int checkdigit() {

while (true) {

int value;

// вводим число, которое хотим представить в двоичной форме

cin >> value; // число целое

if (cin.fail()) { // ecли предыдущее извелечение оказлось неудачным,

cin.clear(); // то возвращаем cin в обычный режим работы

cin.ignore(32767, '\n');

cout << "Недопустимое заданное число. Введите число правильно" << '\n';

}

else {

cin.ignore(32767, '\n');

return value;

}

}

}

///////////////////////// PROTOTYPES //////////////////////////

struct Tree;

void menu(Tree\*& root);

void submenu\_create\_tree(Tree\*& root);

void menu\_choice(Tree\*& root);

void submenu\_create\_tree\_choice(Tree\*& root);

void conf\_val(Tree\*& root);

void subconf\_crT\_val(Tree\*& root);

///////////////////////// PROTOTYPES //////////////////////////

struct Tree {

int key;

char height;

Tree\* right;

Tree\* left;

Tree(int data) { key = data; height = 1; left = right = 0; }

};

char height(Tree\* root) {

if (root) return root->height;

else return 0;

}

int BF(Tree\* root) {

return height(root->right) - height(root->left);

}

void OverHeight(Tree\* root) {

char hleft = height(root->left);

char hright = height(root->right);

root->height = (hleft > hright ? hleft : hright) + 1;

}

Tree\* RightRotation(Tree\* x) {

Tree\* y = x->left;

x->left = y->right;

y->right = x;

OverHeight(x);

OverHeight(y);

return y;

}

Tree\* LeftRotation(Tree\* y) {

Tree\* x = y->right;

y->right = x->left;

x->left = y;

OverHeight(y);

OverHeight(x);

return x;

}

Tree\* Balance(Tree\* root) {

OverHeight(root);

if (BF(root) == 2) {

if (BF(root->right) < 0) root->right = RightRotation(root->right);

return LeftRotation(root);

}

if (BF(root) == -2) {

if (BF(root->left) > 0) root->left = LeftRotation(root->left);

return RightRotation(root);

}

return root;

}

Tree\* Insert(Tree\* root, int data) {

if (!root) return new Tree(data);

if (data < root->key) root->left = Insert(root->left, data);

else root->right = Insert(root->right, data);

return Balance(root);

}

Tree\* SearchMin(Tree\* root) {

if (root->left) return SearchMin(root->left);

else return root;

}

Tree\* DeleteMin(Tree\* root) {

if (root->left == 0) return root->right;

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

return Balance(root);

}

Tree\* Delete(Tree\* root, int data) {

if (!root) return 0;

if (data < root->key) root->left = Delete(root->left, data);

else if (data > root->key) root->right = Delete(root->right, data);

else {

Tree\* y = root->left;

Tree\* z = root->right;

delete root;

if (!z) return y;

Tree\* min = SearchMin(z);

min->right = DeleteMin(z);

min->left = y;

return Balance(min);

}

return Balance(root);

}

Tree\* getTreeByValue(Tree\* root, int value) {

while (root) {

if (root->key > value) {

root = root->left;

continue;

}

else if (root->key < value) {

root = root->right;

continue;

}

else {

return root;

}

}

return NULL;

}

void sizeTree(Tree\* root) {

if (root) {

size\_count++;

sizeTree(root->left);

sizeTree(root->right);

}

}

void widthValue(int step, string object) {

for (int i = 0; i <= step - 1; i++) {

cout << " ";

}

if (object == "right")

cout << ".--->";

else if (object == "left")

cout << "`--->";

else

cout << " -->";

}

void widthValue\_file(int step, string object, ofstream& fout) {

for (int i = 0; i <= step - 1; i++) {

fout << " ";

}

if (object == "right")

fout << ".--->";

else if (object == "left")

fout << "`--->";

else

fout << " -->";

}

void printColorData(int current\_data, int data) {

if (current\_data == data) {

SetColor(2, 0);

cout << current\_data << endl;

SetColor(7, 0);

}

else

cout << current\_data << endl;

}

void printTree\_value(Tree\* root, int step, string object, int value) {

if (root->right) {

printTree\_value(root->right, step + 1, "right", value);

widthValue(step, object);

printColorData(root->key, value);

}

if (!root->right && root->left) {

widthValue(step, object);

printColorData(root->key, value);

printTree\_value(root->left, step + 1, "left", value);

return;

}

if (root->left) {

printTree\_value(root->left, step + 1, "left", value);

return;

}

if (root->right && !root->left)

return;

widthValue(step, object);

printColorData(root->key, value);

}

void printTree(Tree\* root, int step, string object) {

if (root->right) {

printTree(root->right, step + 1, "right");

widthValue(step, object);

cout << root->key << endl;

}

if (!root->right && root->left) {

widthValue(step, object);

cout << root->key << endl;

printTree(root->left, step + 1, "left");

return;

}

if (root->left) {

printTree(root->left, step + 1, "left");

return;

}

if (root->right && !root->left)

return;

widthValue(step, object);

cout << root->key << endl;

}

void printTree\_file(Tree\* root, int step, string object, ofstream& fout) {

if (root->right) {

printTree\_file(root->right, step + 1, "right", fout);

widthValue\_file(step, object, fout);

fout << root->key << endl;

}

if (!root->right && root->left) {

widthValue\_file(step, object, fout);

fout << root->key << endl;

printTree\_file(root->left, step + 1, "left", fout);

return;

}

if (root->left) {

printTree\_file(root->left, step + 1, "left", fout);

return;

}

if (root->right && !root->left)

return;

widthValue\_file(step, object, fout);

fout << root->key << endl;

}

bool fill\_set(set<int>& root\_set, int value) {

int temp, temp\_value;

temp = root\_set.size();

root\_set.insert(value);

if (temp != root\_set.size())

return TRUE;

else

return FALSE;

}

void fill\_root(vector<int>& arr, Tree\*& root) {

for (int i = 0; i < arr.size(); i++) {

root = Insert(root, arr[i]);

}

}

void fill\_set\_via\_Tree(Tree\*& root, set<int>& arr) {

if (root) {

arr.insert(root->key);

fill\_set\_via\_Tree(root->right, arr);

fill\_set\_via\_Tree(root->left, arr);

}

}

void fill\_vec\_via\_Tree(Tree\*& root, vector<int>& arr) {

if (root) {

arr.push\_back(root->key);

fill\_vec\_via\_Tree(root->right, arr);

fill\_vec\_via\_Tree(root->left, arr);

}

}

Tree\* getTreeNoRepeatValues(Tree\*& root) {

Tree\* tmp = NULL;

set <int> TreeVal;

vector <int> numbers\_unic;

vector<int> CopyTree;

fill\_vec\_via\_Tree(root, CopyTree);

vector <int> RepeatValues;

for (int i = 0; i < CopyTree.size(); i++) {

int temp = TreeVal.size();

int temp\_value = CopyTree[i];

TreeVal.insert(temp\_value);

if (temp != TreeVal.size())

numbers\_unic.push\_back(temp\_value);

else

RepeatValues.push\_back(temp\_value);

}

if (CopyTree.size() != numbers\_unic.size()) {

cout << "Дерево имеет повторяющиеся узлы. Эти узлы были удалены" << endl;

cout << "Повторяющиеся элементы: "; for (const auto& token : RepeatValues) { cout << token << " "; } cout << endl;

fill\_root(numbers\_unic, tmp);

return tmp;

}

fill\_root(numbers\_unic, tmp);

return tmp;

}

void individual\_del(Tree\*& root, vector<int>& neg\_values) {

if (root) {

if (root->key < 0) {

neg\_values.push\_back(root->key);

}

individual\_del(root->right, neg\_values);

individual\_del(root->left, neg\_values);

}

}

void del\_values(Tree\*& root, vector<int>& neg\_values) {

while (neg\_values.size()) {

int temp = neg\_values[neg\_values.size() - 1];

neg\_values.pop\_back();

root = Delete(root, temp);

}

}

void individual\_ex(Tree\*& root, int level) {

if (root) {

if (level % 2 != 0) { // умножаем на число в диапазонет от -2 до 2

int temp = (rand() % 5) - 2;

cout << "Текущее число " << root->key << " будет умножено на " << temp << endl;

root->key \*= temp;

}

individual\_ex(root->left, level + 1);

individual\_ex(root->right, level + 1);

}

}

void individual(Tree\*& root) {

system("cls");

chrono::steady\_clock sc;

vector<int> neg\_values;

cout << setw(59) << right << "Индивидуальное задание" << endl;

cout << setw(65) << right << "Удалить все отрицательные числа." << endl;

cout << setw(90) << right << "Каждый нечетный элемент умножить на случайную величину в диапазоне от - 2 до 2. " << endl;

SetColor(6, 0);

cout << setw(65) << right << "Нажмите любую клавишу, чтобы начать.";

SetColor(7, 0);

\_getch();

cout << endl;

system("cls");

cout << setw(30) << right << "АВЛ дерево:" << endl;

printTree(root, 0, " "); cout << endl;

auto start = sc.now();

individual\_del(root, neg\_values);

del\_values(root, neg\_values);

individual\_ex(root, 0);

root = getTreeNoRepeatValues(root);

cout << setw(30) << right << "Измененное АВЛ дерево:" << endl;

printTree(root, 0, " "); cout << endl;

auto end = sc.now();

auto time\_span = static\_cast<chrono::duration<double>>(end - start);

cout << "Скорость работы АВЛ дерева равна: " << time\_span.count() << " секунд" << endl;

cout << "Преобразование успешно завершено. Нажмите любую клавишу" << endl;

\_getch();

menu(root);

}

void delete\_node(Tree\*& root) {

system("cls");

int value;

int prev\_count;

chrono::steady\_clock sc;

size\_count = 0;

sizeTree(root);

prev\_count = size\_count;

size\_count = 0;

cout << setw(30) << right << "АВЛ дерево:" << endl;

printTree(root, 0, " ");

cout << "Введите элемент, который необходимо удалить: ";

value = checkdigit();

auto start = sc.now();

root = Delete(root, value);

sizeTree(root);

if (size\_count == prev\_count) {

cout << "Значение не найдено. Нажмите любую клавишу, чтобы продолжить" << endl;

\_getch();

menu(root);

}

auto end = sc.now();

auto time\_span = static\_cast<chrono::duration<double>>(end - start); // высчитываем время, затраченное на событие

cout << "Скорость удаления узла равна: " << time\_span.count() << " секунд" << endl;

cout << "Узел успешно удален. Нажмите любую клавишу" << endl;

if (!root) {

SetColor(4, 0);

cout << "Дерево было полностью удалено." << endl;

SetColor(7, 0);

fl\_tree = FALSE;

}

\_getch();

menu(root);

}

void insert\_node(Tree\*& root) {

system("cls");

int value;

chrono::steady\_clock sc;

set<int> root\_set;

double time = 0;

fill\_set\_via\_Tree(root, root\_set);

if (root) {

cout << setw(30) << right << "АВЛ дерево:" << endl;

printTree(root, 0, " ");

}

cout << "Введите значение элемента: ";

value = checkdigit();

if (fill\_set(root\_set, value)) {

auto start = sc.now();

root = Insert(root, value);

auto end = sc.now();

auto time\_span = static\_cast<chrono::duration<double>>(end - start);

time += time\_span.count();

}

else {

SetColor(4, 0);

cout << "Значение записано не будет. Такое значение уже есть в АВЛ дереве" << endl;

SetColor(7, 0);

cout << "Нажмите любую клавишу, чтобы продолжить" << endl;

\_getch();

menu(root);

}

cout << "Скорость вставки узла равна: " << time << " секунд" << endl;

cout << "Узел успешно вставлен. Нажмите любую клавишу" << endl;

\_getch();

fl\_tree = TRUE;

menu(root);

}

void find\_node(Tree\*& root) {

system("cls");

int value;

chrono::steady\_clock sc;

Tree\* findTree = NULL;

cout << setw(30) << right << "АВЛ дерево:" << endl;

printTree(root, 0, " ");

cout << "Введите элемент, который необходимо найти: ";

value = checkdigit();

auto start = sc.now();

findTree = getTreeByValue(root, value);

if (findTree == NULL) {

cout << "Значение не найдено. Нажмите любую клавишу, чтобы продолжить" << endl;

\_getch();

menu(root);

}

else {

cout << setw(30) << right << "АВЛ дерево и значение " << findTree->key << ":" << endl;

printTree\_value(root, 0, " ", findTree->key);

}

auto end = sc.now();

auto time\_span = static\_cast<chrono::duration<double>>(end - start); // высчитываем время, затраченное на событие

cout << "Скорость поиска узла равна: " << time\_span.count() << " секунд" << endl;

cout << "Узел успешно найден. Нажмите любую клавишу" << endl;

\_getch();

menu(root);

}

void fill\_root\_random(Tree\*& root) {

system("cls"); // очищаем консоль

root = NULL;

int size = 0;

int index = 0;

chrono::steady\_clock sc;

set<int> root\_set;

cout << "Введите количество элементов в дереве: ";

size = checkdigit();

if (size <= 0) {

cout << "Количество элементов дерева не может быть отрицательным или нулевым. Нажмите любую клавишу, чтобы продолжить.";

\_getch();

menu(root);

}

auto start = sc.now();

while (index <= size - 1) {

int temp = (rand() % 200) - 100;

if (fill\_set(root\_set, temp)) {

root = Insert(root, temp);

index++;

}

}

auto end = sc.now();

auto time\_span = static\_cast<chrono::duration<double>>(end - start); // высчитываем время, затраченное на событие

cout << "Скорость создания АВЛ дерева равна: " << time\_span.count() << " секунд" << endl;

cout << "АВЛ Дерево успешно создано. Нажмите любую клавишу" << endl;

\_getch();

fl\_tree = TRUE;

menu(root);

}

void fill\_root\_self(Tree\*& root) {

system("cls"); // очищаем консоль

root = NULL;

int temp;

int index = 0;

chrono::steady\_clock sc;

bool fl\_stop = TRUE;

set <int> root\_set;

double summ\_time = 0;

cout << "Вводите числовые значения. Если хотите закончить, то введите любой нечисловой символ" << endl;

while (fl\_stop) {

cout << index << ": ";

cin >> temp; // число целое

if (cin.fail()) {

cin.clear(); // то возвращаем cin в обычный режим работы

cin.ignore(32767, '\n');

fl\_stop = FALSE;

}

else {

cin.ignore(32767, '\n');

if (fill\_set(root\_set, temp)) {

auto start = sc.now();

root = Insert(root, temp);

auto end = sc.now();

auto time\_span = static\_cast<chrono::duration<double>>(end - start);

summ\_time += time\_span.count();

index++;

}

else {

SetColor(4, 0);

cout << "Значение записано не будет. Такое значение уже есть в АВЛ дереве" << endl;

SetColor(7, 0);

}

}

}

cout << "Скорость создания АВЛ дерева равна: " << summ\_time << " секунд" << endl;

cout << "АВЛ Дерево успешно создано. Нажмите любую клавишу" << endl;

\_getch();

fl\_tree = TRUE;

menu(root);

}

void fill\_root\_from\_file(Tree\*& root) {

system("cls"); // очищаем консоль

root = NULL;

ifstream fin;

chrono::steady\_clock sc;

set <int> root\_set;

vector<int> RepeatValues;

fin.open("file\_root.txt");

auto start = sc.now();

if (fin.is\_open()) {

int temp = 0;

while (!fin.eof()) {

fin >> temp;

if (fin.fail()) {

fin.clear();

fin.ignore(32767, '\n');

cout << "Файл содержит некорректные числовые данные. Необходимо закрыть программу и ввести в текстовый файл данные корректно (между числовыми значениями пробел, посторонних знаков быть не должно)." << endl;

cout << "Пример: 50 -3 45 14 0 4 10 1" << endl;

cout << "Нажмите, чтобы закрыть программу" << endl;

\_getch();

exit(3);

}

else {

if (fill\_set(root\_set, temp)) {

root = Insert(root, temp);

}

else

RepeatValues.push\_back(temp);

}

}

}

else {

cout << "Файл открыть не удалось! Закройте программу и создайте в корневой папке программы файл file\_root.txt для заполнения дерева через файл." << endl;

cout << "Нажмите, чтобы закрыть программу" << endl;

\_getch();

exit(3);

}

auto end = sc.now(); // устанавливаем конец отсчета времени события

auto time\_span = static\_cast<chrono::duration<double>>(end - start); // высчитываем время, затраченное на событие

fin.close();

if (RepeatValues.size() != 0) {

cout << "Повторяющиеся элементы: "; for (const auto& token : RepeatValues) { cout << token << " "; } cout << endl;

}

cout << "Скорость создания дерева равна: " << time\_span.count() << " секунд" << endl;

cout << "АВЛ Дерево успешно создано. Нажмите любую клавишу" << endl;

\_getch();

fl\_tree = TRUE;

menu(root);

}

void correctBalance(Tree\*& root) {

system("cls");

chrono::steady\_clock sc;

cout << setw(30) << right << "АВЛ дерево:" << endl;

printTree(root, 0, " ");

cout << endl;

auto start = sc.now();

root = Balance(root);

auto end = sc.now(); // устанавливаем конец отсчета времени события

auto time\_span = static\_cast<chrono::duration<double>>(end - start); // высчитываем время, затраченное на событие

SetColor(2, 0);

cout << "Дерево сбалансировано." << endl;

SetColor(7, 0);

cout << "Скорость проверки на сбалансированность равна: " << time\_span.count() << " секунд" << endl;

cout << "Нажмите любую клавишу" << endl;

\_getch();

menu(root);

}

///////////////// MENU ////////////////////

void print\_alarm(Tree\*& root) {

SetColor(4, 0);

cout << endl << "Сперва необходимо создать дерево. Нажмите любую клавишу";

SetColor(7, 0);

\_getch();

submenu\_create\_tree(root);

}

void subconf\_crT\_val(Tree\*& root) {

switch (subm\_crT\_count) {

case 0:

fill\_root\_random(root);

break;

case 1:

fill\_root\_self(root);

break;

case 2:

fill\_root\_from\_file(root);

break;

case 3:

menu(root);

break;

default:

break;

}

}

void conf\_val(Tree\*& root) {

switch (m\_count) {

case 0:

submenu\_create\_tree(root);

break;

case 1:

insert\_node(root);

break;

case 2:

if (fl\_tree)

delete\_node(root);

else

print\_alarm(root);

break;

case 3:

if (fl\_tree)

find\_node(root);

else

print\_alarm(root);

break;

case 4:

if (fl\_tree) {

ofstream fout;

fout.open("AVL\_tree.txt", ios::trunc);

system("cls");

cout << setw(30) << right << "АВЛ дерево:" << endl;

fout << setw(30) << right << "АВЛ дерево:" << endl;

printTree(root, 0, " ");

printTree\_file(root, 0, " ", fout);

cout << "Дерево было сохранено в текстовом файле AVL\_tree.txt" << endl;

cout << "Нажмите любую клавишу, чтобы вернуться в меню.";

\_getch();

fout.close();

}

else

print\_alarm(root);

break;

case 5:

if (fl\_tree)

correctBalance(root);

else

print\_alarm(root);

break;

case 6:

if (fl\_tree)

individual(root);

else

print\_alarm(root);

break;

case 7:

SetColor(0, 0);

exit(0);

break;

default:

break;

}

menu(root);

}

void submenu\_create\_tree\_choice(Tree\*& root) {

int k1;

k1 = \_getch(); // получаем символ стрелки без вывода знака

if (k1 == 0xE0) { // если стрелки

switch (k1) {

case 0x48: // стрелка вверх

subm\_crT\_count--;

if (subm\_crT\_count < 0) subm\_crT\_count = 0;

submenu\_create\_tree(root);

break;

case 0x50: // стрелка вниз

subm\_crT\_count++;

if (subm\_crT\_count > 3) subm\_crT\_count = 3;

submenu\_create\_tree(root);

break;

case 0xD: // подтвердить

subconf\_crT\_val(root);

break;

default:

submenu\_create\_tree\_choice(root);

}

}

switch (k1) {

case 0x48: // стрелка вверх

subm\_crT\_count--;

if (subm\_crT\_count < 0) subm\_crT\_count = 0;

submenu\_create\_tree(root);

break;

case 0x50: // стрелка вниз

subm\_crT\_count++;

if (subm\_crT\_count > 3) subm\_crT\_count = 3;

submenu\_create\_tree(root);

break;

case 0xD: // подтвердить

subconf\_crT\_val(root);

break;

default:

submenu\_create\_tree\_choice(root);

}

}

void menu\_choice(Tree\*& root) {

int k1;

k1 = \_getch(); // получаем символ стрелки без вывода знака

if (k1 == 0xE0) { // если стрелки

switch (k1) {

case 0x48: // стрелка вверх

m\_count--;

if (m\_count < 0) m\_count = 0;

menu(root);

break;

case 0x50: // стрелка вниз

m\_count++;

if (m\_count > 7) m\_count = 7;

menu(root);

break;

case 0xD: // подтвердить

conf\_val(root);

break;

default:

menu\_choice(root);

}

}

switch (k1) {

case 0x48: // стрелка вверх

m\_count--;

if (m\_count < 0) m\_count = 0;

menu(root);

break;

case 0x50: // стрелка вниз

m\_count++;

if (m\_count > 7) m\_count = 7;

menu(root);

break;

case 0xD: // подтвердить

conf\_val(root);

break;

default:

menu\_choice(root);

}

}

void submenu\_create\_tree(Tree\*& root) {

system("cls"); // очищаем консоль

if (subm\_crT\_count == 0) {

cout << right << "\_\_СОЗДАНИЕ ДЕРЕВА\_\_" << endl << endl;

SetColor(3, 0);

cout << "Каким образом создаем АВЛ дерево?" << endl << endl;

SetColor(7, 0);

SetColor(5, 0);

cout << "Автоматически, с указаной размерностью" << endl;

SetColor(7, 0);

cout << "Вручную, заполняя значения" << endl;

cout << "Заполнение с помощью текстового файла" << endl;

cout << "Назад" << endl;

submenu\_create\_tree\_choice(root);

}

if (subm\_crT\_count == 1) {

cout << right << "\_\_СОЗДАНИЕ ДЕРЕВА\_\_" << endl << endl;

SetColor(3, 0);

cout << "Каким образом создаем АВЛ дерево?" << endl << endl;

SetColor(7, 0);

cout << "Автоматически, с указаной размерностью" << endl;

SetColor(5, 0);

cout << "Вручную, заполняя значения" << endl;

SetColor(7, 0);

cout << "Заполнение с помощью текстового файла" << endl;

cout << "Назад" << endl;

submenu\_create\_tree\_choice(root);

}

if (subm\_crT\_count == 2) {

cout << right << "\_\_СОЗДАНИЕ ДЕРЕВА\_\_" << endl << endl;

SetColor(3, 0);

cout << "Каким образом создаем АВЛ дерево?" << endl << endl;

SetColor(7, 0);

cout << "Автоматически, с указаной размерностью" << endl;

cout << "Вручную, заполняя значения" << endl;

SetColor(5, 0);

cout << "Заполнение с помощью текстового файла" << endl;

SetColor(7, 0);

cout << "Назад" << endl;

submenu\_create\_tree\_choice(root);

}

if (subm\_crT\_count == 3) {

cout << right << "\_\_СОЗДАНИЕ ДЕРЕВА\_\_" << endl << endl;

SetColor(3, 0);

cout << "Каким образом создаем АВЛ дерево?" << endl << endl;

SetColor(7, 0);

cout << "Автоматически, с указаной размерностью" << endl;

cout << "Вручную, заполняя значения" << endl;

cout << "Заполнение с помощью текстового файла" << endl;

SetColor(5, 0);

cout << "Назад" << endl;

SetColor(7, 0);

submenu\_create\_tree\_choice(root);

}

}

void menu(Tree\*& root) {

system("cls"); // очищаем консоль

if (m\_count == 0) {

cout << setw(15) << right << "\_\_МЕНЮ\_\_" << endl;

SetColor(5, 0);

cout << "Создать АВЛ дерево" << endl;

SetColor(7, 0);

cout << "Добавить элемент к АВЛ дереву" << endl;

cout << "Удалить элемент АВД дерева" << endl;

cout << "Получить элемент АВЛ дерева" << endl;

cout << "Вывести АВЛ дерево" << endl;

cout << "Проверка на сбалансированность" << endl;

cout << "Задание по варианту" << endl;

cout << "Выход" << endl;

menu\_choice(root);

}

if (m\_count == 1) {

cout << setw(15) << right << "\_\_МЕНЮ\_\_" << endl;

cout << "Создать АВЛ дерево" << endl;

SetColor(5, 0);

cout << "Добавить элемент к АВЛ дереву" << endl;

SetColor(7, 0);

cout << "Удалить элемент АВД дерева" << endl;

cout << "Получить элемент АВЛ дерева" << endl;

cout << "Вывести АВЛ дерево" << endl;

cout << "Проверка на сбалансированность" << endl;

cout << "Задание по варианту" << endl;

cout << "Выход" << endl;

menu\_choice(root);

}

if (m\_count == 2) {

cout << setw(15) << right << "\_\_МЕНЮ\_\_" << endl;

cout << "Создать АВЛ дерево" << endl;

cout << "Добавить элемент к АВЛ дереву" << endl;

SetColor(5, 0);

cout << "Удалить элемент АВД дерева" << endl;

SetColor(7, 0);

cout << "Получить элемент АВЛ дерева" << endl;

cout << "Вывести АВЛ дерево" << endl;

cout << "Проверка на сбалансированность" << endl;

cout << "Задание по варианту" << endl;

cout << "Выход" << endl;

menu\_choice(root);

}

if (m\_count == 3) {

cout << setw(15) << right << "\_\_МЕНЮ\_\_" << endl;

cout << "Создать АВЛ дерево" << endl;

cout << "Добавить элемент к АВЛ дереву" << endl;

cout << "Удалить элемент АВД дерева" << endl;

SetColor(5, 0);

cout << "Получить элемент АВЛ дерева" << endl;

SetColor(7, 0);

cout << "Вывести АВЛ дерево" << endl;

cout << "Проверка на сбалансированность" << endl;

cout << "Задание по варианту" << endl;

cout << "Выход" << endl;

menu\_choice(root);

}

if (m\_count == 4) {

cout << setw(15) << right << "\_\_МЕНЮ\_\_" << endl;

cout << "Создать АВЛ дерево" << endl;

cout << "Добавить элемент к АВЛ дереву" << endl;

cout << "Удалить элемент АВД дерева" << endl;

cout << "Получить элемент АВЛ дерева" << endl;

SetColor(5, 0);

cout << "Вывести АВЛ дерево" << endl;

SetColor(7, 0);

cout << "Проверка на сбалансированность" << endl;

cout << "Задание по варианту" << endl;

cout << "Выход" << endl;

menu\_choice(root);

}

if (m\_count == 5) {

cout << setw(15) << right << "\_\_МЕНЮ\_\_" << endl;

cout << "Создать АВЛ дерево" << endl;

cout << "Добавить элемент к АВЛ дереву" << endl;

cout << "Удалить элемент АВД дерева" << endl;

cout << "Получить элемент АВЛ дерева" << endl;

cout << "Вывести АВЛ дерево" << endl;

SetColor(5, 0);

cout << "Проверка на сбалансированность" << endl;

SetColor(7, 0);

cout << "Задание по варианту" << endl;

cout << "Выход" << endl;

menu\_choice(root);

}

if (m\_count == 6) {

cout << setw(15) << right << "\_\_МЕНЮ\_\_" << endl;

cout << "Создать АВЛ дерево" << endl;

cout << "Добавить элемент к АВЛ дереву" << endl;

cout << "Удалить элемент АВД дерева" << endl;

cout << "Получить элемент АВЛ дерева" << endl;

cout << "Вывести АВЛ дерево" << endl;

cout << "Проверка на сбалансированность" << endl;

SetColor(5, 0);

cout << "Задание по варианту" << endl;

SetColor(7, 0);

cout << "Выход" << endl;

menu\_choice(root);

}

if (m\_count == 7) {

cout << setw(15) << right << "\_\_МЕНЮ\_\_" << endl;

cout << "Создать АВЛ дерево" << endl;

cout << "Добавить элемент к АВЛ дереву" << endl;

cout << "Удалить элемент АВД дерева" << endl;

cout << "Получить элемент АВЛ дерева" << endl;

cout << "Вывести АВЛ дерево" << endl;

cout << "Проверка на сбалансированность" << endl;

cout << "Задание по варианту" << endl;

SetColor(5, 0);

cout << "Выход" << endl;

SetColor(7, 0);

menu\_choice(root);

}

}

///////////////// MENU ////////////////////

int main() {

setlocale(0, "");

srand(time(NULL));

Tree\* root = NULL;

menu(root);

system("pause");

return 0;

}