**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №4**

**по дисциплине «Программирование»**

Тема: Бинарное дерево поиска

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

Санкт-Петербург

2020

# **Цель работы.**

Изучение свойств и организация деревьев как структуры данных. Получение практических навыков в работе с бинарным деревом поиска. Определение преимуществ и недостатков структуры данных вида дерева. Проведение сравнительной характеристики скорости вставки, удаление и поиска элемента в различных структурах данных.

# **Основные теоретические положения.**

Бинарное дерево – это динамическая структура данных, состоящая из узлов, каждый из которых содержит, кроме данных, не более двух ссылок на различные бинарные деревья [1]. Начальный узел называется корнем дерева. Графическое представление бинарного дерева представлено на рисунке 1.

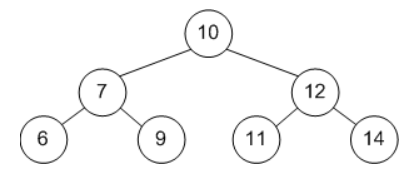


Рисунок 1 – Графическое представление бинарного дерева

Основные операции над стеками:

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

При каждой операции вставки нового или удаления существующего узла отсортированный порядок дерева сохраняется. При поиске элемента сравнивается искомое значение с корнем. Если искомое значение больше корня, то поиск продолжается в правом потомке корня, если меньше, то в левом, если равно, то значение найдено и поиск прекращается.

Дерево является рекурсивной структурой данных, поскольку каждое поддререво также является деревом. Действия с такими структурами лучше всего описывать с помощью рекурсивных алгоритмов.

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

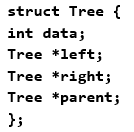


Рисунок 2 – Общий вид структуры данных дерева

Так как одинаковые данные не допускаются, перед вставкой нового элемента необходимо проверить, не ли уже в бинарном дереве такого элемента.

Удаление узла реализуется в зависимости от ситуации:

1. У удаляемого узла нет наследников (узел является листом).
2. У удаляемого узла имеется наследник, при том только один.
3. У удаляемого узла есть оба наследника.

Если у узла нет наследника, то необходимо просто удалить этот узел, а у родителя обнулить указатель на него.

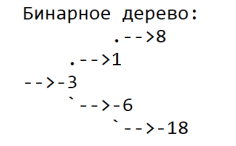
Если у узла имеется лишь один наследник, то необходимо его подменить своим наследником.

Если у узла два наследника, то узел не удаляется, а заменяется егго значение на максимум левого поддерева. После чего удаляется максимум левого поддерева.

# **Постановка задачи.**

Необходимо реализовать программу, которая выполняет следующие действия:

1. Формирование бинарного 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. Сравнение скорости работы бинарного дерева с линейной структурой (двусвязным списком или динамическим массивом).
2. Прямой обход, обратный обход и обход в ширину бинарного дерева.

Также необходимо реализовать индивидуальное задание – уменьшение каждого нечетного элемента на значение, введенное пользователем. Каждый четный элемент необходимо умножить на случайную величину в диапазоне от 1 до 5.

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 обхода дерева и для способа ввода.

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().

Бинарное 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 бинарнго 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 deleteNode(). На вход функция принимает дерево и значение, которое необходимо удалить.

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.

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.

Функции обхода дерева реализованы на основе рекурсивных алгоритмов: в функции preOrderTravers() реализован прямой обход дерева, в функции postOrderTravers() – обратный обход дерева, в функции breadthFirst() – обход в ширину дерева.

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.

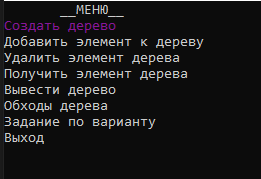


Рисунок 3 – Скриншот меню программы

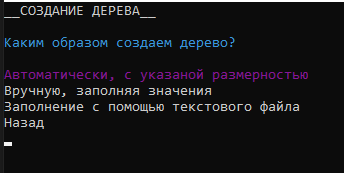


Рисунок 4 – Скриншот подменю создания дерева

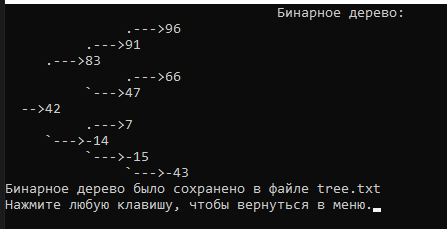


Рисунок 5 – Скриншот выбора пункта «Вывести дерево»

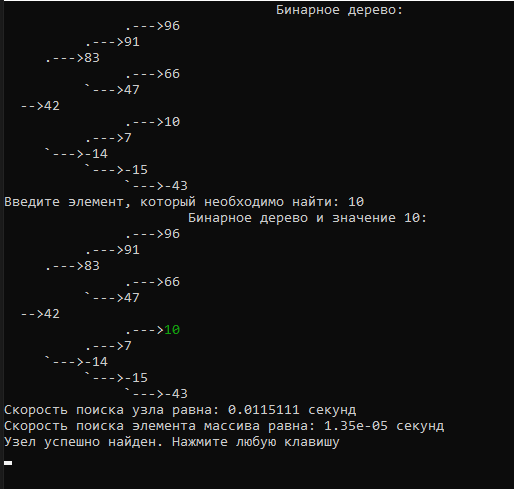


Рисунок 6 – Скриншот поиска элемента в бинарном дереве

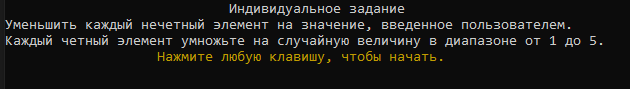


Рисунок 7 – Скриншот пояснения к индвидуальному заданию

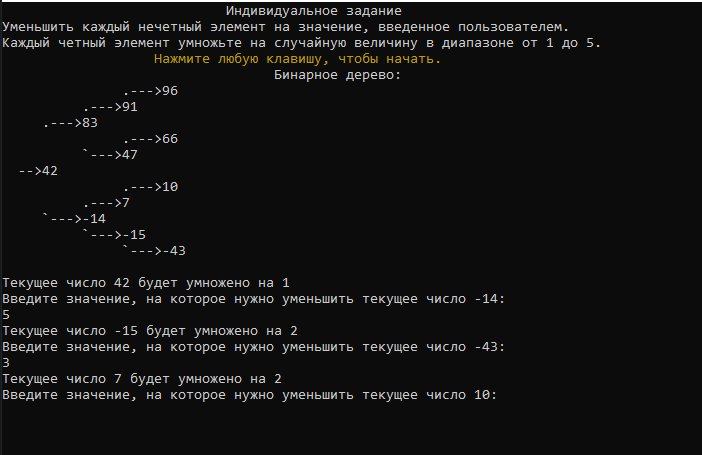


Рисунок 8 – Скриншот выполнения индвидиуального задания

Также было проведено сравнение скорости бинарного дерева и динамического массива при одинаковой размерности. Поиск в массиве был реализован с помощью бинарного поиска. Заполнение значений выполнялось автоматически. Сравнение происходило в поиске, вставки и удаления элемента (см. табл. 1).

Таблица 1 – Сравнение скорости бинарного дерева и динамического массива

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Количество элементов | Бинарное дерево | | | Динамический массив | | |
| Вставка, сек | Удаление, сек | Поиск, сек | Вставка, сек | Удаление, сек | Поиск, сек |
| 50 | 1,8E-06 | 7,7E-06 | 1,9E-06 | 1,2E-06 | 0,0003 | 6E-06 |
| 100 | 1,9E-06 | 1,27E-05 | 2,3E-06 | 1,6E-06 | 0,0001 | 1,05-05 |
| 500 | 2E-06 | 5,42E-05 | 1,7E-06 | 1,5E-06 | 0.0215 | 4,67E-05 |
| 1 000 | 1,7E-06 | 0,0001 | 2,3E-06 | 1,6E-06 | 0,0874 | 9,18E-05 |
| 5 000 | 2,8E-06 | 0,0007 | 3,4E-06 | 1,3E-06 | 2,189 | 0,0005 |
| 10 000 | 2,7E-06 | 0,0013 | 2,8E-06 | 1,5E-06 | 6,6237 | 0,001 |

Из представленных графиков (см. рис. 9 – 11) видно, динамический массив сильно уступает бинарному дереву в удалении элемента и в поиске значения узла. Можно предположить, что это связано с несоврешенством реализации бинарного поиска для массива, а также существенной простотой рекурсивных алгоритмов, что ускоряет обработку.

Рисунок 9 – График зависимости скорости от количества элементов во время вставки элемента

Рисунок 10 – График зависимости скорости от количества элеметов во время удаления элемента

Рисунок 11 – График зависимости скорости от количества элементов во время поиска элемента

# **Выводы.**

Были освоены теоретические знания и закреплены практические навыки по работе с бинарным деревом поиска. Было проведено исследование по практическому применению бинарных деревьев. Была провередена сравнительная характеристика скорости вставки, удаления и поиска элемента в различных структурах данных. Приведённые графики показывают, что использование бинарных деревьев для обработки числовых данных сущетсвенно ускоряет работу, 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 № 4 - Бинарное дерево поиска

#include <iostream>

#include <conio.h>

#include <set>

#include <ctime>

#include <vector>

#include <string>

#include <Windows.h>

#include <chrono>

#include <fstream>

#include <iomanip>

using namespace std;

int m\_count = 0;

int subm\_crT\_count = 0;

int subm\_pT\_count = 0;

int size\_count = 0;

bool fl\_tree = FALSE;

void SetColor(int text, int bg) //Функция смены цвета, взятая из Интернета

{

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

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

}

typedef struct Node {

int data;

struct Node\* left;

struct Node\* right;

struct Node\* parent;

} Tree;

typedef struct \_Nodes {

void\* value;

struct \_Nodes\* next;

struct \_Nodes\* prev;

} \_Node;

typedef struct \_DblLinkedList {

size\_t size;

\_Node\* head;

\_Node\* tail;

} DblLinkedList;

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

void menu(Tree\*& root, vector<int>& root\_vec);

void submenu\_create\_tree(Tree\*& root, vector<int>& root\_vec);

void submenu\_pass\_tree(Tree\*& root, vector<int>& root\_vec);

void menu\_choice(Tree\*& root, vector<int>& root\_vec);

void submenu\_create\_tree\_choice(Tree\*& root, vector<int>& root\_vec);

void submenu\_pass\_tree\_choice(Tree\*& root, vector<int>& root\_vec);

void conf\_val(Tree\*& root, vector<int>& root\_vec);

void subconf\_crT\_val(Tree\*& root, vector<int>& root\_vec);

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

///////////////////////// CHECK ////////////////////

int checkdigit() // проверка на корректный ввод

{

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'); // и удаляем из буфера значения предыдущего ввода

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

}

else

{

cin.ignore(32767, '\n'); // удаляем из буфера значения предыдущего ввода

return value;

}

}

}

void gnomeSort(vector<int>& c) // при нахождении неотсортированной пары делается шаг назад, отсортированной - шаг вперед

{

int index = 0;

while (index < c.size()) {

if (index == 0)

index++;

if (c[index] >= c[index - 1])

index++;

else {

swap(c[index], c[index - 1]);

index--;

}

}

return;

}

int Search\_Binary(vector<int>& digits\_array, int left, int right, int key) {

if(digits\_array.size() != 1)

gnomeSort(digits\_array);

bool counter = 1;

int midd = 0;

while (counter == 1) {

midd = (left + right) / 2;

if (key < digits\_array[midd])

right = midd - 1;

else if (key > digits\_array[midd])

left = midd + 1;

else

return midd;

if (left > right)

counter = 0;

}

return -1;

}

///////////////////////// CHECK ////////////////////

///////////////////////// QUEUE ////////////////////

DblLinkedList\* createDblLinkedList() {

DblLinkedList\* tmp = (DblLinkedList\*)malloc(sizeof(DblLinkedList));

tmp->size = 0;

tmp->head = tmp->tail = NULL;

return tmp;

}

void deleteDblLinkedList(DblLinkedList\*\* list) {

\_Node\* tmp = (\*list)->head;

\_Node\* next = NULL;

while (tmp) {

next = tmp->next;

free(tmp);

tmp = next;

}

free(\*list);

(\*list) = NULL;

}

void\* popFront(DblLinkedList\* list) {

\_Node\* prev;

void\* tmp;

if (list->head == NULL) {

exit(2);

}

prev = list->head;

list->head = list->head->next;

if (list->head) {

list->head->prev = NULL;

}

if (prev == list->tail) {

list->tail = NULL;

}

tmp = prev->value;

free(prev);

list->size--;

return tmp;

}

void pushBack(DblLinkedList\* list, void\* value) {

\_Node\* tmp = (\_Node\*)malloc(sizeof(\_Node));

if (tmp == NULL) {

exit(3);

}

tmp->value = value;

tmp->next = NULL;

tmp->prev = list->tail;

if (list->tail) {

list->tail->next = tmp;

}

list->tail = tmp;

if (list->head == NULL) {

list->head = tmp;

}

list->size++;

}

///////////////////////// QUEUE ////////////////////

///////////////////////// TREE /////////////////////

Tree\* getFreeTree(int value, Tree\* parent) {

Tree\* tmp = (Tree\*)malloc(sizeof(Tree));

tmp->left = tmp->right = NULL;

tmp->data = value;

tmp->parent = parent;

return tmp;

}

void insert(Tree\*\* head, int value) {

Tree\* tmp = NULL;

Tree\* ins = NULL;

if (\*head == NULL) {

\*head = getFreeTree(value, NULL);

return;

}

tmp = \*head;

while (tmp) {

if (value > tmp->data) {

if (tmp->right) {

tmp = tmp->right;

continue;

}

else {

tmp->right = getFreeTree(value, tmp);

return;

}

}

else if (value < tmp->data) {

if (tmp->left) {

tmp = tmp->left;

continue;

}

else {

tmp->left = getFreeTree(value, tmp);

return;

}

}

else {

cout << value << " EQUAL " << tmp->data << endl;

exit(2);

}

}

}

Tree\* getMinTree(Tree\* root) {

while (root->left) {

root = root->left;

}

return root;

}

Tree\* getMaxTree(Tree\* root) {

while (root->right) {

root = root->right;

}

return root;

}

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

while (root) {

if (root->data > value) {

root = root->left;

continue;

}

else if (root->data < value) {

root = root->right;

continue;

}

else {

return root;

}

}

return NULL;

}

void sizeTree(Tree\* root) {

if (root) {

size\_count++;

sizeTree(root->left);

sizeTree(root->right);

}

}

Tree\* deleteNode(Tree\* root, int z) { // корень поддерева, удаляемый ключ

if (root == NULL)

return root;

if (z < root->data)

root->left = deleteNode(root->left, z);

else if (z > root->data)

root->right = deleteNode(root->right, z);

else if (root->left != NULL and root->right != NULL) {

root->data = getMinTree(root->right)->data;

root->right = deleteNode(root->right, root->data);

}

else {

if (root->left != NULL)

root = root->left;

else if (root->right != NULL)

root = root->right;

else

root = NULL;

}

return root;

/////////////////////////////

}

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 << " -->";

}

///////////////////////// TREE /////////////////////

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->data, value);

}

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

widthValue(step, object);

printColorData(root->data, 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->data, value);

}

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

if (root->right) {

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

widthValue(step, object);

cout << root->data << endl;

}

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

widthValue(step, object);

cout << root->data << 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->data << 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->data << endl;

}

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

widthValue\_file(step, object, fout);

fout << root->data << 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->data << endl;

}

void preOrderTravers(Tree\* root) {

if (root) {

cout << root->data << " ";

preOrderTravers(root->left);

preOrderTravers(root->right);

}

}

void postOrderTravers(Tree\* root) {

if (root) {

postOrderTravers(root->left);

postOrderTravers(root->right);

cout << root->data << " ";

}

}

void breadthFirst(Tree\* root) {

DblLinkedList\* q = createDblLinkedList();

//Для начала поместим в очередь корень

pushBack(q, root);

while (q->size != 0) {

Tree\* tmp = (Tree\*)popFront(q);

cout << root->data << " ";

//Если есть левый наследник, то помещаем его в очередь для дальнейшей обработки

if (tmp->left) {

pushBack(q, tmp->left);

}

//Если есть правый наследник, то помещаем его в очередь для дальнейшей обработки

if (tmp->right) {

pushBack(q, tmp->right);

}

}

deleteDblLinkedList(&q);

}

///////////////////////////// FILL /////////////////////

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\_random(Tree\*& root, vector<int>& root\_vec) {

system("cls"); // очищаем консоль

root = NULL;

root\_vec.clear();

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, root\_vec);

}

auto start = sc.now(); // устанавливаем начало отсчета времени события

while (index <= size - 1)

{

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

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

insert(&root, temp);

root\_vec.push\_back(temp);

index++;

}

}

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

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

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

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

\_getch();

fl\_tree = TRUE;

menu(root, root\_vec);

}

void fill\_root\_self(Tree\*& root, vector<int>& root\_vec)

{

system("cls"); // очищаем консоль

root = NULL;

root\_vec.clear();

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()) // ecли предыдущее извелечение оказлось неудачным,

{

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

cin.ignore(32767, '\n'); // и удаляем из буфера значения предыдущего ввода

fl\_stop = FALSE;

}

else

{

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

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

auto start = sc.now();

insert(&root, temp);

auto end = sc.now();

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

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

index++;

root\_vec.push\_back(temp); // для заполнения вектора

}

else {

SetColor(4, 0);

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

SetColor(7, 0);

}

}

}

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

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

\_getch();

fl\_tree = TRUE;

menu(root, root\_vec);

}

void fill\_root\_from\_file(Tree\*& root, vector<int>& root\_vec)

{

system("cls"); // очищаем консоль

root = NULL;

root\_vec.clear();

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)) {

insert(&root, temp);

root\_vec.push\_back(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 << "Дерево успешно создано. Нажмите любую клавишу" << endl;

\_getch();

fl\_tree = TRUE;

menu(root, root\_vec);

}

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

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

insert(&root, arr[i]);

}

}

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

if (root) {

if (level % 2 == 0) { // добавляем от 1 до 5

int temp = (rand() % 5) + 1;

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

root->data \*= temp;

}

else {

int value;

cout << "Введите значение, на которое нужно уменьшить текущее число " << root->data << ": " << endl;

value = checkdigit();

root->data -= value;

}

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

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

}

}

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

if (root) {

arr.insert(root->data);

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->data);

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

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

}

}

///////////////////////////// FILL /////////////////////

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;

}

return root;

}

void delete\_node(Tree\*& root, vector<int>& root\_vec) {

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(50) << right << "Бинарное дерево:" << endl;

printTree(root, 0, " ");

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

value = checkdigit();

auto start = sc.now();

root = deleteNode(root, value);

sizeTree(root);

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

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

\_getch();

menu(root, root\_vec);

}

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

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

auto start1 = sc.now();

int fValue = Search\_Binary(root\_vec, 0, root\_vec.size(), value);

if (fValue == -1) {

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

\_getch();

menu(root, root\_vec);

}

else

root\_vec.erase(root\_vec.begin() + fValue);

auto end1 = sc.now(); // устанавливаем конец отсчета времени события

auto time\_span1 = static\_cast<chrono::duration<double>>(end1 - start1); // высчитываем время, затраченное на событие

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

cout << "Скорость удаления элемента массива равна: " << time\_span1.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, root\_vec);

}

void insert\_node(Tree\*& root, vector<int>& root\_vec) {

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(50) << right << "Бинарное дерево:" << endl;

printTree(root, 0, " ");

}

cout << "Введите значение элемента: ";

value = checkdigit();

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

auto start = sc.now();

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 дереве" << endl;

SetColor(7, 0);

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

\_getch();

menu(root, root\_vec);

}

auto start1 = sc.now();

root\_vec.push\_back(value);

auto end1 = sc.now(); // устанавливаем конец отсчета времени события

auto time\_span1 = static\_cast<chrono::duration<double>>(end1 - start1); // высчитываем время, затраченное на событие

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

cout << "Скорость вставки элемента массива равна: " << time\_span1.count() << " секунд" << endl;

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

\_getch();

menu(root, root\_vec);

}

void find\_node(Tree\*& root, vector<int>& root\_vec) {

system("cls");

int value;

chrono::steady\_clock sc;

Tree\* findTree = NULL;

cout << setw(50) << right << "Бинарное 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, root\_vec);

}

else {

cout << setw(50) << right << "Бинарное дерево и значение " << findTree->data << ":" << endl;

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

}

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

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

auto start1 = sc.now();

int fValue = Search\_Binary(root\_vec, 0, root\_vec.size(), value);

if (fValue == -1) {

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

\_getch();

menu(root, root\_vec);

}

auto end1 = sc.now(); // устанавливаем конец отсчета времени события

auto time\_span1 = static\_cast<chrono::duration<double>>(end1 - start1); // высчитываем время, затраченное на событие

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

cout << "Скорость поиска элемента массива равна: " << time\_span1.count() << " секунд" << endl;

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

\_getch();

menu(root, root\_vec);

}

void individual(Tree\*& root, vector<int>& root\_vec) {

system("cls");

chrono::steady\_clock sc;

cout << setw(50) << right << "Индивидуальное задание" << endl;

cout << setw(70) << right << "Уменьшить каждый нечетный элемент на значение, введенное пользователем." << endl;

cout << setw(70) << right << "Каждый четный элемент умножьте на случайную величину в диапазоне от 1 до 5." << endl;

SetColor(6, 0);

cout << setw(55) << right << "Нажмите любую клавишу, чтобы начать.";

SetColor(7, 0);

\_getch();

cout << endl;

cout << setw(50) << right << "Бинарное дерево:" << endl;

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

auto start = sc.now();

individual\_ex(root, 0);

root = getTreeNoRepeatValues(root);

cout << setw(50) << right << "Измененное бинарное дерево:" << endl;

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

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

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

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

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

\_getch();

menu(root, root\_vec);

}

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

void print\_alarm(Tree\*& root, vector<int>& root\_vec) {

SetColor(4, 0);

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

SetColor(7, 0);

\_getch();

submenu\_create\_tree(root, root\_vec);

}

void subconf\_pT\_val(Tree\*& root, vector<int>& root\_vec) {

if (subm\_pT\_count == 3)

menu(root, root\_vec);

system("cls");

cout << setw(50) << right << "Бинарное дерево:" << endl;

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

switch (subm\_pT\_count) {

case 0: {

cout << "Прямой обход дерева: ";

preOrderTravers(root);

break;

}

case 1: {

cout << "Обратный обход дерева обход дерева: ";

postOrderTravers(root);

break;

}

case 2: {

cout << "Прямой обход дерева: ";

breadthFirst(root);

break;

}

case 3:

menu(root, root\_vec);

break;

default:

break;

}

cout << endl;

SetColor(2, 0);

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

SetColor(7, 0);

\_getch();

menu(root, root\_vec);

}

void subconf\_crT\_val(Tree\*& root, vector<int>& root\_vec) {

switch (subm\_crT\_count) {

case 0:

fill\_root\_random(root, root\_vec);

break;

case 1:

fill\_root\_self(root, root\_vec);

break;

case 2:

fill\_root\_from\_file(root, root\_vec);

break;

case 3:

menu(root, root\_vec);

break;

default:

break;

}

}

void conf\_val(Tree\*& root, vector<int>& root\_vec) {

switch (m\_count) {

case 0:

submenu\_create\_tree(root, root\_vec);

break;

case 1:

insert\_node(root, root\_vec);

break;

case 2:

if (fl\_tree)

delete\_node(root,root\_vec);

else

print\_alarm(root, root\_vec);

break;

case 3:

if (fl\_tree)

find\_node(root, root\_vec);

else

print\_alarm(root, root\_vec);

break;

case 4:

if (fl\_tree) {

ofstream fout;

fout.open("tree.txt", ios::trunc);

system("cls");

cout << setw(50) << right << "Бинарное дерево:" << endl;

fout << setw(50) << right << "Бинарное дерево:" << endl;

printTree(root, 0, " ");

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

cout << "Бинарное дерево было сохранено в файле tree.txt" << endl;

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

\_getch();

fout.close();

}

else

print\_alarm(root, root\_vec);

break;

case 5:

if (fl\_tree)

submenu\_pass\_tree(root, root\_vec);

else

print\_alarm(root, root\_vec);

break;

case 6:

if (fl\_tree)

individual(root, root\_vec);

else

print\_alarm(root, root\_vec);

break;

case 7:

SetColor(0, 0);

exit(0);

break;

default:

break;

}

menu(root, root\_vec);

}

void submenu\_pass\_tree\_choice(Tree\*& root, vector<int>& root\_vec) {

int k1;

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

if (k1 == 0xE0) { // если стрелки

switch (k1) {

case 0x48: // стрелка вверх

subm\_pT\_count--;

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

submenu\_pass\_tree(root, root\_vec);

break;

case 0x50: // стрелка вниз

subm\_pT\_count++;

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

submenu\_pass\_tree(root, root\_vec);

break;

case 0xD: // подтвердить

subconf\_pT\_val(root, root\_vec);

break;

default:

submenu\_pass\_tree\_choice(root, root\_vec);

}

}

switch (k1) {

case 0x48: // стрелка вверх

subm\_pT\_count--;

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

submenu\_pass\_tree(root, root\_vec);

break;

case 0x50: // стрелка вниз

subm\_pT\_count++;

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

submenu\_pass\_tree(root, root\_vec);

break;

case 0xD: // подтвердить

subconf\_pT\_val(root, root\_vec);

break;

default:

submenu\_pass\_tree\_choice(root, root\_vec);

}

}

void submenu\_create\_tree\_choice(Tree\*& root, vector<int>& root\_vec) {

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, root\_vec);

break;

case 0x50: // стрелка вниз

subm\_crT\_count++;

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

submenu\_create\_tree(root, root\_vec);

break;

case 0xD: // подтвердить

subconf\_crT\_val(root, root\_vec);

break;

default:

submenu\_create\_tree\_choice(root, root\_vec);

}

}

switch (k1) {

case 0x48: // стрелка вверх

subm\_crT\_count--;

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

submenu\_create\_tree(root, root\_vec);

break;

case 0x50: // стрелка вниз

subm\_crT\_count++;

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

submenu\_create\_tree(root, root\_vec);

break;

case 0xD: // подтвердить

subconf\_crT\_val(root, root\_vec);

break;

default:

submenu\_create\_tree\_choice(root, root\_vec);

}

}

void menu\_choice(Tree\*& root, vector<int>&root\_vec) {

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, root\_vec);

break;

case 0x50: // стрелка вниз

m\_count++;

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

menu(root, root\_vec);

break;

case 0xD: // подтвердить

conf\_val(root, root\_vec);

break;

default:

menu\_choice(root, root\_vec);

}

}

switch (k1) {

case 0x48: // стрелка вверх

m\_count--;

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

menu(root, root\_vec);

break;

case 0x50: // стрелка вниз

m\_count++;

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

menu(root, root\_vec);

break;

case 0xD: // подтвердить

conf\_val(root, root\_vec);

break;

default:

menu\_choice(root, root\_vec);

}

}

void submenu\_pass\_tree(Tree\*& root, vector<int>& root\_vec) {

system("cls"); // очищаем консоль

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

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

SetColor(3, 0);

cout << "Какой способ обхода дерева использовать?" << endl << endl;

SetColor(7, 0);

SetColor(5, 0);

cout << "Прямой обход" << endl;

SetColor(7, 0);

cout << "Обратный обход" << endl;

cout << "Обход в ширину дерева" << endl;

cout << "Назад" << endl;

submenu\_pass\_tree\_choice(root, root\_vec);

}

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

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

SetColor(3, 0);

cout << "Какой способ обхода дерева использовать?" << endl << endl;

SetColor(7, 0);

cout << "Прямой обход" << endl;

SetColor(5, 0);

cout << "Обратный обход" << endl;

SetColor(7, 0);

cout << "Обход в ширину дерева" << endl;

cout << "Назад" << endl;

submenu\_pass\_tree\_choice(root, root\_vec);

}

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

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

SetColor(3, 0);

cout << "Какой способ обхода дерева использовать?" << endl << endl;

SetColor(7, 0);

cout << "Прямой обход" << endl;

cout << "Обратный обход" << endl;

SetColor(5, 0);

cout << "Обход в ширину дерева" << endl;

SetColor(7, 0);

cout << "Назад" << endl;

submenu\_pass\_tree\_choice(root, root\_vec);

}

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

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

SetColor(3, 0);

cout << "Какой способ обхода дерева использовать?" << endl << endl;

SetColor(7, 0);

cout << "Прямой обход" << endl;

cout << "Обратный обход" << endl;

cout << "Обход в ширину дерева" << endl;

SetColor(5, 0);

cout << "Назад" << endl;

SetColor(7, 0);

submenu\_pass\_tree\_choice(root, root\_vec);

}

}

void submenu\_create\_tree(Tree\*& root, vector<int>& root\_vec) {

system("cls"); // очищаем консоль

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

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

SetColor(3, 0);

cout << "Каким образом создаем 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, root\_vec);

}

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

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

SetColor(3, 0);

cout << "Каким образом создаем 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, root\_vec);

}

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

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

SetColor(3, 0);

cout << "Каким образом создаем 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, root\_vec);

}

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

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

SetColor(3, 0);

cout << "Каким образом создаем 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, root\_vec);

}

}

void menu(Tree\*& root, vector<int>& root\_vec) {

system("cls"); // очищаем консоль

if (m\_count == 0) {

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

SetColor(5, 0);

cout << "Создать дерево" << endl;

SetColor(7, 0);

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

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

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

cout << "Вывести дерево" << endl;

cout << "Обходы дерева" << endl;

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

cout << "Выход" << endl;

menu\_choice(root, root\_vec);

}

if (m\_count == 1) {

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

cout << "Создать дерево" << endl;

SetColor(5, 0);

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

SetColor(7, 0);

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

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

cout << "Вывести дерево" << endl;

cout << "Обходы дерева" << endl;

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

cout << "Выход" << endl;

menu\_choice(root, root\_vec);

}

if (m\_count == 2) {

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

cout << "Создать дерево" << endl;

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

SetColor(5, 0);

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

SetColor(7, 0);

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

cout << "Вывести дерево" << endl;

cout << "Обходы дерева" << endl;

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

cout << "Выход" << endl;

menu\_choice(root, root\_vec);

}

if (m\_count == 3) {

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

cout << "Создать дерево" << endl;

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

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

SetColor(5, 0);

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

SetColor(7, 0);

cout << "Вывести дерево" << endl;

cout << "Обходы дерева" << endl;

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

cout << "Выход" << endl;

menu\_choice(root, root\_vec);

}

if (m\_count == 4) {

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

cout << "Создать дерево" << endl;

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

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

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

SetColor(5, 0);

cout << "Вывести дерево" << endl;

SetColor(7, 0);

cout << "Обходы дерева" << endl;

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

cout << "Выход" << endl;

menu\_choice(root, root\_vec);

}

if (m\_count == 5) {

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

cout << "Создать дерево" << endl;

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

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

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

cout << "Вывести дерево" << endl;

SetColor(5, 0);

cout << "Обходы дерева" << endl;

SetColor(7, 0);

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

cout << "Выход" << endl;

menu\_choice(root, root\_vec);

}

if (m\_count == 6) {

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

cout << "Создать дерево" << endl;

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

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

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

cout << "Вывести дерево" << endl;

cout << "Обходы дерева" << endl;

SetColor(5, 0);

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

SetColor(7, 0);

cout << "Выход" << endl;

menu\_choice(root, root\_vec);

}

if (m\_count == 7) {

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

cout << "Создать дерево" << endl;

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

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

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

cout << "Вывести дерево" << endl;

cout << "Обходы дерева" << endl;

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

SetColor(5, 0);

cout << "Выход" << endl;

SetColor(7, 0);

menu\_choice(root, root\_vec);

}

}

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

int main() {

setlocale(0, "");

srand(time(NULL));

Tree\* root = NULL;

vector<int> root\_vec;

menu(root, root\_vec);

\_getch();

return 0;

}