Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе №20**

Дисциплина: «Основы теории алгоритмов и структуры данных»

Тема: Бинарные деревья. Сбалансированные деревья и деревья поиска.

Вариант 19

Выполнил:

Студент группы РИС-20-1б

Ремянников Александр Владимирович

Проверила:

Доцент кафедры ИТАС

Полякова О. А.

**Пермь, 2021**

**Цель работы**

Получить опыт работы с бинарными деревьями

**Постановка задачи**

(Вариант 19)

1. Сформировать идеально сбалансированное бинарное дерево, тип информационного поля int.

2. Распечатать полученное дерево.

3. Найти максимальный элемент в дереве.

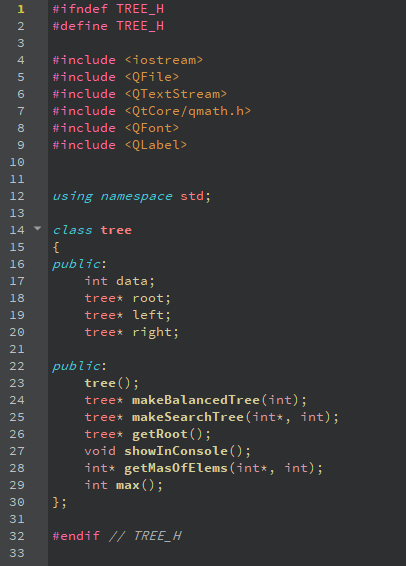
4. Преобразовать идеально сбалансированное дерево в дерево поиска.

5. Распечатать полученное дерево.

**Анализ задачи**

**1.** Для решения задачи необходимо:

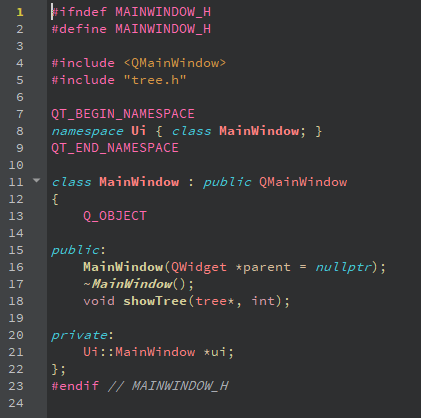
1.1. Описать пользовательский класс tree.



1.2. Реализовать функции создания сбалансированного дерева и дерева поиска.



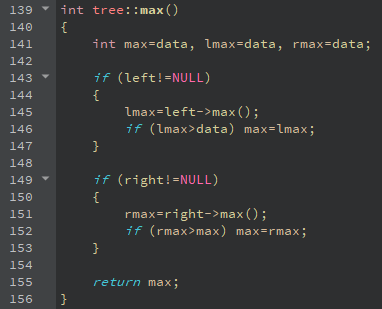
1.3. Описать пользовательский класс MainWindow.



1.4. Реализовать функцию для печати дерева на экран средствами Qt.



1.5. Реализовать функцию поиска максимального элемента дерева.



**2.** В ходе работы были использованы следующие типы данных:

2.1. Для хранения значений элементов дерева использовался тип int



**3.** Для решения задачи данные были представлены в следующем виде:

3.1. Значения элементов были представлены в виде бинарного дерева, для этого использовался пользовательский класс tree.

**4.** Для операций ввода и вывода использовались следующие операторы и функции:

4.1. Для операции вывода использовался оператор cout.



4.2. Для печать дерева средствами Qt использовался метод show().



**5.** Поставленные задачи будут решены следующими действиями:

5.1. Будет реализован код, применяющий ранее описанные методы и классы для создания бинарного сбалансированного дерева, бинарного дерева поиска, а также находящий максимальное значение, хранящееся в дереве.



**Решение**

**Main.cpp**

#include "mainwindow.h"

#include <QApplication>

#include "tree.h"

int main(int argc, char \*argv[])

{

int size=25;

int\* mas=*new* int[1000];

string temp;

tree\* userTree;

bool balanced=*true*;

*if* (balanced)*//* *Создание* *сбалансированного* *дерева*

{

userTree=userTree->makeBalancedTree(size);

}

*if* (!balanced)*//* *Создание* *дерева* *поиска*

{

mas=userTree->getMasOfElems(mas,0);

*//for* *(int* *i=0;* *i<50;* *i++)* *cout<<i<<'* *'<<mas[i]<<'\n';*

userTree=userTree->makeSearchTree(mas, 0);

}

*if* (*true*) *//* *Вывод* *дерева* *в* *консоль*

{

userTree->showInConsole();

}

*if* (*true*)*//* *Вывод* *максимального* *элемента* *дерева*

{

cout<<"\nMax element in tree = "<<userTree->max()<<'\n';

}

*//* *Вывод* *дерева* *в* *окно*

QApplication a(*argc*, argv);

MainWindow w;

w.show();

w.showTree(userTree,1);

*return* a.exec();

}

**MainWindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include "tree.h"

QT\_BEGIN\_NAMESPACE

*namespace* **Ui** { *class* **MainWindow**; }

QT\_END\_NAMESPACE

*class* **MainWindow** : *public* QMainWindow

{

Q\_OBJECT

*public*:

**MainWindow**(QWidget \*parent = *nullptr*);

~***MainWindow***();

void **showTree**(tree\*, int);

*private*:

Ui::MainWindow \*ui;

};

#endif *//* *MAINWINDOW\_H*

**MainWindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(parent)

, ui(*new* Ui::MainWindow)

{

ui->setupUi(*this*);

}

void MainWindow::**showTree**(tree\* tree, int layer)

{

QLabel \*labelForLeaf = *new* QLabel;

labelForLeaf->setMaximumHeight(30);

labelForLeaf->setMinimumHeight(30);

labelForLeaf->setMaximumWidth(50);

QFont bold("Franklin Gothic Demi", 12);

labelForLeaf->setFont(bold);

labelForLeaf->setText(QString::number(tree->data));

labelForLeaf->setAlignment(Qt::*AlignCenter*);

*switch* (layer)

{

*case* 1:

ui->l1->addWidget(labelForLeaf);

*//cout<<"data* *=* *"<<tree->data<<"* *layer* *=* *"<<layer<<'\n';*

*break*;

*case* 2:

ui->l2->addWidget(labelForLeaf);

*//cout<<"data* *=* *"<<tree->data<<"* *layer* *=* *"<<layer<<'\n';*

*break*;

*case* 3:

ui->l3->addWidget(labelForLeaf);

*//cout<<"data* *=* *"<<tree->data<<"* *layer* *=* *"<<layer<<'\n';*

*break*;

*case* 4:

ui->l4->addWidget(labelForLeaf);

*//cout<<"data* *=* *"<<tree->data<<"* *layer* *=* *"<<layer<<'\n';*

*break*;

*case* 5:

ui->l5->addWidget(labelForLeaf);

*//cout<<"data* *=* *"<<tree->data<<"* *layer* *=* *"<<layer<<'\n';*

*break*;

*case* 6:

ui->l6->addWidget(labelForLeaf);

*//cout<<"data* *=* *"<<tree->data<<"* *layer* *=* *"<<layer<<'\n';*

*break*;

*case* 7:

ui->l7->addWidget(labelForLeaf);

*//cout<<"data* *=* *"<<tree->data<<"* *layer* *=* *"<<layer<<'\n';*

*break*;

*case* 8:

ui->l8->addWidget(labelForLeaf);

*//cout<<"data* *=* *"<<tree->data<<"* *layer* *=* *"<<layer<<'\n';*

*break*;

*case* 9:

ui->l9->addWidget(labelForLeaf);

*//cout<<"data* *=* *"<<tree->data<<"* *layer* *=* *"<<layer<<'\n';*

*break*;

*case* 10:

ui->l10->addWidget(labelForLeaf);

*//cout<<"data* *=* *"<<tree->data<<"* *layer* *=* *"<<layer<<'\n';*

*break*;

}

*if* (tree->left!=NULL)

{

showTree(tree->left, layer+1);

}

*else*

{

QLabel \*labelForLeaf = *new* QLabel;

labelForLeaf->setMaximumHeight(30);

labelForLeaf->setMinimumHeight(30);

labelForLeaf->setMaximumWidth(50);

labelForLeaf->setFont(bold);

labelForLeaf->setText("NULL");

labelForLeaf->setAlignment(Qt::*AlignCenter*);

*//cout<<"lay=* *"<<layer<<'\n';*

*switch* (layer+1)

{

*case* 2:

ui->l2->addWidget(labelForLeaf);

*break*;

*case* 3:

ui->l3->addWidget(labelForLeaf);

*break*;

*case* 4:

ui->l4->addWidget(labelForLeaf);

*break*;

*case* 5:

ui->l5->addWidget(labelForLeaf);

*break*;

*case* 6:

ui->l6->addWidget(labelForLeaf);

*break*;

*case* 7:

ui->l7->addWidget(labelForLeaf);

*break*;

*case* 8:

ui->l8->addWidget(labelForLeaf);

*break*;

*case* 9:

ui->l9->addWidget(labelForLeaf);

*break*;

*case* 10:

ui->l10->addWidget(labelForLeaf);

*break*;

}

QLabel \*labelForPadding = *new* QLabel;

labelForPadding->setStyleSheet("background: rgba(255, 0, 0, 0); border:none;");

labelForPadding->setMaximumHeight(30);

labelForPadding->setMinimumHeight(30);

labelForPadding->setMaximumWidth(50);

labelForPadding->setFont(bold);

labelForPadding->setText("");

labelForLeaf->setAlignment(Qt::*AlignCenter*);

*switch* (layer+2)

{

*case* 4:

ui->l4->addWidget(labelForPadding);

ui->l4->addWidget(labelForPadding);

*break*;

*case* 5:

ui->l5->addWidget(labelForPadding);

ui->l5->addWidget(labelForPadding);

*break*;

*case* 6:

ui->l6->addWidget(labelForPadding);

ui->l6->addWidget(labelForPadding);

*break*;

*case* 7:

ui->l7->addWidget(labelForPadding);

ui->l7->addWidget(labelForPadding);

*break*;

*case* 8:

ui->l8->addWidget(labelForPadding);

ui->l8->addWidget(labelForPadding);

*break*;

*case* 9:

ui->l9->addWidget(labelForPadding);

ui->l9->addWidget(labelForPadding);

*break*;

*case* 10:

ui->l10->addWidget(labelForPadding);

ui->l10->addWidget(labelForPadding);

*break*;

}

*switch* (layer+3)

{

*case* 1:

ui->l1->addWidget(labelForPadding);

ui->l1->addWidget(labelForPadding);

ui->l1->addWidget(labelForPadding);

*break*;

*case* 2:

ui->l2->addWidget(labelForPadding);

ui->l2->addWidget(labelForPadding);

ui->l2->addWidget(labelForPadding);

*break*;

*case* 3:

ui->l3->addWidget(labelForPadding);

ui->l3->addWidget(labelForPadding);

ui->l3->addWidget(labelForPadding);

*break*;

*case* 4:

ui->l4->addWidget(labelForPadding);

ui->l4->addWidget(labelForPadding);

ui->l4->addWidget(labelForPadding);

*break*;

*case* 5:

ui->l5->addWidget(labelForPadding);

ui->l5->addWidget(labelForPadding);

ui->l5->addWidget(labelForPadding);

*break*;

*case* 6:

ui->l6->addWidget(labelForPadding);

ui->l6->addWidget(labelForPadding);

ui->l6->addWidget(labelForPadding);

*break*;

*case* 7:

ui->l7->addWidget(labelForPadding);

ui->l7->addWidget(labelForPadding);

ui->l7->addWidget(labelForPadding);

*break*;

*case* 8:

ui->l8->addWidget(labelForPadding);

ui->l8->addWidget(labelForPadding);

ui->l8->addWidget(labelForPadding);

*break*;

*case* 9:

ui->l9->addWidget(labelForPadding);

ui->l9->addWidget(labelForPadding);

ui->l9->addWidget(labelForPadding);

*break*;

*case* 10:

ui->l10->addWidget(labelForPadding);

ui->l10->addWidget(labelForPadding);

ui->l10->addWidget(labelForPadding);

*break*;

}

}

*if* (tree->right!=NULL)

{

showTree(tree->right, layer+1);

}

*else*

{

QLabel \*labelForLeaf = *new* QLabel;

labelForLeaf->setMaximumHeight(30);

labelForLeaf->setMinimumHeight(30);

labelForLeaf->setMaximumWidth(50);

labelForLeaf->setFont(bold);

labelForLeaf->setText("NULL");

labelForLeaf->setAlignment(Qt::*AlignCenter*);

*switch* (layer+1)

{

*case* 1:

ui->l1->addWidget(labelForLeaf);

*break*;

*case* 2:

ui->l2->addWidget(labelForLeaf);

*break*;

*case* 3:

ui->l3->addWidget(labelForLeaf);

*break*;

*case* 4:

ui->l4->addWidget(labelForLeaf);

*break*;

*case* 5:

ui->l5->addWidget(labelForLeaf);

*break*;

*case* 6:

ui->l6->addWidget(labelForLeaf);

*break*;

*case* 7:

ui->l7->addWidget(labelForLeaf);

*break*;

*case* 8:

ui->l8->addWidget(labelForLeaf);

*break*;

*case* 9:

ui->l9->addWidget(labelForLeaf);

*break*;

*case* 10:

ui->l10->addWidget(labelForLeaf);

*break*;

}

QLabel \*labelForPadding = *new* QLabel;

labelForPadding->setStyleSheet("background: rgba(255, 0, 0, 0); border:none;");

labelForPadding->setMaximumHeight(30);

labelForPadding->setMinimumHeight(30);

labelForPadding->setMaximumWidth(50);

labelForPadding->setFont(bold);

labelForPadding->setText("");

labelForLeaf->setAlignment(Qt::*AlignCenter*);

*switch* (layer+2)

{

*case* 1:

ui->l1->addWidget(labelForPadding);

ui->l1->addWidget(labelForPadding);

*break*;

*case* 2:

ui->l2->addWidget(labelForPadding);

ui->l2->addWidget(labelForPadding);

*break*;

*case* 3:

ui->l3->addWidget(labelForPadding);

ui->l3->addWidget(labelForPadding);

*break*;

*case* 4:

ui->l4->addWidget(labelForPadding);

ui->l4->addWidget(labelForPadding);

*break*;

*case* 5:

ui->l5->addWidget(labelForPadding);

ui->l5->addWidget(labelForPadding);

*break*;

*case* 6:

ui->l6->addWidget(labelForPadding);

ui->l6->addWidget(labelForPadding);

*break*;

*case* 7:

ui->l7->addWidget(labelForPadding);

ui->l7->addWidget(labelForPadding);

*break*;

*case* 8:

ui->l8->addWidget(labelForPadding);

ui->l8->addWidget(labelForPadding);

*break*;

*case* 9:

ui->l9->addWidget(labelForPadding);

ui->l9->addWidget(labelForPadding);

*break*;

*case* 10:

ui->l10->addWidget(labelForPadding);

ui->l10->addWidget(labelForPadding);

*break*;

}

*switch* (layer+3)

{

*case* 1:

ui->l1->addWidget(labelForPadding);

ui->l1->addWidget(labelForPadding);

ui->l1->addWidget(labelForPadding);

*break*;

*case* 2:

ui->l2->addWidget(labelForPadding);

ui->l2->addWidget(labelForPadding);

ui->l2->addWidget(labelForPadding);

*break*;

*case* 3:

ui->l3->addWidget(labelForPadding);

ui->l3->addWidget(labelForPadding);

ui->l3->addWidget(labelForPadding);

*break*;

*case* 4:

ui->l4->addWidget(labelForPadding);

ui->l4->addWidget(labelForPadding);

ui->l4->addWidget(labelForPadding);

*break*;

*case* 5:

ui->l5->addWidget(labelForPadding);

ui->l5->addWidget(labelForPadding);

ui->l5->addWidget(labelForPadding);

*break*;

*case* 6:

ui->l6->addWidget(labelForPadding);

ui->l6->addWidget(labelForPadding);

ui->l6->addWidget(labelForPadding);

*break*;

*case* 7:

ui->l7->addWidget(labelForPadding);

ui->l7->addWidget(labelForPadding);

ui->l7->addWidget(labelForPadding);

*break*;

*case* 8:

ui->l8->addWidget(labelForPadding);

ui->l8->addWidget(labelForPadding);

ui->l8->addWidget(labelForPadding);

*break*;

*case* 9:

ui->l9->addWidget(labelForPadding);

ui->l9->addWidget(labelForPadding);

ui->l9->addWidget(labelForPadding);

*break*;

*case* 10:

ui->l10->addWidget(labelForPadding);

ui->l10->addWidget(labelForPadding);

ui->l10->addWidget(labelForPadding);

*break*;

}

}

}

MainWindow::~***MainWindow***()

{

*delete* ui;

}

**Tree.h**

#ifndef TREE\_H

#define TREE\_H

#include <iostream>

#include <QFile>

#include <QTextStream>

#include <QtCore/qmath.h>

#include <QFont>

#include <QLabel>

*using* *namespace* std;

*class* **tree**

{

*public*:

int data;

tree\* root;

tree\* left;

tree\* right;

*public*:

**tree**();

tree\* **makeBalancedTree**(int);

tree\* **makeSearchTree**(int\*, int);

tree\* **getRoot**();

void **showInConsole**();

int\* **getMasOfElems**(int\*, int);

int **max**();

};

#endif *//* *TREE\_H*

**Tree.cpp**

#include "tree.h"

QFile input("C:/Users/Sasha/Documents/QtProjects/tree29051317/elements.txt");

QTextStream stream(&input);

QTextStream stream2(&input);

QString line;

tree::**tree**()

{

data=0;

root=*this*;

left=NULL;

right=NULL;

}

tree\* tree::**makeBalancedTree**(int size)

{

*if* (!input.isOpen()) input.*open*(QFile::*ReadOnly* | QFile::*Text*);

*if* (size == 0) *return* NULL;

line = stream.readLine();

*if* (line == "") *return* NULL;

tree\* nTree= *new* tree;

nTree->data = line.toInt();

nTree->left = NULL;

nTree->right = NULL;

int leftSize, rightSize;

*if* ((size-1)%2 == 0)

{

leftSize=(size-1)/2;

rightSize=leftSize;

}

*else*

{

leftSize=size/2;

rightSize=(size-1)/2;

}

nTree->left=nTree->makeBalancedTree(leftSize);

nTree->right=nTree->makeBalancedTree(rightSize);

*return* nTree;

}

tree\* tree::**makeSearchTree**(int\* mas, int i)

{

tree\* nTree= *new* tree;

nTree->left=NULL;

nTree->right=NULL;

nTree->data=mas[i];

tree\* cur=nTree;

i++;

*while* (i<25)

{

*//cout<<i<<'* *'<<mas[i]<<'\n';*

cur=nTree;

*while* (cur!=NULL)

{

*if* (mas[i]>cur->data)

{

*if* (cur->right!=NULL)

{

cur=cur->right;

}

*else*

{

tree\* mTree=*new* tree;

mTree->data=mas[i];

mTree->left=NULL;

mTree->right=NULL;

cur->right=mTree;

*break*;

}

}

*if* (mas[i]<cur->data)

{

*if* (cur->left!=NULL)

{

cur=cur->left;

}

*else*

{

tree\* mTree=*new* tree;

mTree->data=mas[i];

mTree->left=NULL;

mTree->right=NULL;

cur->left=mTree;

*break*;

}

}

}

i++;

}

*return* nTree;

}

tree\* tree::**getRoot**()

{

*return* root;

}

int\* tree::**getMasOfElems**(int\* mas, int i)

{

*if* (!input.isOpen()) input.*open*(QFile::*ReadOnly* | QFile::*Text*);

line = stream2.readLine();

*if* (line!="")

{

mas[i]=line.toInt();

i++;

getMasOfElems(mas,i);

}

*return* mas;

}

void tree::**showInConsole**()

{

cout<<data<<"\n";

*if* (left!=NULL)

{

cout<<"left=";

left->showInConsole();

}

*else*

{

cout<<"left=NULL\n";

}

*if* (right!=NULL)

{

cout<<"right=";

right->showInConsole();

}

*else*

{

cout<<"right=NULL\n";

}

}

int tree::**max**()

{

int max=data, lmax=data, rmax=data;

*if* (left!=NULL)

{

lmax=left->max();

*if* (lmax>data) max=lmax;

}

*if* (right!=NULL)

{

rmax=right->max();

*if* (rmax>max) max=rmax;

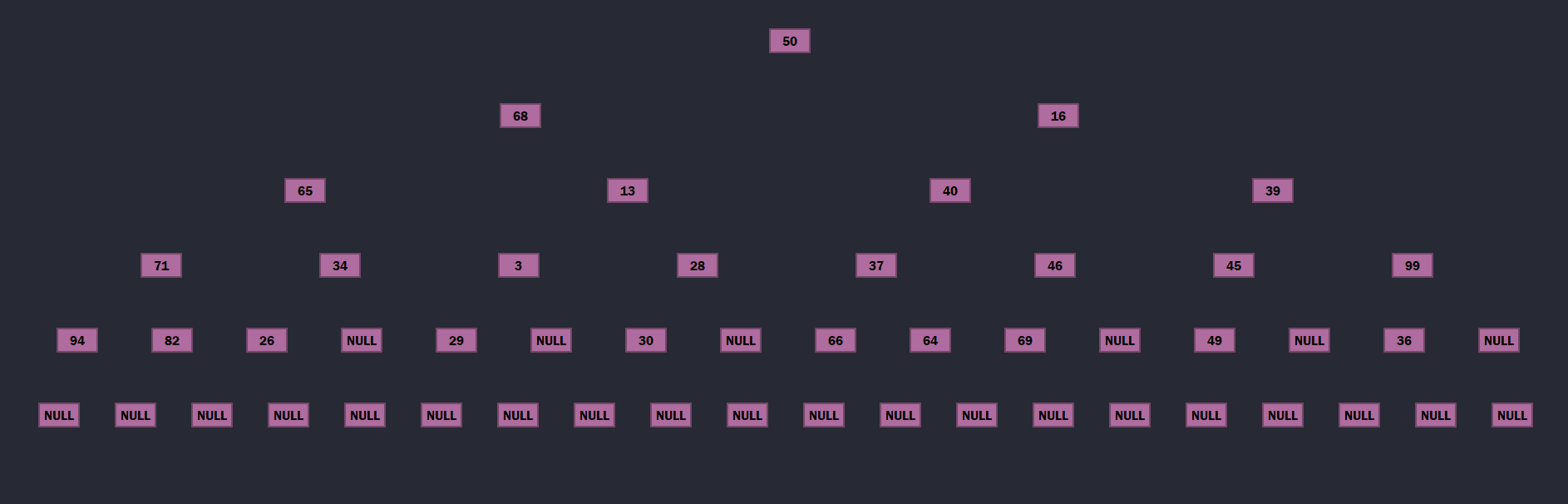
}

*return* max;

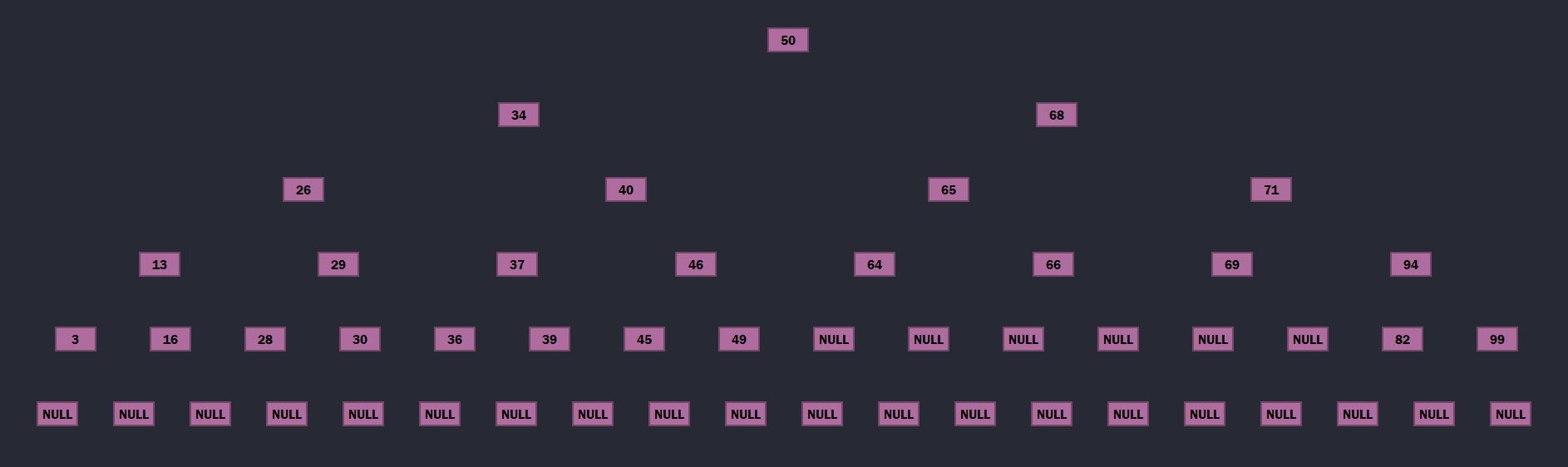
}

**Скриншоты результатов работы программы**

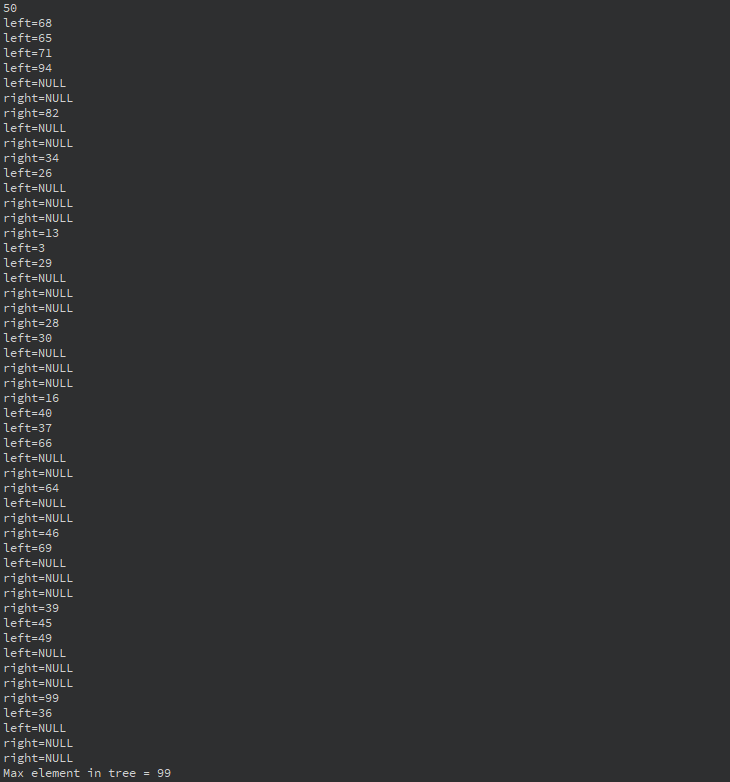
Сбалансированное дерево

****

Дерево поиска



Вывод дерева в консоль и поиск максимального



**GitHub**

[**https://github.com/sugarrrfqs/Lab20**](https://github.com/sugarrrfqs/Lab20)