Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т**

**по лабораторной работе №20**

Дисциплина: «Основы теории алгоритмов и структуры данных»  
Тема: Бинарные деревья

Вариант 14

Выполнил:

студент группы РИС-20-2б

Вичугов Алексей Дмитриевич

Проверила:

доцент кафедры ИТАС

Полякова Ольга Андреевна

Пермь, 2021

**Цель работы**

1) Получить практические навыки по работе с бинарными деревьями

2) Продемонстрировать работу с графическим интерфейсом средствами Qt при выводе бинарного дерева.

**Постановка задачи**

1) Сформировать идеально сбалансированное дерево с типом информационного поля char

2) Распечатать полученное дерево

3) Найти количество вхождений введённого пользователем элемента

4) Преобразовать идеально сбалансированное дерево в дерево поиска

5) Распечатать полученное дерево

**Анализ задачи**

Для реализации задачи необходимо:

1. Запросить у пользователя ввод количества узлов в дереве.
2. Реализовать ввод значений элементов дерева в массив.
3. Построить идеально сбалансированное дерево.
4. Перестроить идеально сбалансированное дерево в идеально сбалансированное дерево поиска.
5. Найти количество вхождений элемента, введённого пользователем.

**Консольная реализация**

1. Ввод значений элементов дерева в массив:

cout << "Enter size (<64): "; cin >> r;

while (cin.fail()||r>63||r<2){

cin.clear();

cin.ignore(5,'\n');

cout << "Incorrect input! Repeat: "; cin >> r;

}

for (int i=0;i<r;i++){

cout << i+1 << ": "; cin >> s[i];

}

1. Функция, идеально балансирующая узлы дерева:

void Maker(){

bool z=0;

while(!z){

if (current->lsize==current->rsize){

if (current->left!=NULL){

current->lsize++;

current=current->left;

dist/=2;

}

else {

current->left= new Uzel;

current->left->top=current;

current->left->left=NULL;

current->left->right=NULL;

current->left->x=current->x-dist;

current->left->y=current->y+2;

current->left->lsize=0;

current->left->rsize=0;

current->lsize++;

current=first;

dist=31;

z=1;

}

}

else{

if (current->right!=NULL){

current->rsize++;

current=current->right;

dist/=2;

}

else {

current->right= new Uzel;

current->right->top=current;

current->right->left=NULL;

current->right->right=NULL;

current->right->x=current->x+dist;

current->right->y=current->y+2;

current->right->lsize=0;

current->right->rsize=0;

current->rsize++;;

current=first;

dist=31;

z=1;

}

}

}

}

1. Алгоритм построения идеально сбалансированного дерева:

while (current!=NULL){

if (current->left!=NULL&&current->left->kras==0){

current=current->left;

glubina++;

}

else if (current->right!=NULL&&current->right->kras==0){

current=current->right;

glubina++;

}

else{

current->kras=1;

pos.X=current->x;

pos.Y=current->y;

SetConsoleCursorPosition(hCon,pos);

cout << s[Stepen(glubina)+number[glubina]-1];

number[glubina]++;

glubina--;

current=current->top;

}

}

1. Алгоритм построения идеально сбалансированного дерева поиска:

if (ans){

int min, posmin;

for (int i=0;i<r;i++) {

min=s[i];

for (int e=i;e<r;e++){

if (s[e]<min){

min=s[e];

posmin=e;

}

}

if (min<s[i]){

s[posmin]=s[i];

s[i]=min;

}

}

int indexer=0;

current=first;

while (current->left!=NULL) current=current->left;

while (current!=NULL){

if (current->left!=NULL&&current->left->kras==1){

current=current->left;

}

else if (current->kras==1){

current->kras=0;

pos.X=current->x;

pos.Y=current->y;

SetConsoleCursorPosition(hCon,pos);

cout << s[indexer];

indexer++;

}

else if (current->right!=NULL&&current->right->kras==1){

current=current->right;

}

else{

current=current->top;

}

}

}

1. Алгоритм нахождения числа вхождения введённого элемента:

char c;

int counter=0;

cout << "Enter the searchable element: "; cin >> c;

for (int i=0;i<r;i++) if (c==s[i]) counter++;

cout << "Element founded " << counter << " times. " << endl;

**Полный код программы**

#include <iostream>

#include <windows.h>

using namespace std;

int r,dist,glubina;

char s[64];

int number[6];

struct Uzel{

bool kras;

int x,y;

int lsize,rsize;

Uzel \*left, \*right, \*top;

};

Uzel \*current=NULL, \*first=NULL;

int Stepen(int s){

int k=1;

for (int i=0;i<s;i++) k\*=2;

return k;

}

void Maker(){

bool z=0;

while(!z){

if (current->lsize==current->rsize){

if (current->left!=NULL){

current->lsize++;

current=current->left;

dist/=2;

}

else {

current->left= new Uzel;

current->left->top=current;

current->left->left=NULL;

current->left->right=NULL;

current->left->x=current->x-dist;

current->left->y=current->y+2;

current->left->lsize=0;

current->left->rsize=0;

current->lsize++;

current=first;

dist=31;

z=1;

}

}

else{

if (current->right!=NULL){

current->rsize++;

current=current->right;

dist/=2;

}

else {

current->right= new Uzel;

current->right->top=current;

current->right->left=NULL;

current->right->right=NULL;

current->right->x=current->x+dist;

current->right->y=current->y+2;

current->right->lsize=0;

current->right->rsize=0;

current->rsize++;;

current=first;

dist=31;

z=1;

}

}

}

}

main(){

HANDLE hCon;

COORD pos;

hCon = GetStdHandle(STD\_OUTPUT\_HANDLE);

cout << "Enter size (<64): "; cin >> r;

while (cin.fail()||r>63||r<2){

cin.clear();

cin.ignore(5,'\n');

cout << "Incorrect input! Repeat: "; cin >> r;

}

for (int i=0;i<r;i++){

cout << i+1 << ": "; cin >> s[i];

}

dist=32;

first = new Uzel;

first->top=NULL;

first->left=NULL;

first->right=NULL;

first->lsize=0;

first->rsize=0;

current=first;

current->x=60;

current->y=70;

for (int i=1;i<r;i++) Maker();

int glubina=0;

current=first;

number[0]=0;number[1]=0;number[2]=0;number[3]=0;number[4]=0;number[5]=0;

while (current!=NULL){

if (current->left!=NULL&&current->left->kras==0){

current=current->left;

glubina++;

}

else if (current->right!=NULL&&current->right->kras==0){

current=current->right;

glubina++;

}

else{

current->kras=1;

pos.X=current->x;

pos.Y=current->y;

SetConsoleCursorPosition(hCon,pos);

cout << s[Stepen(glubina)+number[glubina]-1];

number[glubina]++;

glubina--;

current=current->top;

}

}

bool ans=0;

pos.X=0;

pos.Y=85;

SetConsoleCursorPosition(hCon,pos);

cout << "Rebuild in search tree (1-0)? "; cin >> ans;

if (ans){

int min, posmin;

for (int i=0;i<r;i++) {

min=s[i];

for (int e=i;e<r;e++){

if (s[e]<min){

min=s[e];

posmin=e;

}

}

if (min<s[i]){

s[posmin]=s[i];

s[i]=min;

}

}

int indexer=0;

current=first;

while (current->left!=NULL) current=current->left;

while (current!=NULL){

if (current->left!=NULL&&current->left->kras==1){

current=current->left;

}

else if (current->kras==1){

current->kras=0;

pos.X=current->x;

pos.Y=current->y;

SetConsoleCursorPosition(hCon,pos);

cout << s[indexer];

indexer++;

}

else if (current->right!=NULL&&current->right->kras==1){

current=current->right;

}

else{

current=current->top;

}

}

}

pos.X=0;

pos.Y=87;

SetConsoleCursorPosition(hCon,pos);

char c;

int counter=0;

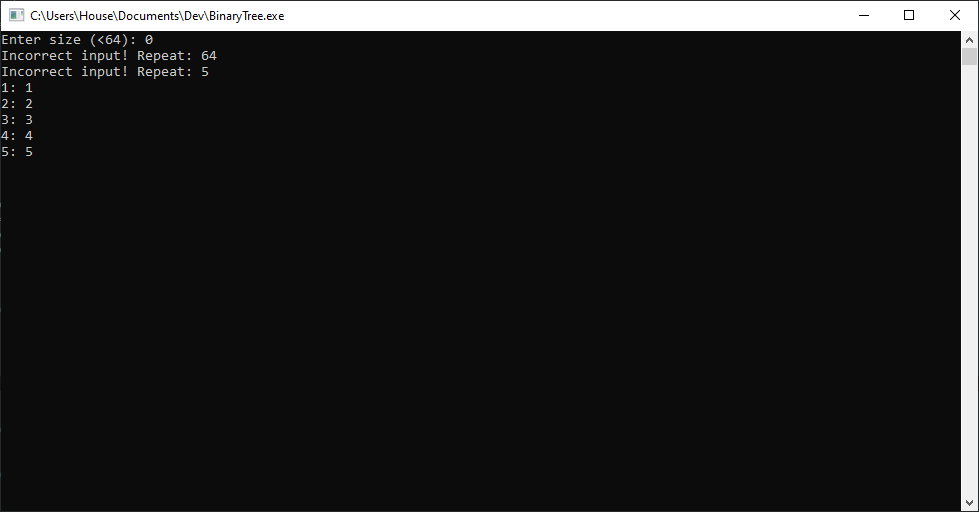
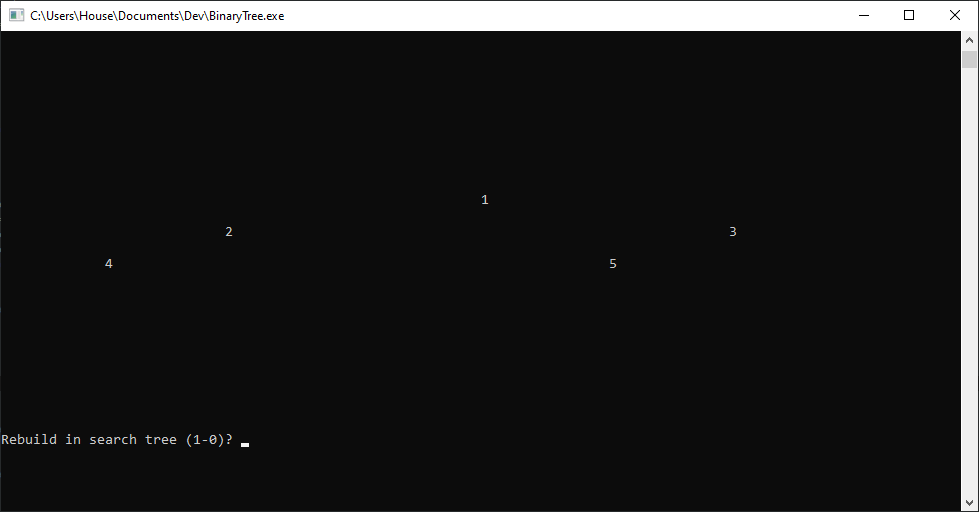
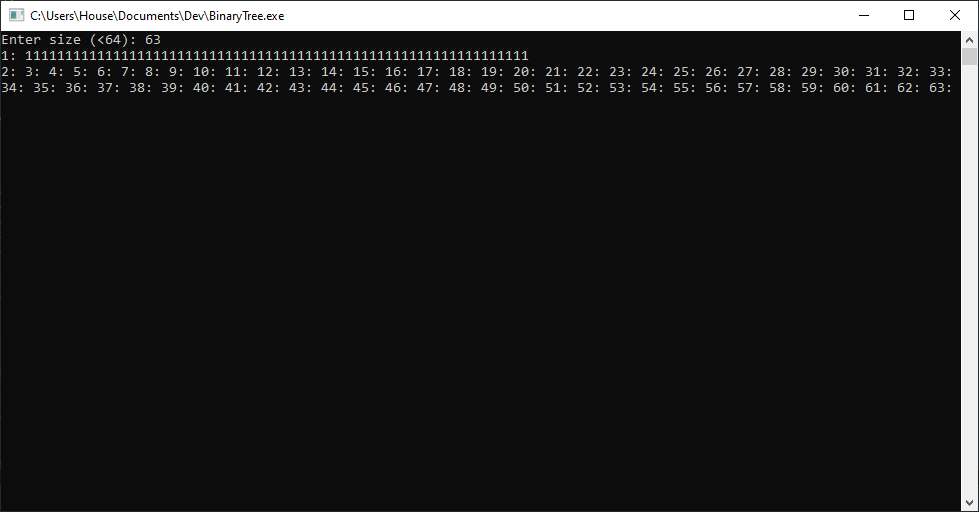
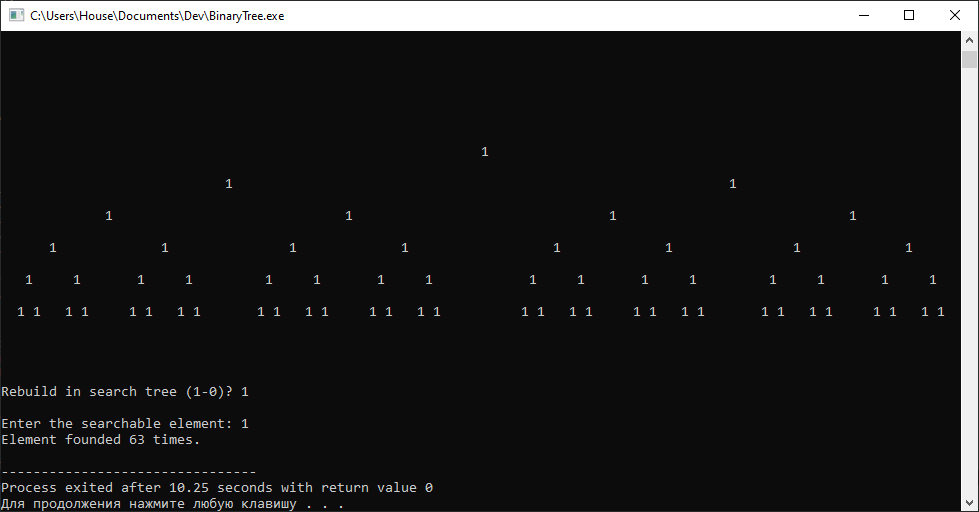
cout << "Enter the searchable element: "; cin >> c;

for (int i=0;i<r;i++) if (c==s[i]) counter++;

cout << "Element founded " << counter << " times. " << endl;

}

**Результат работы программы**

**Реализация на базе Qt**

1. Ввод значений:

void Tree::**on\_enterbutton\_clicked**()

{

if (index<r){

if (ui->inputline->text().length()>0){

s[index++]=\*(ui->inputline->text().begin());

ui->inputline->setText("");

QString vvod = "Введите элемент номер " + QString::number(index+1) + ':';

ui->textlabel->setText(vvod);

ui->statusbar->showMessage("");

}

else ui->statusbar->showMessage("Нельзя вводить пустые значения! ");

if (index==r) EndInput();

}

}

1. Функция, идеально балансирующая узлы дерева:

void Tree::**Maker**(){

bool z=0;

while(!z){

if (current->lsize==current->rsize){

if (current->left!=NULL){

current->lsize++;

current=current->left;

distance/=2;

}

else {

current->left= new Uzel;

current->left->top=current;

current->left->left=NULL;

current->left->right=NULL;

current->left->x=current->x-distance;

current->left->y=current->y+100;

current->left->lsize=0;

current->left->rsize=0;

current->lsize++;

current=first;

distance=300;

z=1;

}

}

else{

if (current->right!=NULL){

current->rsize++;

current=current->right;

distance/=2;

}

else {

current->right= new Uzel;

current->right->top=current;

current->right->left=NULL;

current->right->right=NULL;

current->right->x=current->x+distance;

current->right->y=current->y+100;

current->right->lsize=0;

current->right->rsize=0;

current->rsize++;;

current=first;

distance=300;

z=1;

}

}

}

}

1. Функция, рисующая дерево в окне программы (вид изображённого дерева зависит от целочисленной переменной b):

void Tree::***paintEvent***(QPaintEvent \*)

{

if (b==1){

int glubina=0;

current=first;

number[0]=0;number[1]=0;number[2]=0;number[3]=0;number[4]=0;number[5]=0;

revers=!revers;

pen.setColor(Qt::red);

pen.setWidth(3);

painter.begin(this);

painter.setPen(pen);

while (current!=NULL){

if (current->left!=NULL&&current->left->kras==revers){

current=current->left;

glubina++;

}

else if (current->right!=NULL&&current->right->kras==revers){

current=current->right;

glubina++;

}

else{

current->kras=!revers;

painter.drawEllipse(current->x,current->y,radius,radius);

if (current->top!=NULL) painter.drawLine(current->x+radius/2,current->y,current->top->x+radius/2,current->top->y+radius);

painter.drawText(current->x+radius/2,current->y+radius/2+2,s[Step(glubina)+number[glubina]-1]);

number[glubina]++;

glubina--;

current=current->top;

}

}

painter.end();

}

if (b==2){

int indexer=1;

current=first;

revers=!revers;

pen.setColor(Qt::red);

pen.setWidth(3);

painter.begin(this);

painter.setPen(pen);

while (current->left!=NULL) current=current->left;

while (current!=NULL){

if (current->left!=NULL&&current->left->kras==revers){

current=current->left;

}

else if (current->kras==revers){

current->kras=!revers;

painter.drawEllipse(current->x,current->y,radius,radius);

if (current->top!=NULL) painter.drawLine(current->x+radius/2,current->y,current->top->x+radius/2,current->top->y+radius);

painter.drawText(current->x+radius/2,current->y+radius/2+2,s[indexer-1]);

indexer++;

}

else if (current->right!=NULL&&current->right->kras==revers){

current=current->right;

}

else{

current=current->top;

}

}

painter.end();

}

}

1. Функция, вычисляющая количество вхождений введённого элемента:

void Tree::**on\_gosearchbutton\_clicked**()

{

ui->gosearchbutton->hide();

QChar minsim;

int minpos;

for (int i=0;i<r;i++){

minsim=s[i];

minpos=i;

for (int e=i;e<r;e++){

if (s[e]<minsim){

minpos=e;

minsim=s[e];

}

}

if (minsim<s[i]){

s[minpos]=s[i];

s[i]=minsim;

}

}

b=2;

this->repaint();

}

**Полный код программы**

1. Заголовочный файл:

#ifndef TREE\_H

#define TREE\_H

#include <QPainter>

#include <QMainWindow>

QT\_BEGIN\_NAMESPACE

namespace **Ui** { class **Tree**; }

QT\_END\_NAMESPACE

class **Tree** : public QMainWindow

{

Q\_OBJECT

public:

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

~***Tree***();

private:

Ui::Tree \*ui;

protected:

void ***paintEvent***(QPaintEvent\*) override;

private slots:

void **on\_enterbutton\_clicked**();

void **EndInput**();

void **Go**();

void **Maker**();

void **on\_startbutton\_clicked**();

void **on\_gotree\_clicked**();

void **on\_simvolbutton\_clicked**();

void **on\_gosearchbutton\_clicked**();

};

#endif // TREE\_H

1. Файл-источник:

#include "tree.h"

#include "ui\_tree.h"

#include <QPainter>

bool revers=1;

int b=0,r=0,index=0,radius=50,distance=300;

int number[6];

QChar s[64];

QPainter painter;

QPen pen;

struct **Uzel**{

bool kras=0;

int x=0,y=0;

int lsize,rsize;

Uzel \*left, \*right, \*top;

};

Uzel \*current=NULL, \*first=NULL;

Tree::**Tree**(QWidget \*parent): QMainWindow(parent), ui(new Ui::Tree)

{

ui->setupUi(this);

ui->enterbutton->setShortcut(Qt::Key\_Return);

ui->inputline->hide();

ui->textlabel->hide();

ui->stextlabel->hide();

ui->enterbutton->hide();

ui->gotree->hide();

ui->simvolbutton->hide();

ui->simvolline->hide();

ui->gosearchbutton->hide();

}

Tree::~***Tree***()

{

delete ui;

}

int **Step**(int k){

int result=1;

for (int i=0;i<k;i++) result\*=2;

return result;

}

void Tree::**Maker**(){

bool z=0;

while(!z){

if (current->lsize==current->rsize){

if (current->left!=NULL){

current->lsize++;

current=current->left;

distance/=2;

}

else {

current->left= new Uzel;

current->left->top=current;

current->left->left=NULL;

current->left->right=NULL;

current->left->x=current->x-distance;

current->left->y=current->y+100;

current->left->lsize=0;

current->left->rsize=0;

current->lsize++;

current=first;

distance=300;

z=1;

}

}

else{

if (current->right!=NULL){

current->rsize++;

current=current->right;

distance/=2;

}

else {

current->right= new Uzel;

current->right->top=current;

current->right->left=NULL;

current->right->right=NULL;

current->right->x=current->x+distance;

current->right->y=current->y+100;

current->right->lsize=0;

current->right->rsize=0;

current->rsize++;;

current=first;

distance=300;

z=1;

}

}

}

}

void Tree::**Go**(){

for (int i=1;i<r;i++){

Maker();

}

}

void Tree::**EndInput**()

{

ui->inputline->hide();

ui->textlabel->hide();

ui->enterbutton->hide();

ui->gotree->show();

first = new Uzel;

first->x=600-radius/2; first->y=200; first->top=NULL; first->left=NULL; first->right=NULL; first->lsize=0; first->rsize=0;

current=first;

index=0;

}

void Tree::***paintEvent***(QPaintEvent \*)

{

if (b==1){

int glubina=0;

current=first;

number[0]=0;number[1]=0;number[2]=0;number[3]=0;number[4]=0;number[5]=0;

revers=!revers;

pen.setColor(Qt::red);

pen.setWidth(3);

painter.begin(this);

painter.setPen(pen);

while (current!=NULL){

if (current->left!=NULL&&current->left->kras==revers){

current=current->left;

glubina++;

}

else if (current->right!=NULL&&current->right->kras==revers){

current=current->right;

glubina++;

}

else{

current->kras=!revers;

painter.drawEllipse(current->x,current->y,radius,radius);

if (current->top!=NULL) painter.drawLine(current->x+radius/2,current->y,current->top->x+radius/2,current->top->y+radius);

painter.drawText(current->x+radius/2,current->y+radius/2+2,s[Step(glubina)+number[glubina]-1]);

number[glubina]++;

glubina--;

current=current->top;

}

}

painter.end();

}

if (b==2){

int indexer=1;

current=first;

revers=!revers;

pen.setColor(Qt::red);

pen.setWidth(3);

painter.begin(this);

painter.setPen(pen);

while (current->left!=NULL) current=current->left;

while (current!=NULL){

if (current->left!=NULL&&current->left->kras==revers){

current=current->left;

}

else if (current->kras==revers){

current->kras=!revers;

painter.drawEllipse(current->x,current->y,radius,radius);

if (current->top!=NULL) painter.drawLine(current->x+radius/2,current->y,current->top->x+radius/2,current->top->y+radius);

painter.drawText(current->x+radius/2,current->y+radius/2+2,s[indexer-1]);

indexer++;

}

else if (current->right!=NULL&&current->right->kras==revers){

current=current->right;

}

else{

current=current->top;

}

}

painter.end();

}

}

void Tree::**on\_gotree\_clicked**()

{

Go();

b=1;

this->repaint();

ui->gotree->hide();

ui->stextlabel->show();

ui->simvolbutton->show();

ui->simvolline->show();

ui->gosearchbutton->show();

}

void Tree::**on\_enterbutton\_clicked**()

{

if (index<r){

if (ui->inputline->text().length()>0){

s[index++]=\*(ui->inputline->text().begin());

ui->inputline->setText("");

QString vvod = "Введите элемент номер " + QString::number(index+1) + ':';

ui->textlabel->setText(vvod);

ui->statusbar->showMessage("");

}

else ui->statusbar->showMessage("Нельзя вводить пустые значения! ");

if (index==r) EndInput();

}

}

void Tree::**on\_startbutton\_clicked**()

{

if (ui->startline->text().toInt()>1&&ui->startline->text().toInt()<64){

if (ui->startline->text().toInt()>31) radius=30;

ui->startlabel->hide();

ui->startbutton->hide();

ui->startline->hide();

r=ui->startline->text().toInt();

ui->statusbar->showMessage("");

ui->inputline->show();

ui->textlabel->show();

ui->enterbutton->show();

}

else{

ui->statusbar->showMessage("Количество узлов должно являться числом, быть больше 1 и не превышать 63! ");

ui->startline->setText("");

}

}

void Tree::**on\_simvolbutton\_clicked**()

{

int meet=0;

if (ui->simvolline->text().length()!=1){

ui->simvolline->setText("");

ui->statusbar->showMessage("Должен быть введён символ! ");

}

else{

ui->statusbar->showMessage("");

QChar simvol = \*(ui->simvolline->text().begin());

for (int i=0;i<r;i++) if (s[i]==simvol) meet++;

QString stroka = "Введённый символ встретился в дереве " + QString::number(meet) + " раз(а).";

ui->stextlabel->setText(stroka);

}

}

void Tree::**on\_gosearchbutton\_clicked**()

{

ui->gosearchbutton->hide();

QChar minsim;

int minpos;

for (int i=0;i<r;i++){

minsim=s[i];

minpos=i;

for (int e=i;e<r;e++){

if (s[e]<minsim){

minpos=e;

minsim=s[e];

}

}

if (minsim<s[i]){

s[minpos]=s[i];

s[i]=minsim;

}

}

b=2;

this->repaint();

}

Файл, содержащий функцию main():

#include "tree.h"

#include <QApplication>

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

{

QApplication a(*argc*, argv);

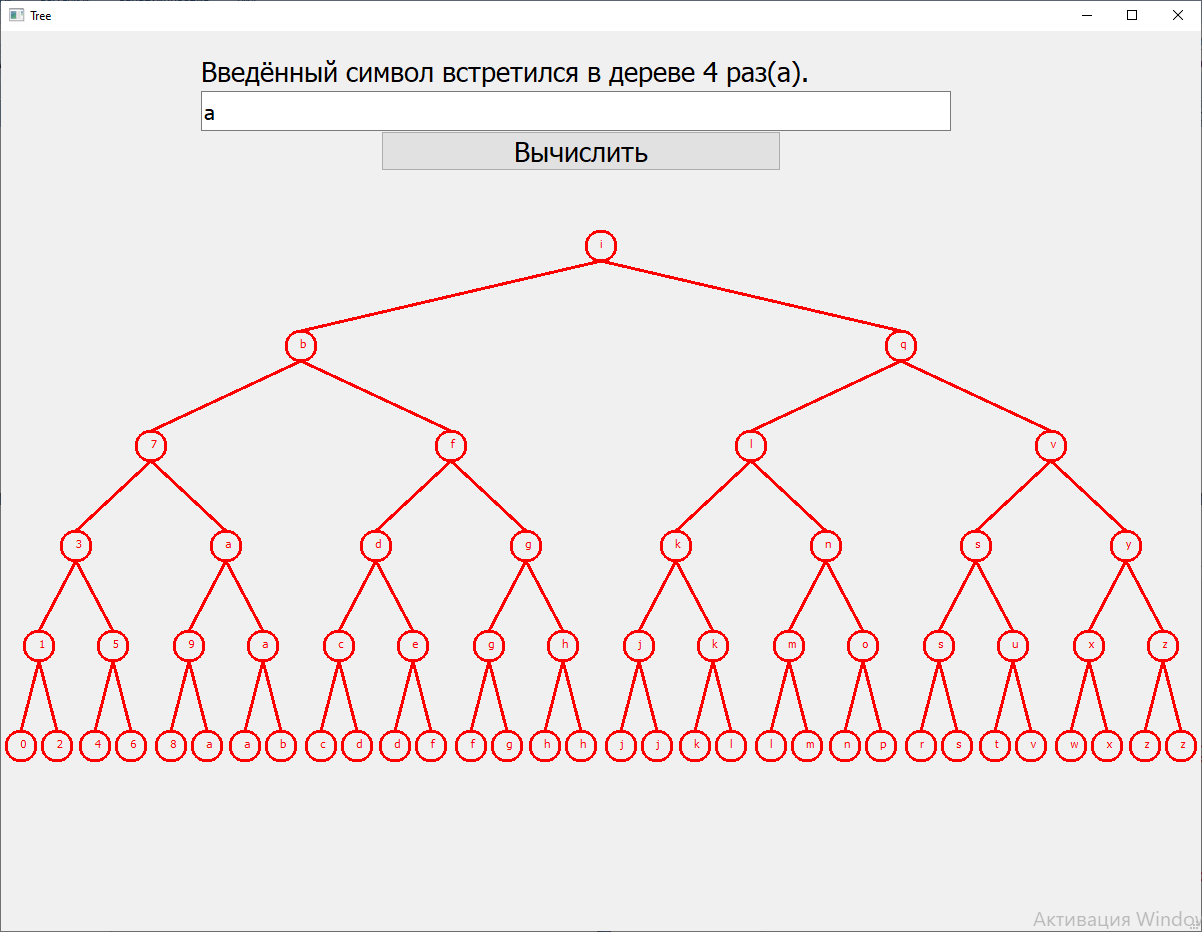
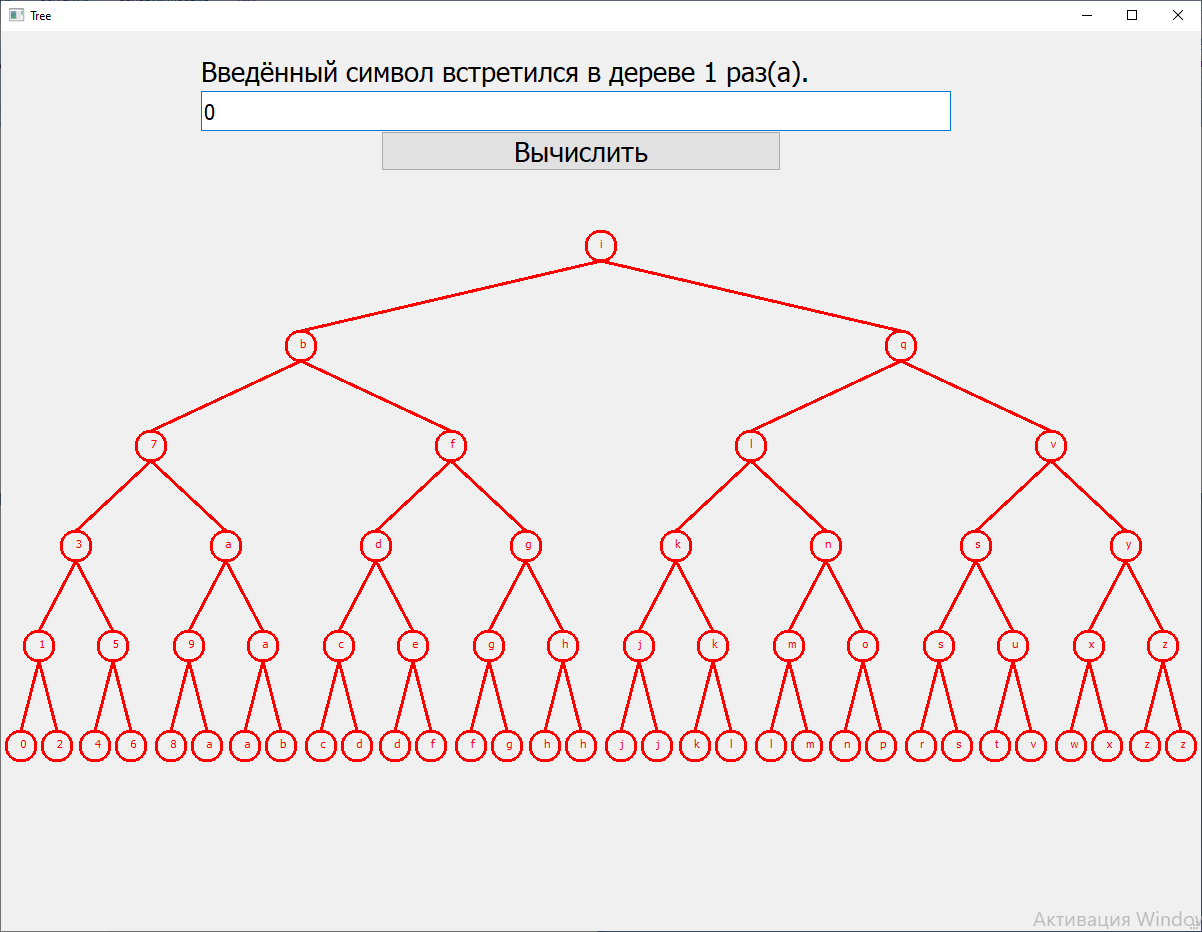
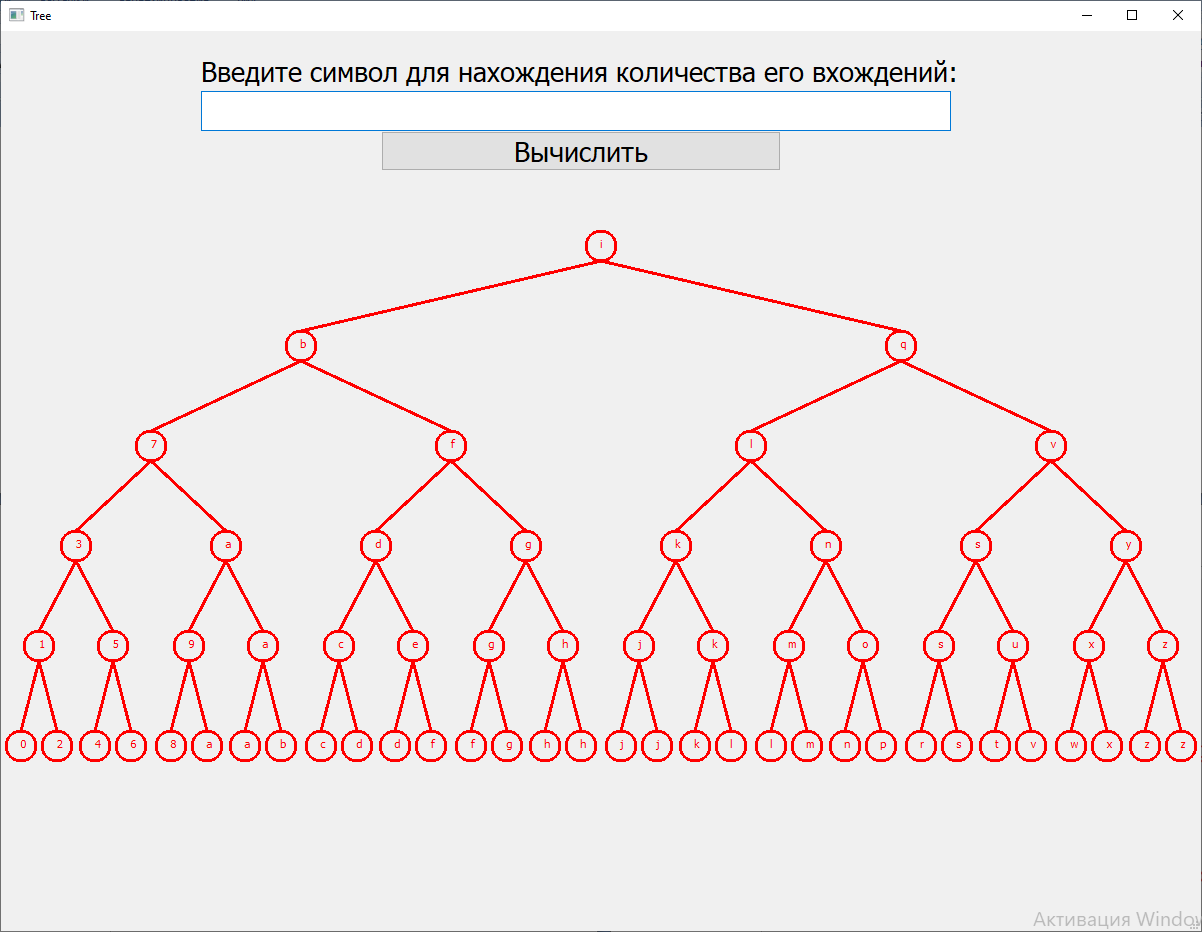
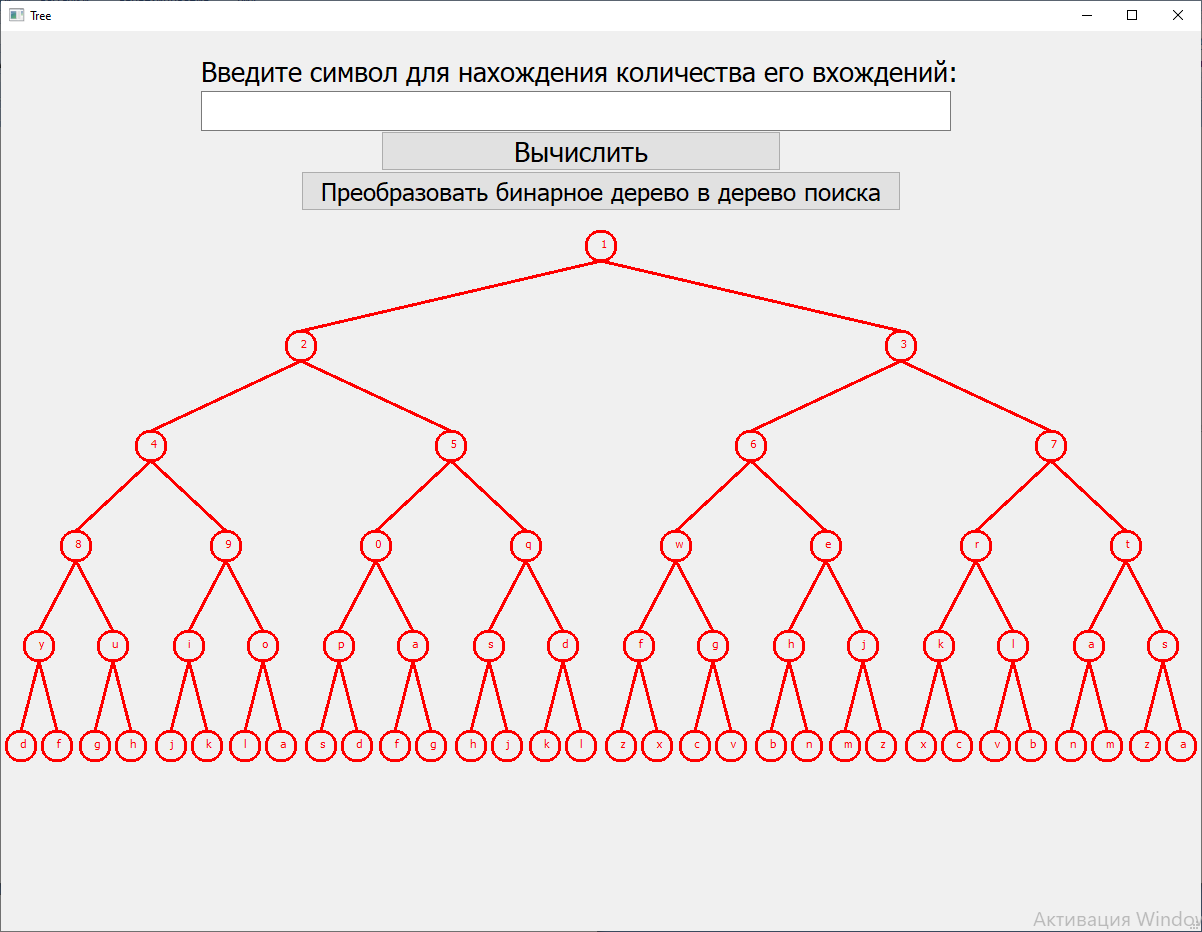
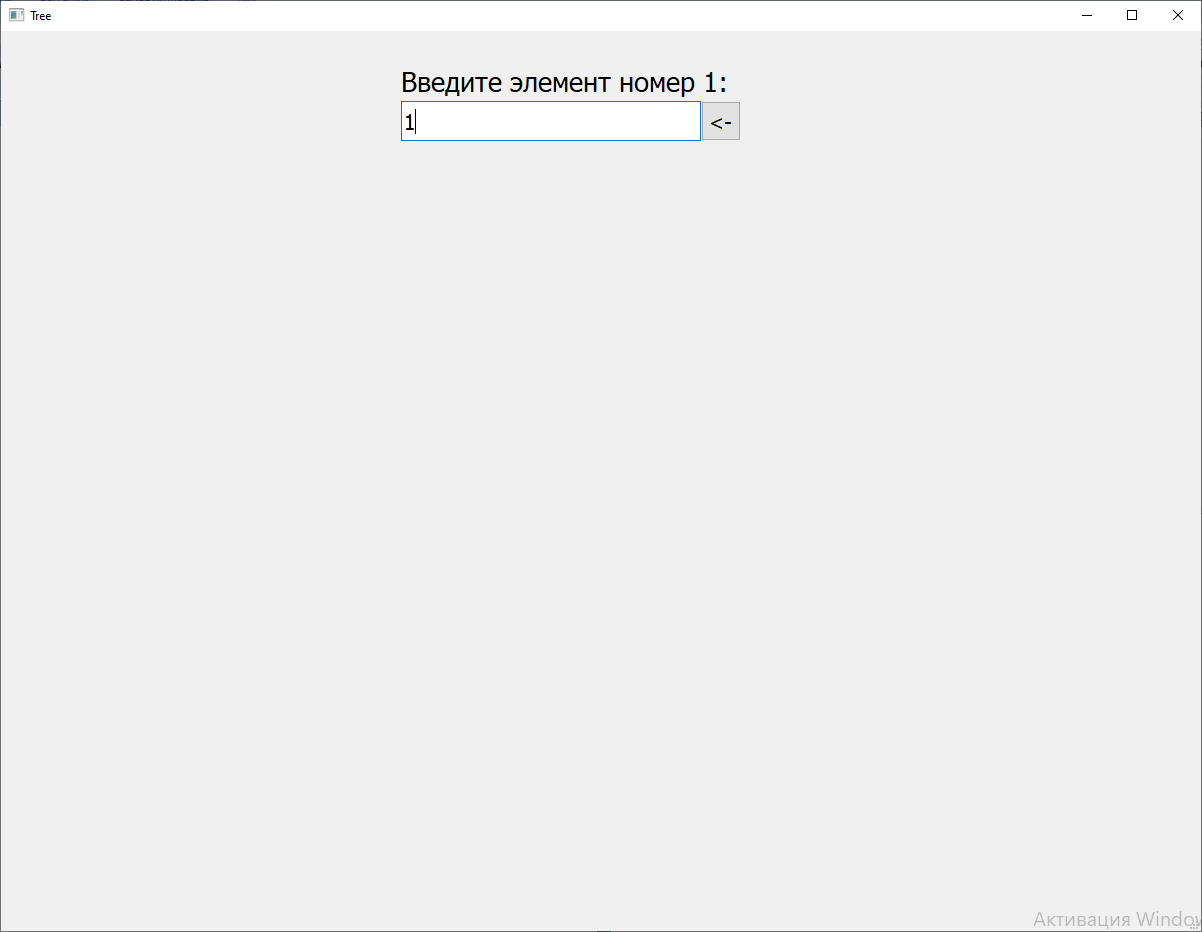
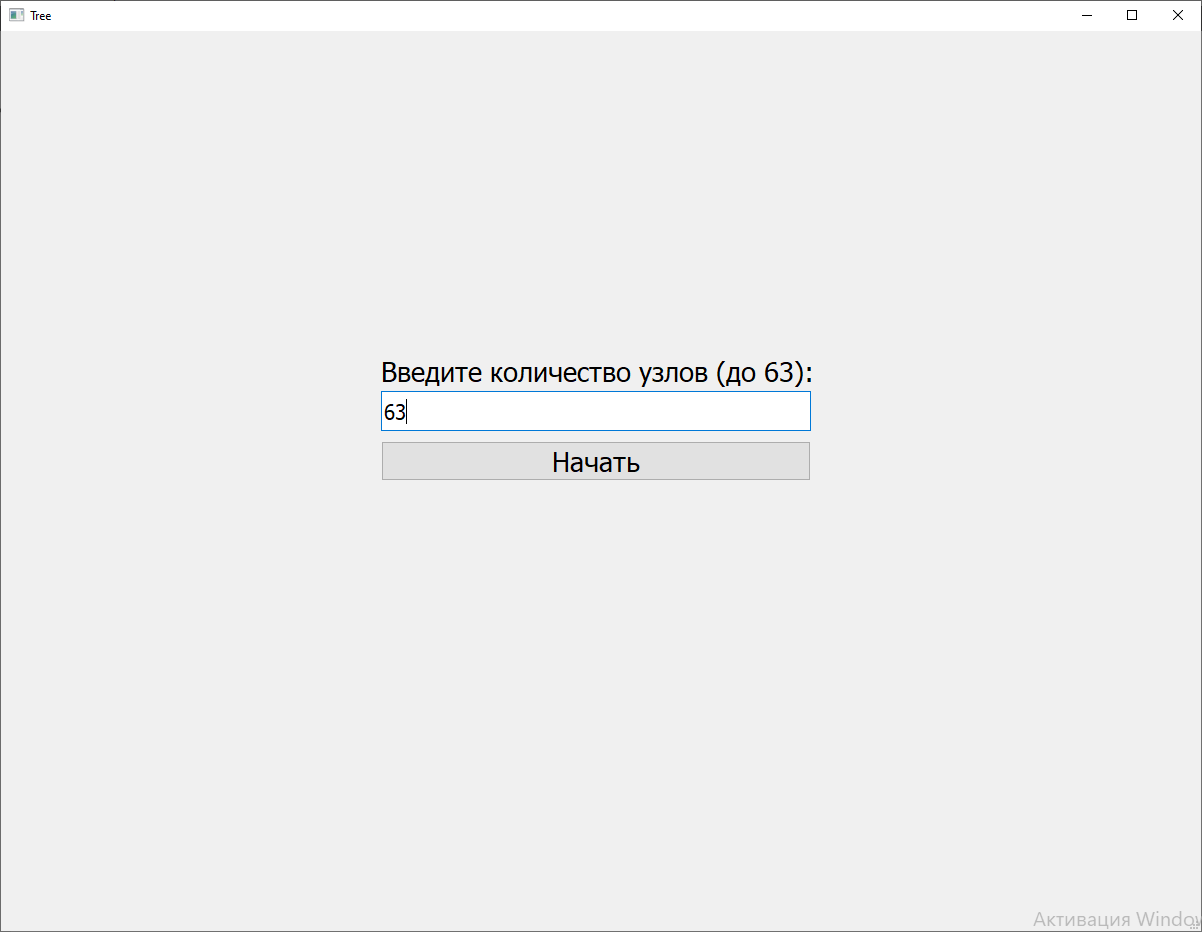
Tree w;

w.show();

return a.exec();

}

**Результат работы программы**



**Диаграмма классов**

