МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Алгоритмы и структуры данных»

Тема: Программирование алгоритмов с бинарными деревьями

Студентка гр. 8381 _____ Звегинцева Е.Н.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2019

Цель работы

Познакомиться с такой часто используемой на практике, особенно при решении задач кодирования и поиска, нелинейной структурой данных, как бинарное дерево, способами её представления и реализации, получить навыки решения задач обработки бинарных деревьев.

Основные теоретические положения

Бинарное дерево − конечное множество узлов, которое либо пусто, либо состоит из корня и двух непересекающихся бинарных деревьев, называемых правым поддеревом и левым поддеревом. Так определенное бинарное дерево не является частным случаем дерева. Например, бинарные деревья, изображенные на рис. 1, различны между собой, так как в одном случае корень имеет пустое правое поддерево, а в другом случае правое поддерево непусто. Если же их рассматривать как деревья, то они идентичны.

Рис. 1: Бинарные дерерья из двух узлов

Определим скобочное представление бинарного дерева (БД):

< БД > ::= < пусто > | < непустое БД >,

< пусто > ::= ^,

< непустое БД > ::= (< корень > < БД > < БД >).

Здесь пустое дерево имеет специальное обозначение − **^.**

Например, бинарное дерево, изображенное на рис. 3.4, имеет скобочное представление

(a (b (d Λ (h Λ Λ)) (e Λ Λ)) (c (f (i Λ Λ) (j Λ Λ)) (g Λ (k (l Λ Λ) Λ)))).

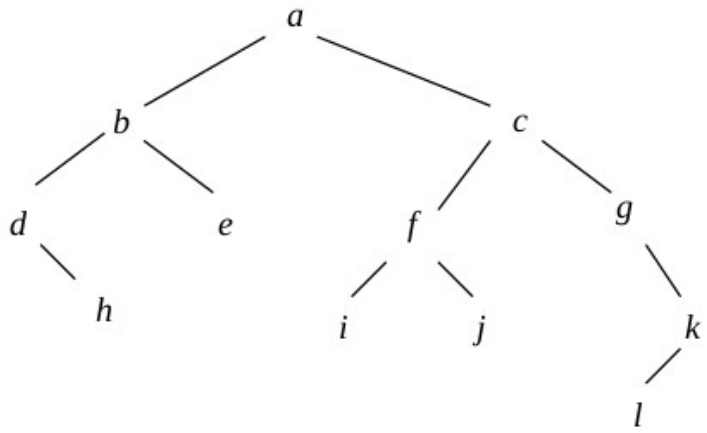

Рис. 2: Бинарное дерево

Можно упростить скобочную запись бинарного дерева, исключив «лишние» знаки по правилам:

1) (< корень > < непустое БД > ^) == (< корень > < непустое БД >),

2) (< корень > ^^) (< корень >).

Тогда, например, скобочная запись бинарного дерева, изображенного на рис. 3.4, будет иметь вид

(a (b (d ^ (h) (e)) (c (f (i) (j)) (g (k (l))))).

Задание

Для заданного бинарного дерева b типа BT с произвольным типом элементов определить, есть ли в дереве b хотя бы два одинаковых элемента.

Выполнение работы

В ходе выполнения работы была разработана программа, преобразующая строку в бинарное дерево и вычисляющая одинаковые элементы в нем, с помощью функций:

- int max_depth(Node *n, int i) – вычисление глубины бинарного дерева
- bool parse_tree(Node*& n, std::string &s, int &i) – разбиение строки на узлы, для формирования бинарного дерева
- void get_elems(array_list& vec, Node* n) – добавление элемента бинарного дерева

- bool get_duplicates(array_list&vec,Node*&first,Node*& second) – поиск одинаковых элементов бинарного дерева с использованием вектора

Также для программы был написан графический интерфейс, визуализирующий дерево и наш поиск элементов в нем.

Примеры деревьев и их представление:

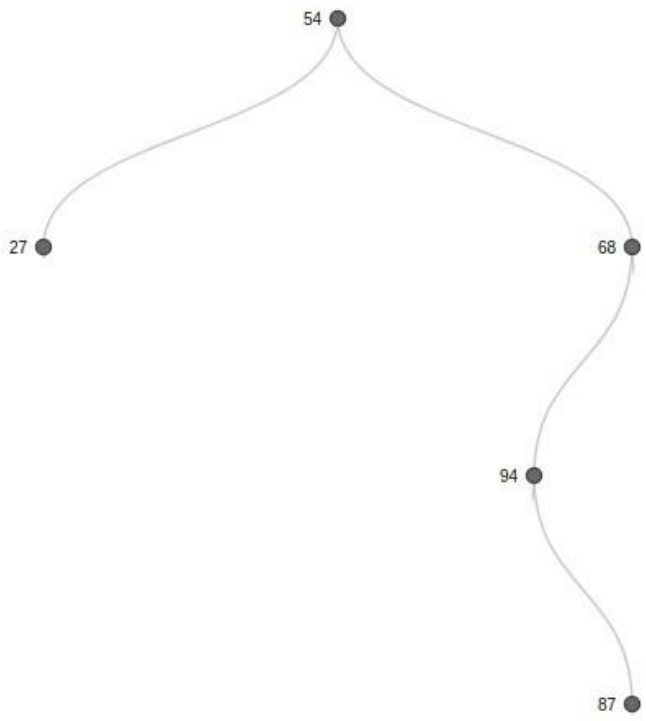

Рис. 3: Бинарное дерево "(54(27)(68(94(87))))"

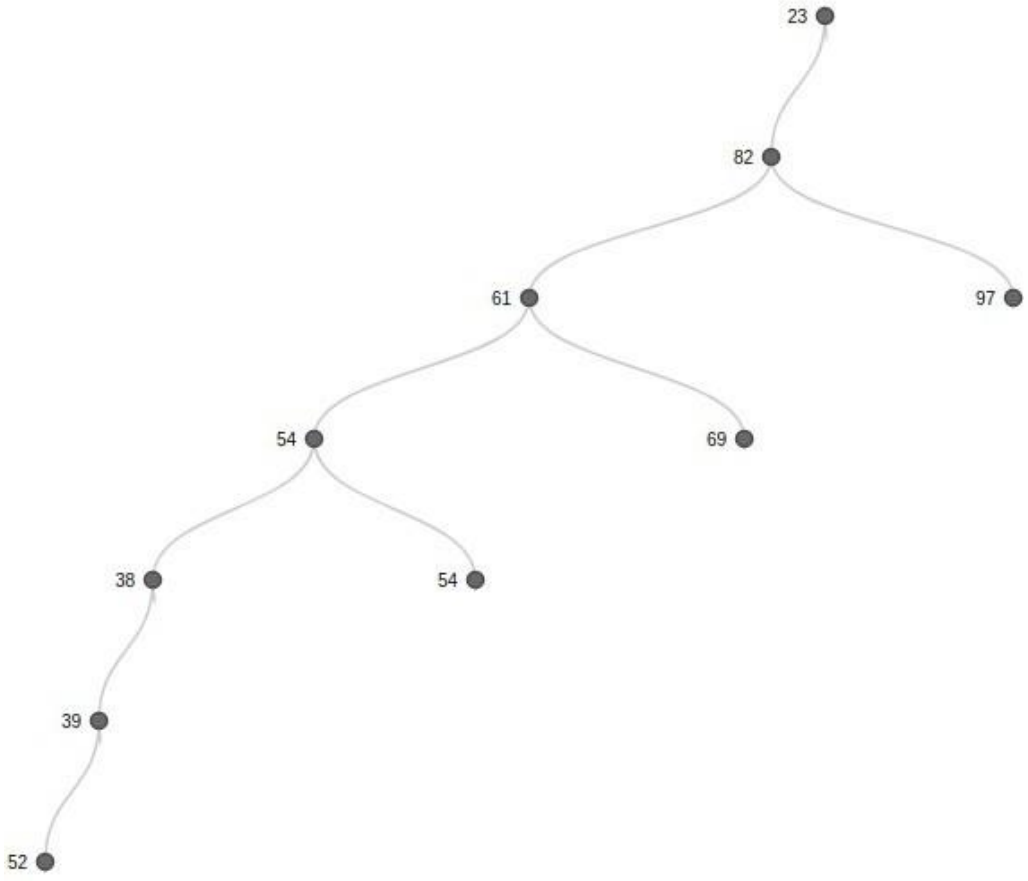

Рис. 4: Бинарное дерево "(23(82(61(54(38(39(52)))(54))(69))(97)))"

Тестирование

Введенные данные	Результат
(23(82(61(54(38(39(52)))(54))(69))(97))	There are two or more equal elems with value 54
(54(27)(68(94(87))))	There are not two or more equal elems
(99(1))	There are not two or more equal elems

Оценка эффективности алгоритма.

Алгоритм нахождения одинаковых элементов бинарного дерева имеет линейную сложность O(n).

При чтении бинарного дерева, мы сравниваем каждый элемент с каждым элементом.

Оценка алгоритма осуществляется соотношением количества элементов к количеству операций, в виду того, что мы опускаем время выполнения различных логических операций в следствии с сильной зависимость от косвенных факторов(загруженность системы и тп). У нас наблюдается увеличение числа операций с увеличением числа элементов.

На основе промежуточных выводов через qDebug() в ходе выполнения программы и последующего анализа выходной последовательности операций был построен график, представленный на рис. 1.

Как мы видим, наш алгоритм обладает линейной сложностью.

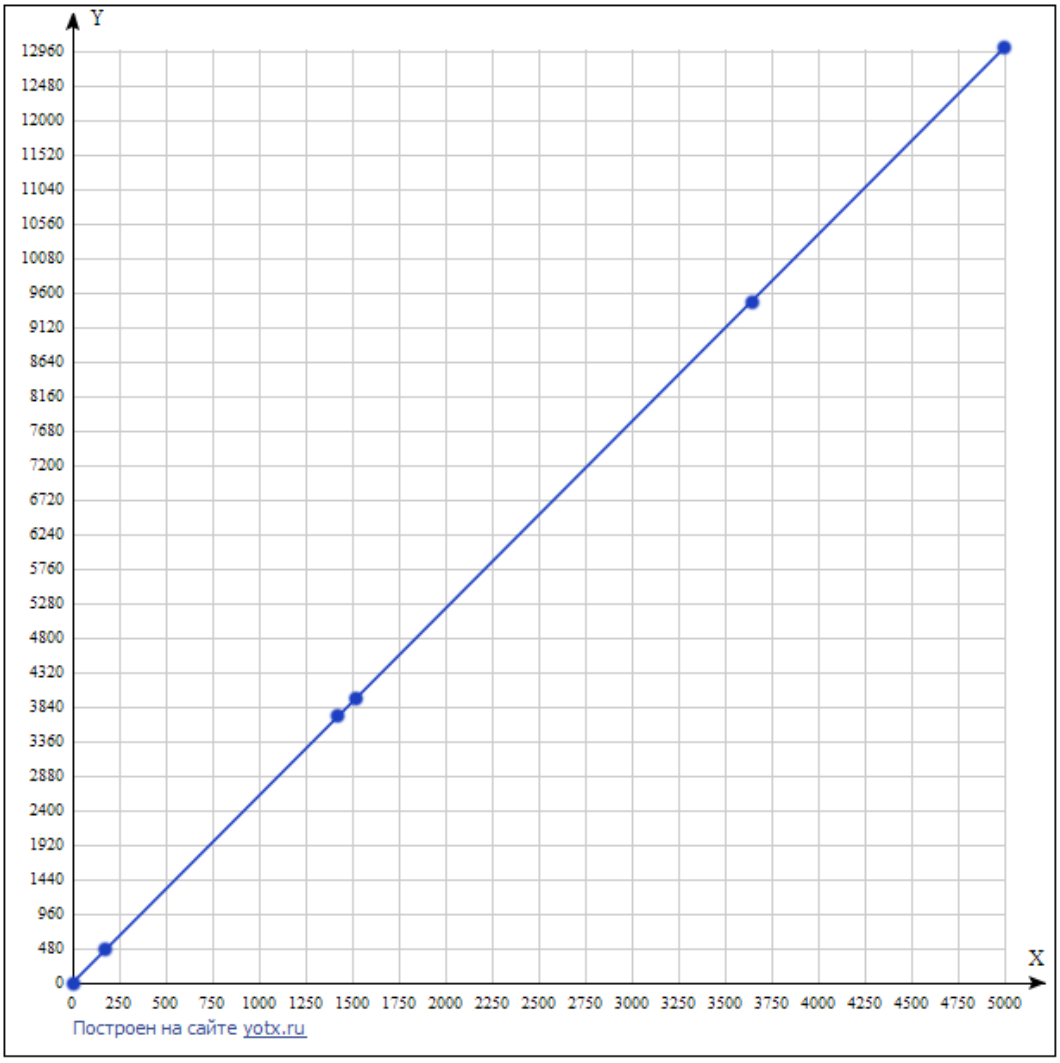

Рисунок 1 - анализ эффективности алгоритма

Вывод

В ходе работы были приобретены навыки работы с бинарными деревьями, изучены методы обхода бинарных деревьев и реализация деревьев на базе указателей на структуру.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include "mainwindow.h"
#include <QApplication>

int execute(int argc, char *argv[])
{
    std::cout << "Starting in console mode..." << std::endl;
//    utils_tree<int>* tree = nullptr;
//    for(int i = 0; i < argc; i++) {
//        if(!strcmp("console", argv[i]) || !strcmp("-console", argv[i]) ||
!strcmp("-c", argv[i])) continue;
//        std::cout << "Tree found: " << argv[i] << std::endl;
//        std::string s = argv[i];
//        tree = new utils_tree<int>(s);
//    }
//    if (!tree) return 1;
//    if (tree->is_bst()) std::cout << "Tree is BST" << std::endl;
//    else std::cout << "Tree is not BST" << std::endl;
//    if (tree->is_pyramid()) std::cout << "Tree is pyramid" << std::endl;
//    else std::cout << "Tree is not pyramid" << std::endl;
//    delete tree;
    return 0;
}


int main(int argc, char *argv[])
{
    for(int i = 0; i < argc; i++) {
        if(!strcmp("console", argv[i]) || !strcmp("-console", argv[i]) ||
!strcmp("-c", argv[i])) {
            return execute(argc - 1, argv + 1);
        }
    }

    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

Название файла: mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"



std::vector<std::string> split(const std::string &s, char delim) {
    std::stringstream ss(s);
    std::string item;
    std::vector<std::string> elems;
    while (std::getline(ss, item, delim)) {
        elems.push_back(item);
        // elems.push_back(std::move(item)); // if C++11 (based on comment from
@mchiasson)
    }
    return elems;
```

```
}

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    QMainWindow::showMaximized();
    mainGraphicsScene = new QGraphicsScene();
    ui->graphicsView->setScene(mainGraphicsScene);
    tree = nullptr;
    pen.setColor(QColor::fromRgb(0, 255, 0));
    started = false;
    first = nullptr;
    second = nullptr;
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_action_triggered()
{
    Help* h = new Help;
    h->show();
}


void MainWindow::visualize()
{
    mainGraphicsScene->clear();
    if (!tree) return;
    draw(tree->root, tree->max_depth(tree->root, 1));

}

void MainWindow::draw(Node *n, int maxdepth, int depth, int x, int y)
{
    if (n == nullptr) return;
    int offset = pow(2, maxdepth + 3 - depth);
    // Lines
    if (n->left) mainGraphicsScene->addLine(x + 32, y + 32, x - offset + 32, y
+ 64 + 32, pen);
    if (n->right) mainGraphicsScene->addLine(x + 32, y + 32, x + offset + 32, y
+ 64 + 32, pen);
    // Ellipse
    QBrush brush(n->color);
    QPen pen(n->color, 3);
    mainGraphicsScene->addEllipse(x, y, 64, 64, pen, brush);
    // Text
    QGraphicsTextItem *numb = new QGraphicsTextItem();
    numb->setPlainText(QString::fromStdString(n->info));
    numb->setDefaultTextColor(Qt::black);
    numb->setScale(2);
    numb->setPos(x + 16, y + 8);
    mainGraphicsScene->addItem(numb);
    // Next
    draw(n->left, maxdepth, depth + 1, x - offset, y + 64);
    draw(n->right, maxdepth, depth + 1, x + offset, y + 64);
}

void MainWindow::on_pushButton_clicked()
```

```
{
    if (first) first->color.setRgb(255, 0, 0);
    if (second) second->color.setRgb(255, 0, 0);

    tree = new binTree();
    elems.clean();
    int i = 0;
    tree->root = new Node();
    std::string s = ui->lineEdit->text().toStdString();
    if(tree->parse_tree(tree->root, s, i)) {
        QMessageBox::warning(this, "Error", "Error parsing tree");
        ui->lineEdit->setText("");
        return;
    }
    tree->get_elems(elems, tree->root);
    Node* first;
    Node* second;
    if (tree->get_duplicates(elems, first, second)) {
        first->color.setRgb(0, 0, 255);
        second->color.setRgb(0, 0, 255);
        QString qs = QString::fromStdString(first->info);
        QMessageBox::information(this, "Result", "There are two or more equal
elems with value " + qs);
    }
    else {
        QMessageBox::warning(this, "Result", "There are not two or more equal
elems");
    }
    started = false;
    visualize();
}

void MainWindow::on_stepButton_clicked()
{
    if (!started) {
        tree = new binTree();
        elems.clean();
        int i = 0;
        tree->root = new Node();
        std::string s = ui->lineEdit->text().toStdString();
        if(tree->parse_tree(tree->root, s, i)) {
            QMessageBox::warning(this, "Error", "Error parsing tree");
            ui->lineEdit->setText("");
            return;
        }
        started = true;
        tree->get_elems(elems, tree->root);
        a = 0;
        b = 0;
    }
    if (a == elems.size() - 1) {
        QMessageBox::warning(this, "Result", "There are not two or more equal
elems");
        started = false;
    }
    else if (b == elems.size() - 1) {
        a++;
        b = a + 1;
    }
    else {
        b++;
    }
```

```
    if (elems[a]->info == elems[b]->info) {
        if (started) {
            QString qs = QString::fromStdString(elems[a]->info);
            QMessageBox::information(this, "Result", "There are two or more
equal elems with value " + qs);
        }
        started = false;
    }

    if (first) first->color.setRgb(255, 0, 0);
    if (second) second->color.setRgb(255, 0, 0);
    first = elems[a];
    second = elems[b];
    first->color.setRgb(0, 0, 255);
    second->color.setRgb(0, 0, 255);

    visualize();
}

void MainWindow::on_fileButton_clicked()
{
    std::string inputStr;
    QString fileName = QFileDialog::getOpenFileName(this, "Open TXT File",
QDir::homePath(), "TXT text (*.txt);;All Files (*)");
    if (fileName == nullptr)
    {
        QMessageBox::warning(this, "Warning", "File name is empty");
        return;
    }
    QFile file(fileName);
    if(file.open(QIODevice::ReadOnly | QIODevice::Text)) {
        QTextStream stream(&file);
        foreach (QString i ,QString(stream.readAll()).split(QRegExp("[ \t]"),
QString::SkipEmptyParts))
        inputStr.append(i.toUtf8().constData());
    }
    if(inputStr.empty())
        return;
    file.close();
    ui->lineEdit->setText(QString::fromUtf8(inputStr.c_str()));
}
```

Название файла: binTree.cpp

```
//
// Created by therealyou on 02.10.2019.
//

#include "binTree.h"

void binTree::lkp() {
    Node* temp = root;
    if (root){
        //elems.push_back(root->info);
        root = temp->left;
        lkp();
        root = temp->right;
        lkp();
        root = temp;
    }
}
```

```
int binTree::treeInit(QStringList lst, unsigned int& index) {
    QString x = lst[index++];
    if (x == "/"){
        delete root;
        root = nullptr;
        return 1;
    } else {
        root->info = x.toUtf8().data();
        Node* temp = root;
        root = temp->left;
        root = new Node;
        temp->left = root;
        if (treeInit(lst, index)){
            temp->left = nullptr;
        }
        root = temp->right;
        root = new Node;
        temp->right = root;
        if (treeInit(lst, index)){
            temp->right = nullptr;
        }
        root = temp;
    }
    return 0;
}

int binTree::checkTwoEqualElem() {
    //elems.clear();
    lkp();
//    for (unsigned int i = 0; i < elems.size(); i++){
//        for (unsigned int j = i + 1; j < elems.size(); j++){
//            //if (elems[i] == elems[j]){
//            //    return 1;
//            //}
//        }
//    }
    return 0;
}

void binTree::printLKP() {
    Node* temp = root;
    if (root){
        std::cout << root->info << std::endl;
        root = temp->left;
        printLKP();
        root = temp->right;
        printLKP();
        root = temp;
    }
}

int binTree::max_depth(Node *n, int i)
{
    if (!n) return i;
    int l = max_depth(n->left, i + 1);
    int r = max_depth(n->right, i + 1);
    if (l > r) return l;
    else return r;
}

bool binTree::parse_tree(Node*& n, std::string &s, int &i) {
    if (i >= s.size() || s[static_cast<unsigned long>(i)] == ')')
    {
```

```
        delete n;
        n = nullptr;
        return false;
    }
    if (s[static_cast<unsigned long>(i)] == '(')
    {
        i++;
    }
    int num;
    int start = i;
    while (i != static_cast<int>(s.size()) && s[static_cast<unsigned long>(i)]
!= '(' && s[static_cast<unsigned long>(i)] != ')')
    {
        i++;
    }
    n->info = s.substr(static_cast<unsigned long>(start), static_cast<unsigned
long>(i) - static_cast<unsigned long>(start));
    n->left = new Node();
    n->right = new Node();
    if(parse_tree(n->left, s, i) || parse_tree(n->right, s, i)) return true;
    if (s[static_cast<unsigned long>(i)] == ')')
    {
        i++;
    }
    return false;
}

void binTree::get_elems(array_list& vec, Node *n)
{
    if (!n) return;
    vec.push_back(n);
    get_elems(vec, n->left);
    get_elems(vec, n->right);
}

bool binTree::get_duplicates(array_list& vec, Node*& first, Node*& second)
{
    int count = vec.size();
    for (int i = 0; i < count; i++) {
        for (int k = i + 1; k < count; k++) {
            if (vec[i]->info == vec[k]->info) {
                first = vec[i];
                second = vec[k];
                return true;
            }
        }
    }
    return false;
}
```

Название файла: array_list.cpp

```
#ifndef UTILS_VECTOR_H
#define UTILS_VECTOR_H
#include "array_list.h"


void array_list::resize(int new_capacity)
{
    auto *arr = new Node*[count];
    for (int i = 0; i < count; ++i)
```

```
    {
        arr[i] = array[i];
    }
    delete [] array;
    array = new Node*[new_capacity];
    for (int i = 0 ; i < count; ++i)
    {
        array[i] = arr[i];
    }
    delete [] arr;
    capacity = new_capacity;
}

array_list::array_list(int start_capacity)
{
    capacity = start_capacity;
    count = 0;
    array = new Node*[capacity];
}

Node*& array_list::operator[] (int index)
{
    return array[index];
}

Node* array_list::at (int index)
{
    return operator[](index);
}

void array_list::clean ()
{
    count = 0;
}

void array_list::insert(int index, Node* element)
{
    if (capacity == count)
    {
        resize(count + 8);
    }
    if (count > 0) {
        for (int i = count; i > index; i--)
        {
            array[i] = array[i - 1];
        }
    }
    count++;
    array[index] = element;
}

Node* array_list::remove(int index)
{
    auto temp = array[index];
    for (int i = index; i < count - 1; i++)
    {
        array[i] = array[i + 1];
    }
    count--;
    return temp;
}

Node* array_list::back()
```

```
{
    return array[count - 1];
}

void array_list::push_back(Node* element)
{
    if (capacity == count)
    {
        resize(count + 8);
    }
    array[count] = element;
    count++;
}

Node* array_list::pop_back()
{
    return array[--count];
}

Node* array_list::front()
{
    return *array;
}

void array_list::push_front(Node* element)
{
    insert(0, element);
}

Node* array_list::pop_front()
{
    return remove(0);
}

int array_list::size()
{
    return count;
}

bool array_list::empty()
{
    return !count;
}

array_list::~array_list()
{
    delete [] array;
}

#endif //VECTOR_VECTOR_H
```

Название файла: binTree.h

```
//
// Created by therealyou on 02.10.2019.
//

#ifndef LAB3_BINTREE_H
#define LAB3_BINTREE_H

#include <iostream>
#include <string>
#include <vector>
```

```
#include <QStringList>
#include <QColor>
#include "array_list.h"
#include "node.h"


class binTree {
public:
    Node* root = new Node;

    int treeInit(QStringList lst, unsigned int& index);
    Node* enterBT();
    int checkTwoEqualElem();
    void lkp();
    void printLKP();
    //
    int max_depth(Node *n, int i);
    bool parse_tree(Node*& n, std::string &s, int &i);
    void get_elems(array_list& vec, Node* n);
    bool get_duplicates(array_list& vec, Node*& first, Node*& second);
};


#endif //LAB3_BINTREE_H
```

Название файла: mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QStringList>
#include <QMessageBox>
#include <iostream>
#include <vector>
#include <sstream>
#include <QGraphicsScene>
#include <QGraphicsTextItem>
#include <QColor>
#include <QFileDialog>
#include <QTextStream>
#include <cmath>
#include "binTree.h"
#include "help.h"

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
    void visualize();
    void draw(Node *n, int maxdepth, int depth = 0, int x = 0, int y = 0);

private slots:
    void on_pushButton_clicked();
```

```
    void on_action_triggered();

    void on_stepButton_clicked();

    void on_fileButton_clicked();

private:
    binTree* tree;
    Ui::MainWindow *ui;
    QGraphicsScene *mainGraphicsScene;
    QPen pen;

    Node* first;
    Node* second;
    int a;
    int b;
    array_list elems;
    bool started;
};

#endif // MAINWINDOW_H
```

Название файла: array_list.h

```
#ifndef ARRAY_LIST_H
#define ARRAY_LIST_H
#include <string>
#include "node.h"

class array_list
{
private:
    Node** array;
    int capacity;
    int count;
    void resize(int new_capacity);

public:
    array_list(int start_capacity = 4);
    Node*& operator[] (int index);
    Node* at (int index);
    void clean();
    void insert(int index, Node* element);
    Node* remove(int index);

    Node* back();
    void push_back(Node* element);
    Node* pop_back();

    Node* front();
    void push_front(Node* element);
    Node* pop_front();

    int size();
    bool empty();
    ~array_list();
};

#endif // ARRAY_LIST_H
```