

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

**ОТЧЕТ**  
**по лабораторной работе №5**  
**по дисциплине «Алгоритмы и структуры данных»**  
**Тема: Бинарные деревья поиска**

Студент гр. 9381

\_\_\_\_\_

Игнашов В.М.

Преподаватель

\_\_\_\_\_

Фирсов М.А.

Санкт-Петербург

2020

## **Цель работы.**

Научиться работе с бинарными деревьями поиска, обходом их, добавлением элементов.

## **Задание.**

7.

БДП: случайное\* БДП

Действие:  $1+2a$

## **Выполнение работы.**

Для выполнения данной лабораторной работы воспользуемся интегрируемой средой разработки QtCreator для создания графической оболочки программы.

Было реализовано четыре класса – класс узлов Node, класс самого окна программы ProgWindow, класс виджета линий для ввода элементов LineOfAmounts и класс отображения построенного дерева – VisualizeTree.

Для осознания, как должно строится дерево определим, что случайное бинарное дерево поиска – дерево, строящееся по заданному массиву.

### **Node:**

В данном классе присутствуют поля `int a`(сам элемент), `int elemNum`(номер элемента в дереве), `Node* left`(ссылка на левый узел), `Node* right`(ссылка на правый узел), `int depth`(глубина вхождения в дерево), конструктор и 4 метода – добавление нового элемента в дерево, поиск максимальной глубины в дереве, поиск элемента в дереве и удаление дерева.

*Конструктор `Node(int am, int depth, int elemNum)`*

В конструкторе – поля инициализируются переданными аргументами, а левая и правая ссылка инициализируются `nullptr`.

*Метод добавления `void addnew(int x, int num)`*

В качестве аргументов передаются сам элемент и номер нового элемента в дереве. В методе создается временный узел и инициализируется деревом – это необходимо сделать, чтоб не испортить уже созданное дерево. Переходим далее

с условиями поиска в бдп до места в дереве, куда должен быть добавлен элемент и, выделив под него память, создаем его.

*Метод поиска максимальной глубины int maxdepth()*

Обходя дерево, считаем максимальную глубину в левом и правом поддереве. Тем самым в конце рекурсии для главного элемента будем знать наибольшую глубину в дереве.

*Метод поиска элемента void findElem(int elem, int\* count)*

Данный метод необходим для выполнения задания из условия «действие».

В качестве аргумента в метод передается значение искомого элемента, и указатель на адрес, в котором будет храниться количество найденных элементов. Обходя дерево как обычно если элемент соответствует, увеличиваем количество найденных на один.

*Метод удаления дерева void deleteTree()*

Метод используется для освобождения памяти, выделенной под дерево, обходя дерево, с конечных элементов удаляем их, очищая память и добравшись до главного элемента, также его удаляем.

### **LineOfAmounts:**

Данный класс содержит одно приватное поле – int n – количество элементов в дереве, 3 поля для графического отображения – массив линий, их вертикальный лейаут, форма, в которую поместится данный лейаут. Сам же класс наследуется от QScrollArea, что дает возможность вводить больше элементов с удобством.

В данном классе присутствует конструктор, деструктор и метод перерасчета размеров.

*Конструктор LineOfAmounts()*

В конструкторе выделяется память под все необходимые поля(включая максимальное количество линий, ненужные – будут становиться невидимыми для пользователя), создается графическое отображение виджета.

*Деструктор ~LineOfAmounts()*

Очищается выделенная ранее память под класс.

*Метод resize(int k)*

В зависимости от переданного количества должных элементов – становятся видимыми пользователю новые, или скрываются старые. Полю `n` присваивается переданное значение – новое количество элементов.

### *VisualizeTree:*

Данный класс наследуется от `QGraphicsScene` и подразумевает собой графическое представление получившегося дерева. У него есть три приватных поля – максимальное количество элементов, `QGraphicsItem`'ы – `items` отвечают за сами элементы, `lineItems` отвечают за линии между элементами.

Также, у этого класса есть конструктор, деструктор и метод обновления отображения.

#### *Конструктор VisualizeTree()*

В конструкторе данного класса выделяется память под все его поля и все `item`'ы становятся невидимыми для пользователя, пока они не потребуются.

#### *Деструктор ~VisualizeTree()*

В деструктор данного класса очищается память ранее выделенная под поля.

#### *Метод update(Node\* head, int numOfLeaves)*

В качестве аргументов для создания дерева – передается количество элементов и главный элемент дерева.

Для начала – все элементы становятся невидимыми, чтобы избежать моментов, когда новое дерево меньше предыдущего и остаются на сцене ненужные элементы.

Далее нам потребуется структура `pt` для осознания необходимой отрисовки для каждого элемента. Так, нам потребуются координаты предыдущего и нынешнего элемента для отрисовки линии, а также что содержится в элементе, для отображения этого. Поле `made` будет использоваться в функции `findPlace` только для возвращения верного значения.

#### *Функция findPlace(Node\* tmphead, int elemNum, int leftMove, int rightMove, int widthch, bool toLeft)*

В качестве аргументов в нее передается дерево для поиска места элемента, номер искомого элемента, информация о том, насколько надо сдвинуться влево и вправо для получения координаты по X, также, значение изменяемое в геометрической прогрессии с шагом  $\frac{1}{2}$ , чтобы построить более точное изображение дерева и информация, с какой стороны пришел элемент.

В самой функции создается возвращаемый элемент `getpt`, который всю информацию будет совмещать. Рекурсивно движемся по дереву, пока не уткнемся в элемент, номер которого будет совпадать с искомым. Тогда инициализируем все необходимые поля. И возвращаемся.

Тем самым, по итогу мы имеем координаты, по которым надо отрисовать линии и значение, которое соответствует элементу. Все это и отображаем для всех кроме нуля, тк для нуля нет линий сверху.

### **ProgWindow:**

Данный класс необходим для непосредственно самого окна программы. У него присутствуют поля – главный элемент дерева, количество элементов и массив введенных элементов. Также, у него присутствуют поля для графического отображения – кнопки, поля для ввода, метки, и `QGraphicsView` для отображения `VisualizeTree treeScene`. Также, у него присутствуют три слота для связи действий и конструктор.

#### *Конструктор ProgWindow()*

В конструкторе выделяется память под все, необходимые объекты, компонуется виджеты и связываются сигналы и слоты. Так, при вводе любого значения в строку количества значений – будет обрабатываться слот `addArray()`, При нажатии на кнопку генерации дерева – будет обрабатываться слот `createTree()`, а при нажатии на кнопку подсчета количества введенных элементов и добавления нового элемента – слот `findElem()`.

#### *Слот addArray()*

В данном слоте обрабатываются возможные ошибки – ввода не числа, слишком большого числа – вызовом функции `error(typeOfError)` с передачей в

нее соответствующего типа ошибки, описанного в перечислении `enum typeOfError`. И вызывается функция `resize()` у списка входных элементов.

#### *Слот `createTree()`*

В данном слоте также обрабатываются возможные ошибки. Записываются значения в строках в массив значений. Удаляется предыдущее дерево и заново записываются все элементы. Вызывается метод `update` у отображения дерева.

#### *Слот `findElem()`*

В данном слоте также обрабатываются возможные ошибки. После чего выводится в `QLabel`, который содержит количество найденных элементов значение, возвращенное в качестве аргумента из функции `findElem` главного элемента дерева, добавляется элемент в список элементов и заново строится дерево.

В главной функции программы отображается окно `ProgWindow`.

Разработанный программный код см. в приложении А.

Тестирование программы см. в приложении Б.

#### **Выводы.**

Были разобраны основные методы работы с бинарными деревьями поиска, реализована программа, предоставляющая одно из них – случайное, а также возможность найти элементы дерева.

# ПРИЛОЖЕНИЕ А

## ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <QApplication>

#include "progwindow.h"

using namespace std;

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    ProgWindow* pg = new ProgWindow;
    pg->setWindowTitle("Случайные БДП");
    pg->showMaximized();
    return a.exec();
}
```

Название файла: node.h

```
#ifndef NODE_H
#define NODE_H

class Node
{
private:
    int a;
    int elemNum;
    Node* left;
    Node* right;
    int depth;
public:
    Node(int am, int depth, int elemNum);
    void addnew(int x, int num);
    int maxdepth();
    void findElem(int elem, int* count);
    void deleteTree();

    Node* getLeft() {return left;}
    Node* getRight() {return right;}
    int getA() {return a;}
    int getElemNum() {return elemNum;}
    int getDepth() {return depth;}
};

#endif // NODE_H
```

Название файла: node.cpp

```
#include "node.h"
Node::Node(int am, int depth, int elNum)
{
    elemNum=elNum;
    a=am;
    this->depth=depth;
    left=nullptr;
    right=nullptr;
}

void Node::addnew(int x, int num)
{
    Node* tmp = this;
```

```

while(true) {
    if(tmp->a>x) {
        if(tmp->left==nullptr) {
            tmp->left=new Node(x,tmp->depth+1,num);
            break;
        }
        tmp=tmp->left;
    }
    else{
        if(tmp->right==nullptr) {
            tmp->right=new Node(x,tmp->depth+1,num);
            break;
        }
        tmp=tmp->right;
    }
}
}

void Node::findElem(int elem, int* count) {
    if(elem==this->a)
        *count+=1;

    if(this->left!=nullptr)
        left->findElem(elem,count);
    if(this->right!=nullptr)
        right->findElem(elem,count);
}

int Node::maxdepth()
{
    int maxdepth_left=0;
    int maxdepth_right=0;
    if(left!=nullptr)
        maxdepth_left=left->maxdepth();
    if(right!=nullptr)
        maxdepth_right=right->maxdepth();
    if(left==nullptr&&right==nullptr)
        return depth;
    else
        return maxdepth_left>maxdepth_right ? maxdepth_left : maxdepth_right;
}

void Node::deleteTree()
{
    if(left!=nullptr)
        left->deleteTree();
    if(right!=nullptr)
        right->deleteTree();
    delete this;
}

```

Название файла: progwindow.h

```

#ifndef PROGWINDOW_H
#define PROGWINDOW_H

#include <QWidget>
#include <QPushButton>
#include <QHBoxLayout>
#include <QVBoxLayout>
#include <QLineEdit>
#include <QLabel>
#include <QMessageBox>
#include <QDesktopWidget>
#include "visualizetree.h"

const int maxn=100;

```



```

class LineOfAmounts: public QScrollArea{
private:
    int n=0;
public:
    LineOfAmounts ();
    QFrame* fr;

    QVBoxLayout* l;
    QLineEdit* arr[maxn];
    void resize(int k);
    ~LineOfAmounts();
};

```

```

class ProgWindow: public QWidget
{
    Q_OBJECT
private:
    Node* head=nullptr;
    int numOfLEi=0;
    int amounts[maxn];
public:
    ProgWindow();
    QLineEdit* numOfLE;
    LineOfAmounts* loa = nullptr;
    QPushButton* calc;

    QLineEdit* whichElem;
    QPushButton* findElemButton;
    QLabel* howManyElems;

    VisualizeTree* treeScene=nullptr;
    QGraphicsView* view;

public slots:
    void findElem();
    void addArray();
    void createTree();
};

```

```

#endif // PROGWINDOW_H

```

Название файла: progwindow.cpp

```

#include "progwindow.h"

```

```

enum typeOfError{
    WRONGINPUT,
    NOTREE,
    NOINPUT,
    BIGAMOUNT
};

```

```

void error(typeOfError type){
    switch(type){
        case WRONGINPUT:
            QMessageBox::warning(nullptr, "Warning!", "Wrong input!");
            break;
        case NOTREE:
            QMessageBox::warning(nullptr, "Warning!", "No tree found!");
            break;
        case NOINPUT:
            QMessageBox::warning(nullptr, "Warning!", "No input found!");
            break;
        case BIGAMOUNT:
            QMessageBox::warning(nullptr, "Warning!", "You've inputted too huge
amount!");
    }
}

```

```

        break;
    }
}

ProgWindow::ProgWindow()
{
    numOfLE=new QLineEdit();
    loa=new LineOfAmounts;
    whichElem=new QLineEdit;
    findElemButton=new QPushButton("Find and Add");
    howManyElems=new QLabel("Amount:  ");
    calc=new QPushButton("Generate");
    treeScene = new VisualizeTree(maxn);
    view=new QGraphicsView;
    view->setScene(treeScene);

    QHBoxLayout* layh = new QHBoxLayout;
    QLabel *NumberOf = new QLabel("Number of elements:");

    layh->addWidget(NumberOf);
    layh->addWidget(numOfLE);
    layh->addWidget(calc);

    QHBoxLayout* layh2=new QHBoxLayout;
    layh2->addWidget(whichElem);
    layh2->addWidget(findElemButton);
    layh2->addWidget(howManyElems);

    QVBoxLayout* layv=new QVBoxLayout;
    layv->addLayout(layh);
    layv->addWidget(loa);
    layv->addLayout(layh2);

    QWidget* leftThing=new QWidget;
    leftThing->setLayout(layv);
    leftThing->setFixedWidth(275);

    QHBoxLayout* layh3=new QHBoxLayout;
    layh3->addWidget(leftThing);
    layh3->addWidget(view);

    connect(calc,SIGNAL(clicked()),this,SLOT(createTree()));
    connect(findElemButton,SIGNAL(clicked()),this,SLOT(findElem()));
    connect(numOfLE,SIGNAL(textChanged(const QString &)),this,SLOT(addArray()));

    this->setLayout(layh3);
}

void ProgWindow::createTree()
{
    if(numOfLEi==0){
        error(NOINPUT);
        if(head!=nullptr)
            head->deleteTree();
        head=nullptr;
        treeScene->update(head,0);
        return;
    }
    for(int i=0;i<numOfLEi;i++){
        if(!(amounts[i]=loa->arr[i]->text().toInt())&&(loa->arr[i]->text()!="0")){
            error(WRONGINPUT);
            if(head!=nullptr)
                head->deleteTree();

```

```

        head=nullptr;
        treeScene->update(head,0);
        return;
    }
}
if(head!=nullptr)
    head->deleteTree();
head = new Node(amounts[0],0,0);
for(int i=1;i<numOfLEi;i++)
    head->addnew(amounts[i],i);
treeScene->update(head,numOfLEi);
}

void ProgWindow::findElem()
{
    int e;
    int count=0;
    if(whichElem->text()=="")
        return;
    if(head==nullptr){
        error(NOTREE);
        return;
    }
    if(whichElem->text()=="0"){
        e=0;
    }else
        if(!(e=whichElem->text().toInt())){
            error(WRONGINPUT);
            return;
        }
    head->findElem(e,&count);
    howManyElems->setText("Amount: "+QString::number(count));
    head->addnew(e,numOfLEi);
    amounts[numOfLEi]=e;
    loa->arr[numOfLEi]->setVisible(true);
    loa->arr[numOfLEi]->setText(whichElem->text());
    numOfLEi+=1;
    createTree();
}

void ProgWindow::addArray()
{
    for(int i=0;i<maxn;i++){
        loa->arr[i]->setVisible(false);
    }
    if(numOfLE->text()==""){
        loa->resize(0);
        for(int i=0;i<maxn;i++)
            loa->arr[i]->setText("");
        numOfLEi=0;
        return;
    }
    if(numOfLEi=numOfLE->text().toInt()){
        if(numOfLEi>=100){
            error(BIGAMOUNT);
            return;
        }
        loa->resize(numOfLEi);
    }else{
        error(WRONGINPUT);
        numOfLE->setText("");
    }
}

LineOfAmounts::LineOfAmounts()
{

```

```

fr=new QFrame;
l = new QVBoxLayout;
for(int i=0;i<maxn;i++){
    arr[i]=new QLineEdit;
    arr[i]->setVisible(false);
    arr[i]->setFixedHeight(20);
    l->addWidget(arr[i]);
}
setWidgetResizable(true);
l->setAlignment(Qt::AlignTop);
fr->setLayout(l);

setWidget(fr);
}

void LineOfAmounts::resize(int k)
{
    if(k<n){
        for(int i=k;i<n;i++){
            arr[i]->setVisible(false);
        }
    }else{
        for(int i=n;i<k;i++){
            arr[i]->setVisible(true);
        }
    }
    this->n=k;
}

LineOfAmounts::~LineOfAmounts()
{
    for(int i=0;i<maxn;i++){
        delete arr[i];
    }
}

```

Название файла: visualizetree.h

```

#ifndef VISUALIZETREE_H
#define VISUALIZETREE_H
#include <QGraphicsScene>
#include <QGraphicsView>
#include <QGraphicsItem>
#include "node.h"

class VisualizeTree: public QGraphicsScene
{
private:
    int maxn;
    QGraphicsSimpleTextItem** items;
    QGraphicsLineItem** lineItems;
public:
    VisualizeTree(int maxn);
    void update(Node* head, int numOfLeaves);
    ~VisualizeTree();
};

#endif // VISUALIZETREE_H

```

Название файла: visualizetree.cpp

```

#include "visualizetree.h"

struct pt{
    int x;
    int y;
}

```

```

    int xlast;
    int ylast;
    int contains;
    bool made=false;
};

pt findPlace(Node* tmphead, int elemNum, int leftMove, int rightMove, int
widthch, bool toLeft){
    pt retpt;
    if(tmphead->getElemNum()==elemNum){
        retpt.x=-leftMove+rightMove;
        retpt.y=tmphead->getDepth();
        retpt.ylast=retpt.y-1;
        if(toLeft){
            retpt.xlast=retpt.x+widthch*2;
        }else{
            retpt.xlast=retpt.x-widthch*2;
        }
        retpt.contains=tmphead->getA();
        retpt.made=true;
        return retpt;
    }
    if(tmphead->getLeft()!=nullptr){
        retpt=findPlace(tmphead->
getLeft(),elemNum,leftMove+widthch,rightMove,widthch/2,true);
        if(retpt.made){
            return retpt;
        }
    }
    if(tmphead->getRight()!=nullptr){
        retpt=findPlace(tmphead->
getRight(),elemNum,leftMove,rightMove+widthch,widthch/2,false);
        if(retpt.made){
            return retpt;
        }
    }
    return retpt;
}

```

```

VisualizeTree::VisualizeTree(int maxn)
{
    this->maxn=maxn;
    items=new QGraphicsSimpleTextItem*[maxn];
    lineItems=new QGraphicsLineItem*[maxn];
    for(int i=0;i<maxn;i++){
        items[i]=new QGraphicsSimpleTextItem;
        lineItems[i]=new QGraphicsLineItem;
        addItem(items[i]);
        addItem(lineItems[i]);
        items[i]->setVisible(false);
        lineItems[i]->setVisible(false);
    }
}

void VisualizeTree::update(Node* head, int numofLeaves)
{
    for(int i=0;i<maxn;i++){
        items[i]->setVisible(false);
        lineItems[i]->setVisible(false);
    }
    for(int i=0;i<numofLeaves;i++){
        pt pl = findPlace(head,i,0,0,pow(2,(head->maxdepth())) ,true);
        items[i]->setText(QString::number(pl.contains));
        items[i]->setPos(pl.x*10,pl.y*10);
        if(i!=0){

```

```

        lineItems[i]->setLine(pl.x*10,pl.y*10,pl.xlast*10,pl.ylast*10);
        lineItems[i]->setPen(Qt::DashLine);
        lineItems[i]->setVisible(true);
    }
    items[i]->setVisible(true);
}

VisualizeTree::~VisualizeTree()
{
    for(int i=0;i<maxn;i++){
        delete items[i];
        delete lineItems[i];
    }
    delete[] items;
    delete[] lineItems;
}

```

Название файла: CurseAiSD.pro

```

QT      += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++11

# You can make your code fail to compile if it uses deprecated APIs.
# In order to do so, uncomment the following line.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs
deprecated before Qt 6.0.0

SOURCES += \
    main.cpp \
    node.cpp \
    progwindow.cpp \
    visualizetree.cpp

HEADERS += \
    node.h \
    progwindow.h \
    visualizetree.h

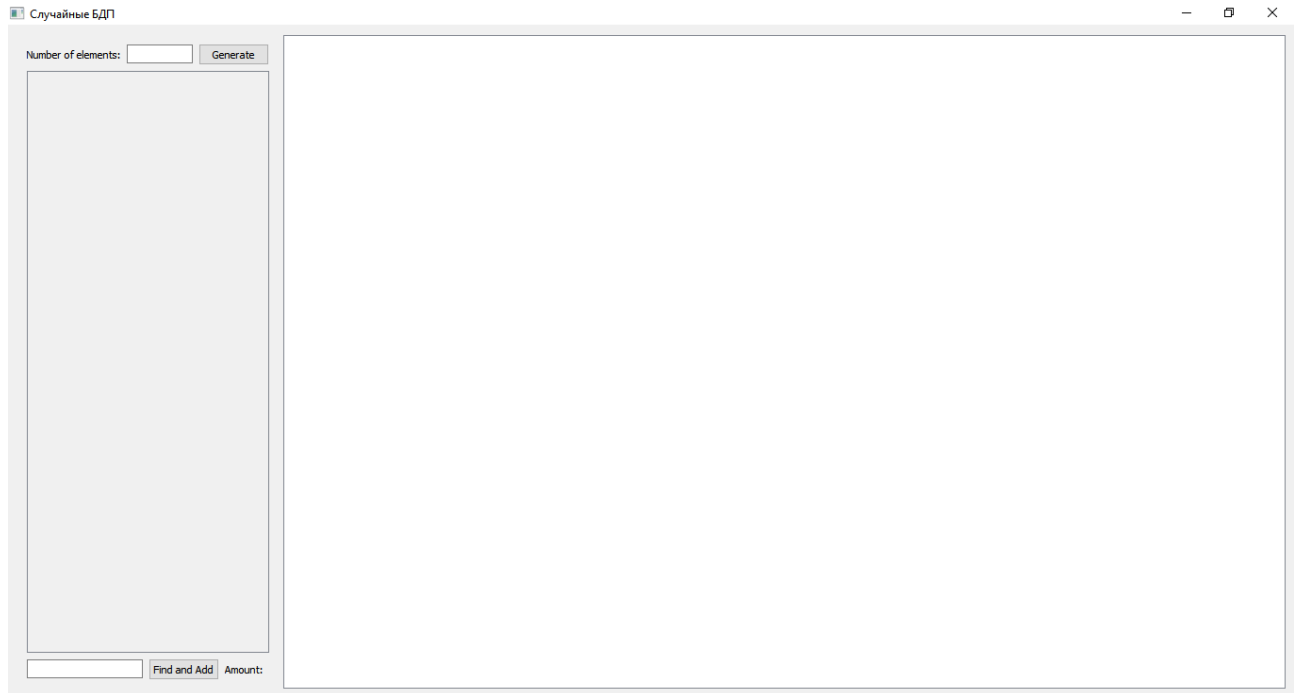
# Default rules for deployment.
qnx: target.path = /tmp/${TARGET}/bin
else: unix:!android: target.path = /opt/${TARGET}/bin
!isEmpty(target.path): INSTALLS += target

```

# ПРИЛОЖЕНИЕ Б

## ТЕСТИРОВАНИЕ

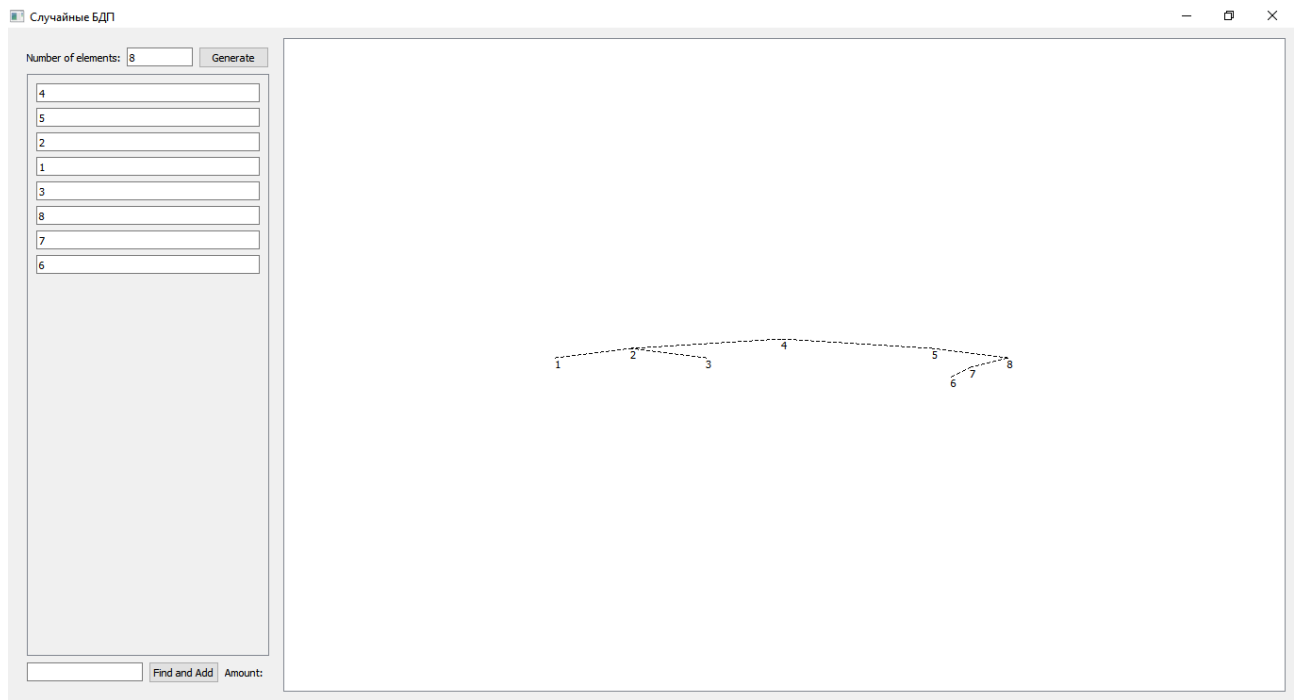
Внешний вид программы при открытии ее.



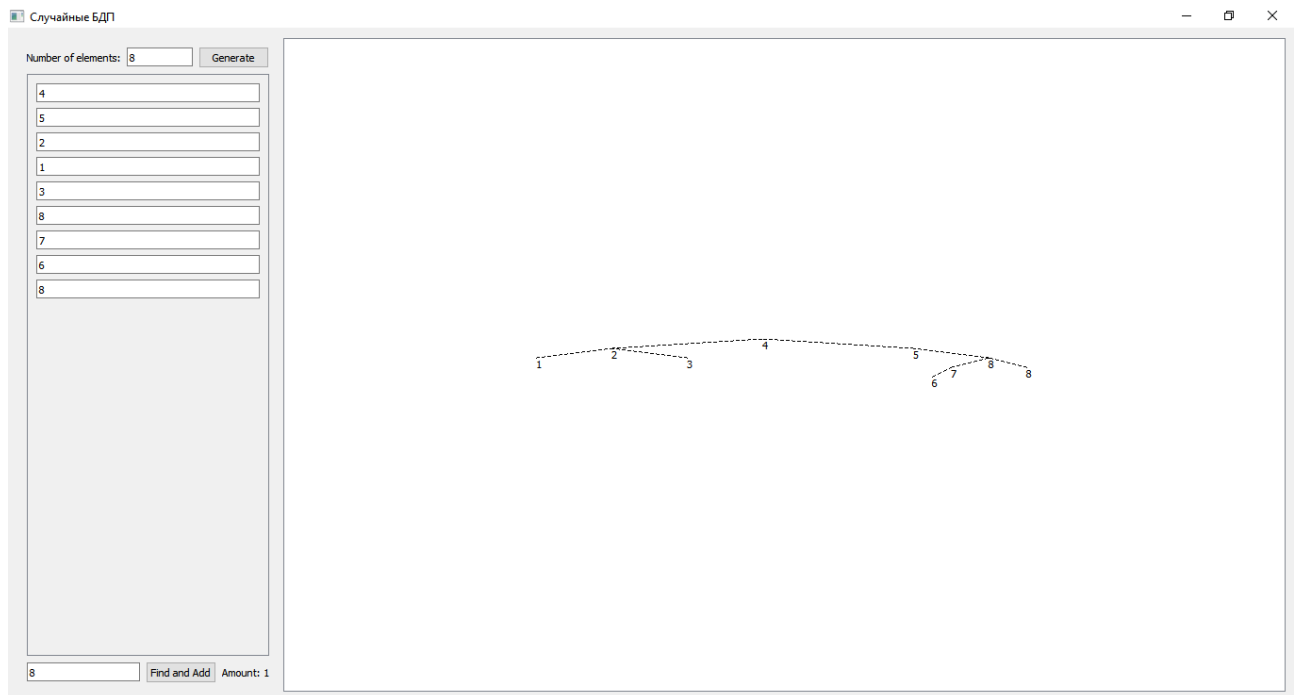
Создание дерева.



Изменение дерева.

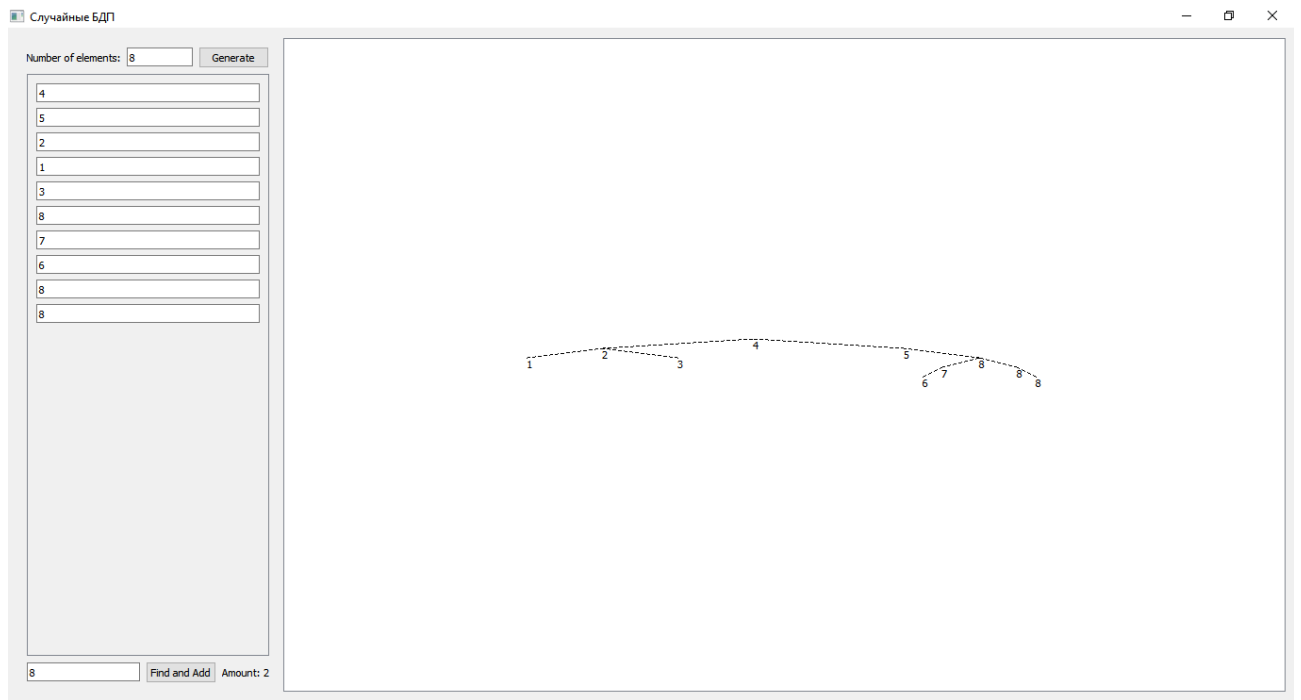


Поиск элемента и его добавление.



Предусмотренные повторные поиски.





Обработанные ошибки:

