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

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: Бинарные деревья поиска

Студент гр. 7383 _____ Ласковенко Е.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2018

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Ласковенко Е.А.		
Группа 7383		
Тема работы: Случайные бинарные деревья поиска с рандомизацией. Демонстрация		
Исходные данные (цель): Реализовать построение случайного бинарного дерева поиска с рандомизацией.		
Содержание пояснительной записки: 1. Содержание 2. Введение 3. Первый раздел: формулировка задачи 4. Второй раздел: решение задачи 5. Третий раздел: тестирование 6. Заключение 7. Приложение А. Примеры работы программы 8. Приложение Б. Исходный код Предполагаемый объем пояснительной записки: не менее 15 страниц.		
Дата выдачи задания:		
Дата сдачи реферата:		
Дата защиты реферата:		
Студент		Ласковенко Е.А.
Преподаватель		Размочаева Н.В.

АННОТАЦИЯ

В данной курсовой работе представлена программа для работы с бинарными деревьями поиска, реализованная в виде оконного графического приложения с использованием C++ фреймворка Qt. Программа содержит: функции для добавления элемента в дерево, функцию создания произвольного бинарного дерева поиска по заданному количеству вершин, а также функцию рисования существующего бинарного дерева. В данной курсовой работе представлены: полный исходный код программы, описание работы представленных функций и примеры работы программы: примеры вставки и удаления элемента, создания бинарного дерева.

SUMMARY

This course work presents a program for working with binary search trees, implemented as a windowed graphical application using the C ++ Qt framework. The program contains: functions for adding an element to the tree, a function for creating an arbitrary binary search tree for a given number of vertices, and also a function for drawing an existing binary tree. In the course work are presented: the full source code of the program, description of the functions presented and examples of the program: examples of inserting and deleting an element, creating a binary tree.

СОДЕРЖАНИЕ

	Введение	5
1.	Формулировка задачи	6
2.	Решение задачи	7
2.1	Класс BST	7
2.1.1	Интерфейс класса BST	7
2.1.2	Реализация класса BST	7
2.2	Класс MainWindow	9
2.2.1	Интерфейс класса MainWindow	9
2.2.2	Реализация класса MainWindow	9
3.	Тестирование	11
	Заключение	12
	Приложение А. Примеры работы программы	13
	Приложение Б. Исходный код	17

ВВЕДЕНИЕ

Целью работы является практическое освоение алгоритма построения случайного бинарного дерева поиска с рандомизацией.

Задачей является создать программу, которая позволяет по шагам проследить ход построения случайного бинарного дерева поиска с выводом промежуточных результатов.

1. ФОРМУЛИРОВКА ЗАДАЧИ И ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Создать программу для построения случайного бинарного дерева поиска с рандомизацией с представлением промежуточных результатов. Результаты должны быть представлены в удобной для понимания форме.

Случайное бинарное дерево поиска с рандомизацией.

При построении рандомизированного дерева основное исходное положение заключается в том, что любой элемент может с одинаковой вероятностью быть корнем дерева. Причем это справедливо и для поддеревьев рандомизированного дерева. Поэтому при включении нового элемента в дерево, случайным образом выбирается включение элемента в качестве листа, или в качестве корня. Вероятность появления нового узла в корне дерева или поддерева определяется, как 1/(n+1), где n – число узлов в дереве или поддереве. То есть дерево выглядит так, будто элементы поступали в случайном порядке. Поскольку принимаемые алгоритмом решения являются случайными, то при каждом выполнении алгоритма при одной и то же последовательности включений будут получаться деревья различной конфигурации. Таким образом, несмотря на увеличение трудоемкости операции включений, за счет рандомизации получается структура дерева, близкая к сбалансированной.

2. РЕШЕНИЕ ЗАДАЧИ

2.1. Класс BST

2.1.1. Интерфейс класса BST:

В классе были обьявлены: конструктор, которые инициализирует все поля объекта, методы, возвращающие значения полей, метод, инициализирующий поля объекта переданными данными, метод, инициализирующий вес объекта, метод нахождения элемента по ключу, метод, добавляющий элемент в качестве сына к текущему объекту и деструктор. Были заведены следующие поля: содержимое символьного типа, два указателя на правое и левое поддеревья, поле ключа целочисленного типа, поле веса целочисленного типа.

2.1.2. Реализация класса BST:

2.1.2.1. Конструктор и деструктор

Были описаны конструктор и деструктор. В конструкторе происходит инициализация полей объекта класса, а деструкторе – удаление дерева с использованием рекурсивного обхода.

2.1.2.2. Метод rebuild

Данный метод заменяет корень дерева для случая, когда это необходимо. Применяется в публичных методах для изменения структуры дерева (поворот налево\направо).

2.1.2.3. Метод add_rand

Данный метод добавляет (вставляет) новый узел в дерево, предварительно подыскав для него место, чтобы после добавления узла дерево по-прежнему оставалось случайным деревом бинарного поиска. Применяет рандомизацию для вставки элемента в корень текущего поддерева.

2.1.2.4. Метод find

Данный метод получает в качестве аргумента ключ искомого элемента, и, найдя элемент в данном дереве, хранимая в котором величина ключа совпадает с полученной в качестве аргумента, возвращает TRUE. Если такой элемент не найден, функция возвращает FALSE.

2.1.2.5. Методы value, key, size, left, right

Данные методы возвращают значения поля текущего объекта: значения узла, ключа, веса, указатель на левое и правое поддерево соответственно.

2.1.2.6. Методы set_lf, set_rt, set_vl

Данные методы получают в качестве аргумента значения полей, которыми инициализируют поля текущего объекта: указатель на левое/правое поддерево, значение узла.

2.1.2.7. Метод fix_size

Данный метод пересчитывает веса всех элеменов поддерева путем рекурсивного ЛКП обхода по всему бинарному дереву.

2.2Класс MainWindow

2.2.1. Интерфейс класса MainWindow

Класс унаследован от класса QMainWindow. Были подключены библиотеки <QGraphicsScene>, <QMainWindow>. Были объявлены: конструкторы и деструктор, методы drawTree(), drawCircle(), addItemOnScene(), setPointsOfChildren(), setPaintStyle, а также слоты, привязанные к различным кнопкам меню.

2.2.2. Реализация класса MainWindow

2.2.2.1. Конструктор и деструктор

В конструкторе выделяется память под сцену, которая заполняется белым цветом, а также устанавливаются характеристики виджетов, которые содержатся в главном окне. Деструктор же освобождает выделенную в конструкторе под соответсвующие поля класса память.

2.2.2.2. Метод setPointsOfChildren

Данный метод устанавливает высчитанные координаты для сыновей текущего узла в узлы этих самых сыновей, переданных по ссылке.

2.2.2.3. Метод setPaintStyle

Данный метод устанавливает цвета рисования и заливки в переданные по ссылке аргументы.

2.2.2.4. Метод drawTree

Данный рекурсивный метод отрисовывает дерево, выводя в узлах значения элементов.

2.2.2.5. Метод drawCircle

Данный метод отрисовывает окружность с центром в переданной точке.

2.2.2.6. Метод addItemOnScene

Данный метод добавляет текстовое поле, содержащее значение переданного узла, в точку на сцене.

2.2.2.7. Слот on_pushButton_clicked

В данном слоте происходит добавление одного элемента в бинарное дерево поиска — выполнение одного шага алгоритма построения. Затем измененное дерево выводится в специальное поле.

2.2.2.8. Слот on_pushButton_2_clicked

В данном слоте происходит добавление всех оставшихся элементов в бинарное дерево поиска — выполнение всех оставшихся шагов алгоритма построения. Затем измененное дерево выводится в специальное поле.

2.2.2.9. Слот on_pushButton_3_clicked

В данном слоте происходит откат дерева до предыдущего состояния — выполнение шага назад. Затем измененное дерево выводится в специальное поле.

2.2.2.10. Слот on_pushButton_4_clicked

В данном слоте происходит очистка поля отрисовки дерева, сброс всех переменных, необходимых для построения дерева.

2.2.2.11. Слот on_lineEdit_returnPressed

В данном слоте происходит считывание строки, введенной пользователем и подготовка всех необходимых средств для дальнейшего построения случайного бинарного дерева поиска с рандомизацией.

3. ТЕСТИРОВАНИЕ

Процесс тестирования

Программа собрана в операционной системе Linux Mint 19, с использованием компилятора G++. В других ОС и компиляторах тестирование не проводилось.

Результаты тестирования

Тестовые случаи представлены в Приложении А.

По результатам тестирования было показано, что поставленная задача была выполнена.

ЗАКЛЮЧЕНИЕ

Была написана программа для работы со случайными бинарными деревьями поиска с рандомизацией, реализованная в виде оконного графического приложения с использованием C++ фреймворка Qt. Программа позволяет пользователю создавать бинарное дерево из указанного количества элементов, хранящих произвольные символы, а также по шагам просмотреть ход работы алгоритма построения.

ПРИЛОЖЕНИЕ А
ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

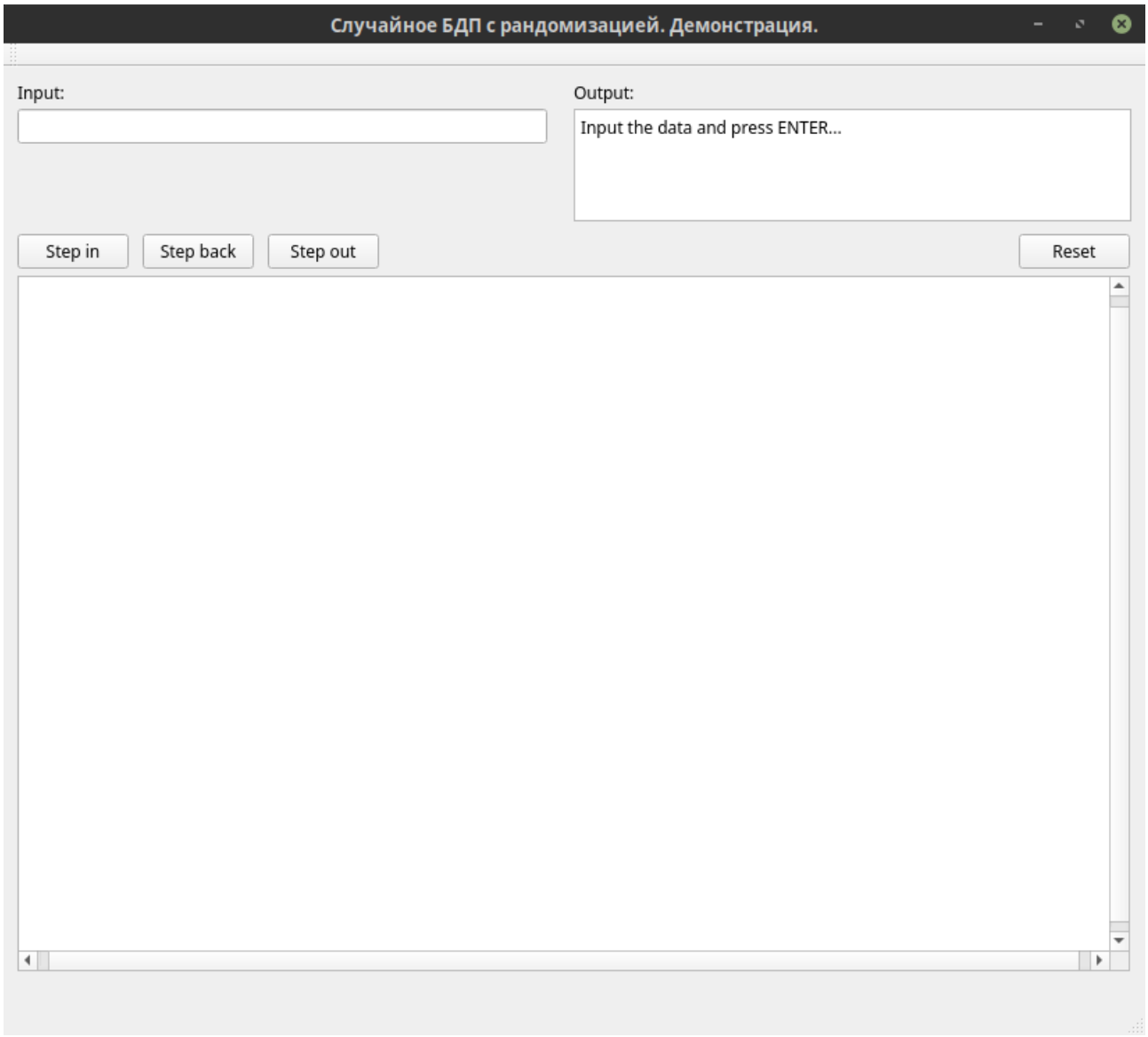

Рисунок 1 – Вид окна для взаимодействия с пользователем

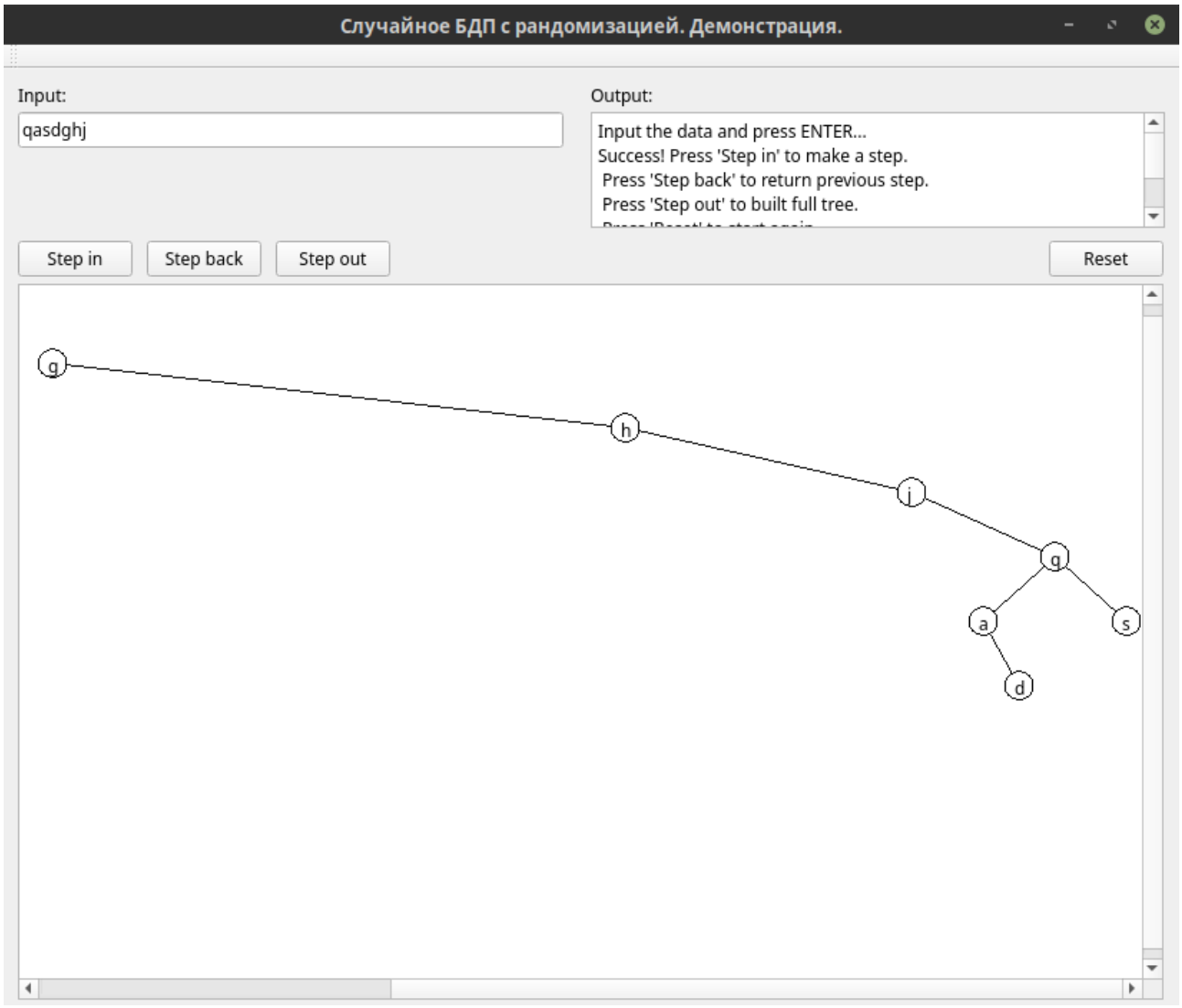

Рисунок 2 – Тестовый случай (step out)

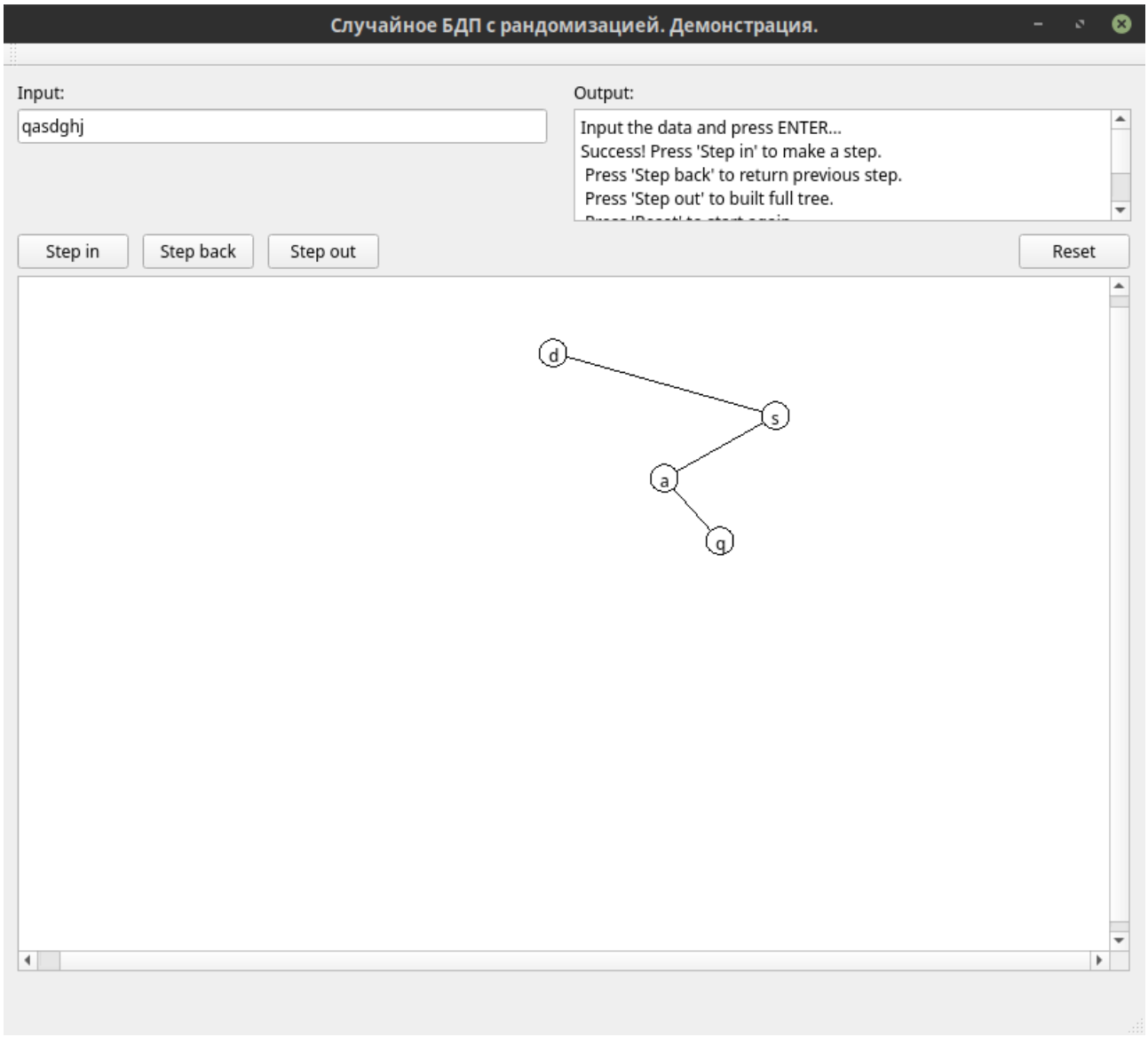

Рисунок 3 – Тестовый случай (step in)

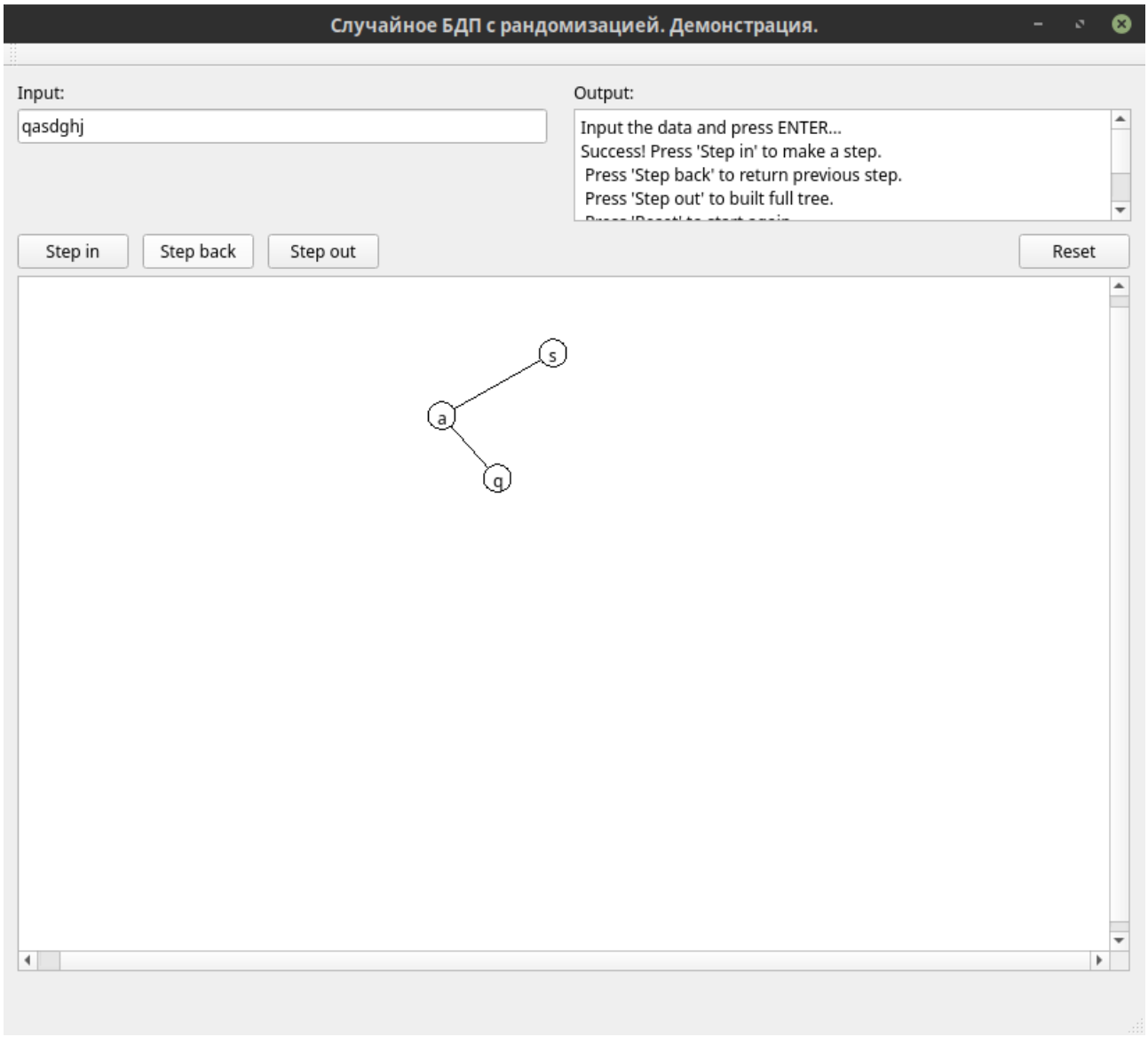

Рисунок 4 – Тестовый случай (step back)

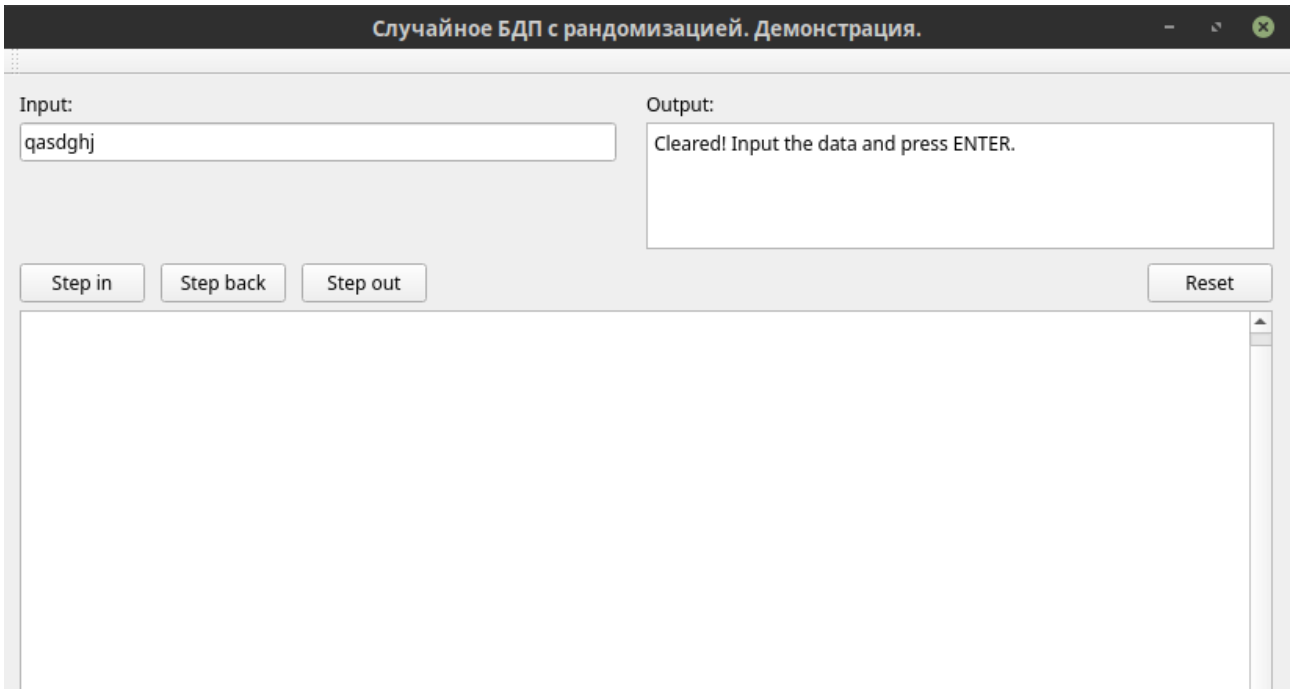

Рисунок 5 – Тестовый случай (reset)

ПРИЛОЖЕНИЕ Б.
ИСХОДНЫЙ КОД

Файл **bst.h:**

```
#include <iostream>
#include <cstdlib>

#ifndef BST_H
#define BST_H

class BST
{
private:
    char val;
    int k;
    int sz;
    BST* lf;
    BST* rt;

    void rebuild(BST* el)
    {
        char tmp_vl = this->value();
        this->set_vl(el->value());
        el->set_vl(tmp_vl);
        el->set_lf(this->left());
        el->set_rt(this->right());
        if(tmp_vl < this->value())
        {
            this->set_lf(el);
            this->set_rt(nullptr);
        }
        else
        {
            this->set_rt(el);
            this->set_lf(nullptr);
        }
    }
public:
    BST()
    {
        val = 0;
        k = -1;
        sz = 1;
        lf = nullptr;
        rt = nullptr;
    }

    char value()
    { return val; }

    int key()
    { return k; }

    int size()
    { return sz; }

    BST* left()
    { return lf; }

    BST* right()
    { return rt; }

    void set_lf(BST* lft)
```

```
{ this->lf = lft; }

void set_rt(BST* rgt)
{ this->rt = rgt; }

void set_vl(char vl)
{
    val = vl;
    srand(val);
    k = std::rand();
}

void fix_size()
{
    this->sz = 1;
    if(this->left())
    {
        this->left()->fix_size();
        this->sz += this->left()->size();
    }
    if(this->right())
    {
        this->right()->fix_size();
        this->sz += this->right()->size();
    }
}

bool find(int key)
{
    bool flag=false;
    if(this)
    {
        if(key == this->key())
            return true;

        flag += this->left()->find(key);
        flag += this->right()->find(key);
    }
    else
        return flag;
}

void add_rand(char val)
{
    BST* bst_el = new BST;
    bst_el->set_vl(val);

    srand(val);
    if(!this->find(rand()))
    {
        this->fix_size();
        srand(time(nullptr));
        if(rand()%(this->size()+1) == this->size())
                this->rebuild(bst_el);
        else
        {
            if(this->value() > bst_el->value())
            {
                if(!this->left())
                    this->set_lf(bst_el);
                this->left()->add_rand(val);
            }
            else
            {
                if(!this->right())
                    this->set_rt(bst_el);
```

```
                    this->right()->add_rand(val);
                }
            }
        }
    }

    ~BST()
    {
        if(this->left()!=nullptr)
            delete lf;
        if(this->right()!=nullptr)
            delete rt;
    }
};

#endif // BST_H
```

Файл **mainwindow.h:**

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QGraphicsScene>
#include <vector>
#include "bst.h"

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);

    void drawTree(QPointF point, BST* root, double offset, QPen pen, QBrush
brush); // отрисовывает дерево на сцене
    void drawCircle(QPointF center, QPen pen, QBrush brush);
    void addItemOnScene(QPointF point, BST* node);
    void setPointsOfChildren(QPointF father, double offset, QPointF &pointLeft,
QPointF &pointRight);
    void setPaintStyle(QPen & pen, QBrush & brush);
    void set_scene();

    BST* copy_bst(BST* root, BST* cp_root);

    ~MainWindow();

private slots:
    void on_pushButton_clicked();
    void on_pushButton_3_clicked();
    void on_lineEdit_returnPressed();
    void on_pushButton_2_clicked();
    void on_pushButton_4_clicked();

private:
    Ui::MainWindow* ui;
    QGraphicsScene* scene;

    QString str;
    std::vector<char> arr;
```

```
    BST* root = new BST;
    std::vector<BST*> arr_pv_rts;
    unsigned counter = 0;
};

#endif // MAINWINDOW_H
```

Файл **main.cpp:**

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

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

Файл **mainwindow.cpp:**

```
#include <ctype.h>
#include <vector>
#include <QGraphicsScene>
#include <QGraphicsTextItem>
#include <cmath>
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "bst.h"

using namespace std;

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    ui->textBrowser->setText(ui->textBrowser->toPlainText() + "Input the data
and press ENTER...\n");
    scene = new QGraphicsScene(this);
    QPixmap pixmap(ui->graphicsView->width(), ui->graphicsView->height());
    pixmap.fill(Qt::white);
    scene->addPixmap(pixmap);
    ui->graphicsView->setScene(scene);
}

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

BST* MainWindow::copy_bst(BST* root, BST* cp_root=nullptr)
{
    if(root)
    {
        cp_root = new BST;
        cp_root->set_vl(root->value());
        if(root->left())
            cp_root->set_lf(copy_bst(root->left()));
        if(root->right())
```

```
            cp_root->set_rt(copy_bst(root->right()));
        cp_root->fix_size();
    }
    return cp_root;
}

void MainWindow::setPointsOfChildren(QPointF father, double offset, QPointF &
pointLeft, QPointF & pointRight)
{
    if(!offset)
        offset=30;
    pointLeft.setX(father.x() - offset*4);
    pointLeft.setY(father.y() + 45);
    pointRight.setX(father.x() + offset*4);
    pointRight.setY(father.y() + 45);
}

void MainWindow::setPaintStyle(QPen & pen, QBrush & brush)
{
    pen.setColor(Qt::black);
    brush.setColor(Qt::white);
    brush.setStyle(Qt::SolidPattern);
}

void MainWindow::drawTree(QPointF point, BST* root, double offset, QPen pen,
QBrush brush)
{
    if (!root) // если вершина пустой лист
        return;

    if (root->left() || root->right()) // если вершина не лист
    {
        QPointF newPointLeft;  // вершина левого поддерева
        QPointF newPointRight; // вершина правого поддерева
        setPointsOfChildren(point, offset, newPointLeft, newPointRight);

        if (root->left() != nullptr) // если левое поддерево не пусто, рисуем
ребро к нему и повторяем
        {
            scene->addLine(point.x(), point.y(), newPointLeft.x(),
newPointLeft.y(), pen);
            drawTree(newPointLeft, root->left(), offset / 2, pen, brush);
        }
        if (root->right() != nullptr) // если правое поддерево не пусто, рисуем
ребро к нему и повторяем
        {
            scene->addLine(point.x(), point.y(), newPointRight.x(),
newPointRight.y(), pen);
            drawTree(newPointRight, root->right(), offset / 2, pen, brush);
        }
        // отрисовка узла окружностью
        drawCircle(point, pen, brush);
        addItemOnScene(point, root);
        return;
    }

    // если вершина лист
    drawCircle(point, pen, brush);
    addItemOnScene(point, root);
}

void MainWindow::drawCircle(QPointF center, QPen pen, QBrush brush)
{
    QRectF rectangle;
    rectangle.setCoords(center.x() + 10, center.y() + 10, center.x() - 10,
center.y() - 10);
```

```
    scene->addEllipse(rectangle, pen, brush);
}

void MainWindow::addItemOnScene(QPointF point, BST* root)
{
    // перерасчет точки для вывода значения узла, т.к. textItem задаст положение
topLeft
    QPointF positionPoint;
    positionPoint.setX(point.x() - 7);
    positionPoint.setY(point.y() - 11);
    // преобразование содержимого узла в строку и добавление на scene
    QString data;
    data.append(root->value());
    QGraphicsTextItem *textItem = new QGraphicsTextItem(data);
    textItem->setPos(positionPoint);
    scene->addItem(textItem);
}

void MainWindow::set_scene()
{
    scene = new QGraphicsScene(this);
    QPixmap pixmap(ui->graphicsView->width(), ui->graphicsView->height());
    pixmap.fill(Qt::white);
    scene->addPixmap(pixmap);
    ui->graphicsView->setScene(scene);

    QPointF startPoint; // точка, откуда начнет выводиться дерево
    startPoint.setX((double)scene->width() / 2);
    startPoint.setY((double)scene->height() / 8);

    int diametr = 10; // диаметр окружности
    int height = root->size(); // высота дерева
    double offset = 2*(height - 2) * diametr;
    // настройка параметров отрисовки
    QPen pen;
    QBrush brush;
    setPaintStyle(pen, brush);

    drawTree(startPoint, root, offset, pen, brush);
}

void MainWindow::on_lineEdit_returnPressed()
{
    if(!arr.size())
    {
        str = ui->lineEdit->text();
        int i=0;
        while(i<str.size())
        {
            if(str[i].toLatin1()!=' ' && str[i].toLatin1()!='\n')
                arr.push_back(str[i].toLatin1());
            i++;
        }
        ui->textBrowser->setText(ui->textBrowser->toPlainText() + "Success!
Press 'Step in' to make a step.\n "
                                                                  + "Press 'Step
back' to return previous step.\n "
                                                                  + "Press 'Step
out' to built full tree.\n "
                                                                  + "Press 'Reset'
to start again.\n");
    }
    else
        ui->textBrowser->setText(ui->textBrowser->toPlainText() + "First press
'Reset'.\n");
}
```

```
void MainWindow::on_pushButton_clicked()
{
    if(!arr.size())
    {
        ui->textBrowser->setText(ui->textBrowser->toPlainText() + "Input first!\
n");
        return;
    }

    if(counter<arr.size())
    {
        arr_pv_rts.push_back(copy_bst(root));
        if(!counter)
            root->set_vl(arr[counter]);
        else
            root->add_rand(arr[counter]);
        root->fix_size();
        counter++;
    }
    else
        ui->textBrowser->setText(ui->textBrowser->toPlainText() + "Building
finished!\n");
    set_scene();

}

void MainWindow::on_pushButton_2_clicked()
{
    if(!arr.size())
    {
        ui->textBrowser->setText(ui->textBrowser->toPlainText() + "Input first!\
n");
        return;
    }
    unsigned i = counter;
    if(i<arr.size())
        arr_pv_rts.push_back(copy_bst(root));

    for(; i<arr.size(); i++)
    {
        if(!i)
            root->set_vl(arr[i]);
        else
            root->add_rand(arr[i]);
    }
    root->fix_size();
    counter = i;
    ui->textBrowser->setText(ui->textBrowser->toPlainText() + "Building
finished!\n");
    set_scene();
}

void MainWindow::on_pushButton_3_clicked()
{
    if(!arr.size())
    {
        ui->textBrowser->setText(ui->textBrowser->toPlainText() + "Input first!\
n");
        return;
    }
    arr.clear();
    str.clear();
    if(root)
        delete root;
    root = new BST;
    counter = 0;
    scene->clear();
```

```
    ui->textBrowser->clear();
    ui->textBrowser->setText(ui->textBrowser->toPlainText() + "Cleared! Input
the data and press ENTER.\n");
}

void MainWindow::on_pushButton_4_clicked()
{
    scene->clear();
    if(!counter)
        return;
    else if(counter==1)
    {
        counter--;
        return;
    }
    else
        counter = arr_pv_rts.back()->size();

    root = arr_pv_rts.back();
    arr_pv_rts.pop_back();
    set_scene();
}
```