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: Статическое кодирование и декодирование

Студент гр. 9303 _____ Молодцев Д.А.

Преподаватель _____ Филатов Ар.Ю.

Санкт-Петербург

2020

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Молодцев Д.А.

Группа 9303

Тема работы: Статическое кодирование и декодирование методами Хаффмана и Фано-Шеннона – текущий контроль

Исходные данные:

Исходные данные генерируются автоматически (например, условие задачи).

Содержание пояснительной записки:

1. Аннотация
2. Введение
3. Основные теоретические положения.
4. Описание кода программы
5. Заключение
6. Список использованных источников.

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания: 06.11.2020

Дата сдачи реферата: 25.12.2020

Дата защиты реферата: 25.12.2020

Студент _____ Молодцев Д.А.

Преподаватель _____ Филатов Ар.Ю.

АННОТАЦИЯ

Курсовая работа представляет собой программу, предназначенную для генерации заданий, связанных с кодированием и декодированием строк методами Хаффмана и Фано-Шеннона. Код программы написан на языке программирования C++, запуск программы подразумевается на операционных системах семейства Windows. При разработке кода программы активно использовались функции стандартных библиотек языка C++, основные управляющие конструкции языка C++. Код был написан по парадигме ООП. Для проверки работоспособности программы проводилось тестирование. Исходный код, скриншоты, показывающие корректную работу программы, и результаты тестирования представлены в приложениях.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы — разработка программы для генерации и проверки задач на кодирование и декодирование латинских строк методами Хаффмана и Фано-Шеннона. Также для наиболее удобного взаимодействия с программой был создан графический интерфейс, с использованием Qt.

Для достижения поставленной цели требуется реализовать следующие задачи:

1. Изучение теоретического материала по написанию кода на языке C++ и об алгоритмах Фано-Шеннона и Хаффмана.

2. Разработка программного кода в рамках полученного задания.

3. Написание программного кода.

4. Тестирование программного кода.

Полученное задание:

Статическое кодирование и декодирование текстового файла методами Хаффмана и Фано-Шеннона – текущий контроль.

1. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

1.1. Основные теоретические положения об алгоритме Хаффмана.

Алгоритм Хаффмана – алгоритм оптимального префиксного кодирования с минимальной избыточностью. Был разработан в 1952 году аспирантом Массачусетского технологического института Дэвидом Хаффманом. Идея алгоритма состоит в следующем: зная вероятности появления символов в сообщении, можно описать процедуру построения кодов переменной длины, состоящих из целого количества битов. Символам с большей вероятностью ставятся в соответствие более короткие коды. Коды Хаффмана обладают свойством префиксности (то есть ни одно кодовое слово не является префиксом другого), что позволяет однозначно их декодировать.

Классический алгоритм Хаффмана на входе получает таблицу частот встречаемости символов в сообщении. Далее на основании этой таблицы строится дерево кодирования Хаффмана (H-дерево).[1]

1. Символы входного алфавита образуют список свободных узлов. Каждый лист имеет вес, который может быть равен либо вероятности, либо количеству вхождений символа в сжимаемое сообщение.
2. Выбираются два свободных узла дерева с наименьшими весами.
3. Создается их родитель с весом, равным их суммарному весу.
4. Родитель добавляется в список свободных узлов, а два его потомка удаляются из этого списка.
5. Одной дуге, выходящей из родителя, ставится в соответствие бит 1, другой — бит 0. Битовые значения ветвей, исходящих от корня, не зависят от весов потомков.
6. Шаги, начиная со второго, повторяются до тех пор, пока в списке свободных узлов не останется только один свободный узел. Он и будет считаться корнем дерева.

1.2. Основные теоретические положения об алгоритме Фано-Шеннона.

Кодирование Шеннона — Фано (англ. Shannon–Fano coding) — алгоритм префиксного неоднородного кодирования. Относится к вероятностным методам сжатия (точнее, методам контекстного моделирования нулевого порядка). Подобно алгоритму Хаффмана, алгоритм Шеннона — Фано использует избыточность сообщения, заключённую в неоднородном распределении частот символов его (первичного) алфавита, то есть заменяет коды более частых символов короткими двоичными последовательностями, а коды более редких символов — более длинными двоичными последовательностями.

Код Шеннона — Фано строится с помощью дерева. Построение этого дерева начинается от корня. Всё множество кодируемых элементов соответствует корню дерева (вершине первого уровня). Оно разбивается на два подмножества с примерно одинаковыми суммарными вероятностями. Эти подмножества соответствуют двум вершинам второго уровня, которые соединяются с корнем. Далее каждое из этих подмножеств разбивается на два подмножества с примерно одинаковыми суммарными вероятностями. Им соответствуют вершины третьего уровня. Если подмножество содержит единственный элемент, то ему соответствует концевая вершина кодового дерева; такое подмножество разбиению не подлежит. Подобным образом поступаем до тех пор, пока не получим все концевые вершины. Ветви кодового дерева размечаем символами 1 и 0, как в случае кода Хаффмана.

1.3. Описание реализованных методов

Было реализовано 2 класса и одна структура, не относящиеся к интерфейсу программы: классы `Facade` и `BinTree` и структура `simbol`. Класс `Facade` является основным классом программы, в нем реализованы методы:

- `BuildFrequencyStr()` – Метод, создающий строку из символов и их частот для одного из типов заданий;
- `SetAnwserInt(int a)` - Метод, запоминающий длину кода искомого символа;
- `GetAnswerInt()` - Метод, «достающий» приватное поле, хранящее длину искомого символа;
- `CountFrequency(std::string message)` - Метод, создающий массив структур `symbol` и считающий частоту встреч каждого символа;
- `CreateTreeByFano(BinTree* tree)` - Метод, создающий дерево методом Фано-Шеннона;
- `CreateTreeByHaffman()`- Метод, создающий дерево методом Хаффмана;
- `CreateHaffmanNodes()`- Метод, создающий вектор из узлов будущего дерева;
- `MergeNodes(BinTree* left,BinTree* right)` - Метод, «присоединяющий» два узла дерево с общему родителю;
- `PrintTree(BinTree* tree)` - Метод, выводящий дерево в консоль;
- `MakeHead()`- Метод, создающий голову бинарного дерева, создаваемого методом Хаффмана;
- `Sort()`- Метод, сортирующий массив структур типа `simbol` по возрастанию частоты встреч каждого символа;
- `CodingByFano()` - Метод, обходящий дерево и заданную строку, тем самым кодируя эту самую строку;
- `Decoding(BinTree* head, BinTree* fict)` - Метод, совершающий обход дерева, тем самым декодируя строку;
- `CodingByHaffman(BinTree* head)` - Метод, кодирующий строку методом Хаффмана;
- `cmp(BinTree* t1, BinTree* t2)` – Метод-компаратор для сортировки вектора;
- `BuildTreeFile()`- Метод, создающий файл с синтактисом, подходящим для работы утилиты graphviz;
- `BuildNode(BinTree* head)` - Метод, рекурсивно обходящий дерево и записывающий в файл данные каждого узла;

Так же были реализованы методы для доступа к приватным полям класса Facade. Сами приватные поля класса:

- task_string – Заданная для кодирования-декодирования;
- simbols – Строка, состоящая из символов, частота которых не 0;
- size – размер массива структур типа `simbol`;
- index – индекс, использующийся при прохождении по символам заданной строки во время процесса декодирования
- arr – массив структур типа `simbol`;
- code_tree_ – указатель на голову дерева для метода Хаффмана;
- Haffman_tree – вектор указателей на узлы бинарного дерева для метода Хаффмана;
- str_answer – ответ для проверки типа «строка»;
- int_answer – ответ для проверки типа «число»;

Класс `BinTree` является классом бинарного дерева, в нем реализованы методы для обращения и изменения следующих приватных полей:

- data – все символы текущего узла;
- sum – сумма частот встреч всех символов данного узла дерева;
- left – указатель на левое поддерево;
- right – указатель на правое поддерево;
- code – код на текущем узле;

Структура `simbol` не имеет методов и имеет следующие публичные поля:

- n – частота встреч конкретного символа;
- c – символ (буква латинского алфавита или запятая, точка, восклицательный или вопросительный знак);
- code – строка с кодом конкретного символа;

2. ОПИСАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

2.1. Описание главного окна программы

Для осуществления взаимодействия с пользователем в ходе курсовой работы был реализован графический интерфейс с использованием фреймворка `Qt`. Ядром интерфейса является главное окно, для описания которого был реализован класс `MainWindow`.

Класс `MainWindow` имеет следующие поля:

`Ui::MainWindow* ui` – форма, которая хранит все элементы главного окна.

`QGraphicsScene* scene` – сцена, на которой отображается задание (картинка дерева для кодирования).

`QGraphicsPixmapItem* item` – сам графический элемент-картинка, который хранит картинку-визуализацию дерева.

Форма содержит кнопки генерации задания, проверки ответа, а также поле для ввода ответа на задание. Также, справа имеется поле для показа условия – картинки с визуализированным сгенерированным деревом кодирования.

Также класс `MainWindow` содержит следующие методы:

`WriteTaskText()` – выводит текст случайно сгенерированного задания в окно приложения Тип генерируемого задания выбирается случайным образом.

`WriteFile()`- выводит в файл вопрос, находящийся на данный момент на экране, а так же правильный ответ;

`VizualizeTree()`– генерирует и визуализирует дерево.

Так же реализованы слоты:

`on_Generate_Task_clicked()` – Слот запускает методы для генерации очередного задания;

`Password()` – Проверяет введенный пароль для доступа к дереву;

`on_CheckButton_clicked()` – Проверяет полученный ответ с верным;

on_pushButton_clicked() – Вызывает диалоговое окно для доступа к дереву;

2.2. Описание вспомогательных окон.

Было реализовано диалоговое окно для ограничения доступа к дереву, так как имея уже построенное дерево, все задачи, предлагаемые программой, теряют какой-либо смысл, становясь слишком простыми.

Диалоговое окно вызывается при нажатии на кнопку «Показать дерево» и запрашивает пароль. При введении верного пароля в правой части приложения будет выведена картинка нужного дерева, построенная с помощью утилиты *graphviz*.

2.3. Описание создаваемого файла.

При генерации задачи, ее условие выводится в приложении и, по требованию задания, в файл QandA.txt. В данный файл поочередно записываются задания, ранее выведенные в приложении, и ответы к каждому из них.

Пример генерируемого файла представлен в приложении Б.

ЗАКЛЮЧЕНИЕ

Для успешного достижения поставленной цели — написания программы для генерации задач, соответствующих заданию курсовой работы, были выполнены соответствующие задачи:

1. Изучен теоретический материал по теме курсовой работы.
2. Разработан программный код.
3. Реализован программный код.
4. Проведено тестирование программы.

Исходный код программы представлен в приложении А, результаты тестирования - в приложении Б.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Язык программирования СИ / Керниган Б., Ритчи Д. СПб.: Издательство "Невский Диалект", 2001. 352 с.

2. Основы программирования на языках Си и С++ [Электронный ресурс URL: http://cplusplus.com

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

MAINWINDOW.H:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "libs.h"
#include "facade.h"
#include "asklog.h"

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow{
    Q_OBJECT
public:
    explicit MainWindow(QWidget *parent = nullptr);
    void WriteTaskText(); //method writes text of task to label
    void WriteFile();    //method writes task and answer to file
    void VizualizeTree();     //method calls dot and pace picture
to QGraphicsScene
    ~MainWindow();
private slots:
    void on_Generate_Task_clicked(); //method generates task
    void Password();     //slot that checks password
    void on_CheckButton_clicked(); //slot checks answer
    void on_pushButton_clicked();  //slot starts dialog window
which asks password
private:
    Ui::MainWindow *ui;  //qt form
    Asklog* ask;    //dialog window
    std::fstream file;   //file which contains questions and
answers
    Facade* facade=nullptr;
    QString task="";     //string of generated task
    QGraphicsScene* scene;    //pointer to Graphics scene
    QGraphicsPixmapItem* item=nullptr; //picture of tree
    int aim_ind=-1;      //integer answer
    int curr_type=0;     //type of generated task
};
#endif // MAINWINDOW_H
```

MainWindow.cpp:

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

```
MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent),ui(new
Ui::MainWindow){
    srand(time(0));
    ui->setupUi(this);
    scene = new QGraphicsScene();
    ui->graphicsView->setScene(scene);
    ui->CheckButton->setEnabled(false);
    connect(ui->Text_of_Answer,&QLineEdit::textChanged,this,[&](){
        ui->CheckButton->setEnabled(!ui->Text_of_Answer-
>text().isEmpty());
    });
}

MainWindow::~MainWindow(){
    if(facade!=nullptr){
        delete facade;
    }
    delete ui;
}

void MainWindow::on_Generate_Task_clicked(){
    QString str="qwertyuiopasdfghjklzxcvbnm,.!?";
    ui->Generate_Task->setEnabled(false);
    int len=rand() % 30 + 10;
    for (int i=0;i<len;i++) {
        int j = rand() % 30;
        task[i]=str[j];
    }
    curr_type=rand() % 6+1;
    facade= new Facade(task.toStdString());
    facade->CountFrequency(facade->GetTaskStr());
    facade->Sort();
    if(curr_type<4){
        facade->MakeHead();
        facade->CreateTreeByFano(facade->GetTree());
        facade->CodingByFano();
        facade->Decoding(facade->GetTree(),facade->GetTree());
    }else{
        facade->CreateHaffmanNodes();
        facade->CreateTreeByHaffman();
        facade->CodingByHaffman(facade->GetHaffman_());
        facade->CodingByFano();
        facade->Decoding(facade->GetHaffman_(),facade->GetHaffman_());
    }
    WriteTaskText();
}

void MainWindow::WriteTaskText(){
    switch (curr_type) {
    case 1:
        ui->Text_of_exersize->setText("Имеется следующее сообщение:\n   "
+ task
                                      +"\nТребуется закодировать его
методом Фано-Шеннона\n"
                                       "и ввести закодированное
сообщение\n"
```

```
                                        "Пробел, запятая, точка, а также
восклицательные \nи вопросительные знаки "
                                        "тоже кодируются вместе с
буквами.");
        break;
    case 2:
        ui->Text_of_exersize->setText("Имеется некоторое сообщение."
                                      "\nТребуется декодировать его
методом Фано-Шеннона\nпо следующей частоте встреч:\n"
                                       + facade->BuildFrequencyStr() +
                                      "Пробел, запятая, точка, а также
восклицательные \nи вопросительные знаки\n"
                                      "тоже кодируются вместе с
буквами.");
        break;
    case 3:
        aim_ind = 30;
        ui->Text_of_exersize->setText("Имеется следующее сообщение:\n   "
+ task
                                      +"\nТребуется закодировать его
методом Фано-Шеннона\n"
                                       "И ввести длину кода символа:\n  "
                                      + QString::fromStdString(facade-
>GetArr()[aim_ind].c) +
                                       "\nПробел, запятая, точка, а также
восклицательные \nи вопросительные знаки "
                                       "тоже кодируются вместе с
буквами.");
        break;
    case 4:
        ui->Text_of_exersize->setText("Имеется следующее сообщение:\n   "
+ task
                                      +"\nТребуется закодировать его
методом Хаффмана\n"
                                       "и ввести закодированное
сообщение\n"
                                       "Пробел, запятая, точка, а также
восклицательные \nи вопросительные знаки "
                                       "тоже кодируются вместе с
буквами.");
        break;
    case 5:
        ui->Text_of_exersize->setText("Имеется некоторое сообщение.\n"
                                      "\nТребуется декодировать его
методом Хаффмана\nпо следующей частоте встреч:\n"
                                       + facade->BuildFrequencyStr() +
                                      "Пробел, запятая, точка, а также
восклицательные \nи вопросительные знаки\n"
                                      "тоже кодируются вместе с
буквами.");
        break;
    case 6:
        aim_ind = 30;
        ui->Text_of_exersize->setText("Имеется следующее сообщение:\n   "
+ task
                                      +"\nТребуется закодировать его
```

```
методом Фано-Шеннона\n"
                                          "И ввести длину кода символа:\n  "
                                         + QString::fromStdString(facade-
>GetArr()[aim_ind].c) +
                                          "\nПробел, запятая, точка, а также
восклицательные \nи вопросительные знаки "
                                          "тоже кодируются вместе с
буквами.");
        break;
    }
}

void MainWindow::WriteFile(){
    file.open("QandA.txt",  std::fstream::in | std::fstream::out |
std::fstream::app);
    file<<ui->Text_of_exersize->text().toStdString();
    file<<"\nAnswer:\n";
    switch (curr_type) {
        case 1:
            file<<facade->coded_str;
            break;
        case 2:
            file<<facade->decoded_str;
            break;
        case 3:
            file<<facade->GetAnswerInt();
            break;
        case 4:
            file<<facade->coded_str;
            break;
        case 5:
            file<<facade->decoded_str;
            break;
        case 6:
            file<<facade->GetAnswerInt();
            break;
    }
    file<<"\n\n";
    file.close();
}

void MainWindow::on_CheckButton_clicked(){
    switch (curr_type) {
    case 1:
        if(ui->Text_of_Answer->text()==QString::fromStdString(facade-
>coded_str)){
            QMessageBox::about(this,"Answer","Your answer is right!");
            ui->Generate_Task->setEnabled(true);
        }else{
            QMessageBox::about(this,"Answer","Your answer is wrong!");
        }
        break;
    case 2:
        if(ui->Text_of_Answer->text()==QString::fromStdString(facade-
>decoded_str)){
            QMessageBox::about(this,"Answer","Your answer is right!");
```

```
            ui->Generate_Task->setEnabled(true);
        }
        else{
            QMessageBox::about(this,"Answer","Your answer is wrong!");
        }
        break;
    case 3:
        if(ui->Text_of_Answer->text().toInt()==facade->GetAnswerInt()){
            QMessageBox::about(this,"Answer","Your answer is right!");
            ui->Generate_Task->setEnabled(true);
        }else{
            QMessageBox::about(this,"Answer","Your answer is wrong!");
        }
        break;
    case 4:
        if(ui->Text_of_Answer->text()==QString::fromStdString(facade-
>coded_str)){
            QMessageBox::about(this,"Answer","Your answer is right!");
            ui->Generate_Task->setEnabled(true);
        }else{
            QMessageBox::about(this,"Answer","Your answer is wrong!");
        }
        break;
    case 5:
        if(ui->Text_of_Answer->text()==QString::fromStdString(facade-
>decoded_str)){
            QMessageBox::about(this,"Answer","Your answer is right!");
            ui->Generate_Task->setEnabled(true);
        }
        else{
            QMessageBox::about(this,"Answer","Your answer is wrong!");
        }
        break;
    case 6:
        if(ui->Text_of_Answer->text().toInt()==facade->GetAnswerInt()){
            QMessageBox::about(this,"Answer","Your answer is right!");
            ui->Generate_Task->setEnabled(true);
        }else{
            QMessageBox::about(this,"Answer","Your answer is wrong!");
        }
        break;
    }
    WriteFile();
}

void MainWindow::on_pushButton_clicked(){
    scene->clear();
    ask = new Asklog();
    ask->show();
    connect(ask, &Asklog::myclose,this,&MainWindow::Password);
}

void MainWindow::Password(){
       if(ask->line->text() == "0990" && ui->Text_of_exersize-
>text()!=""){
            facade->BuildTreeFile();
```

```
            ask->close();
            item= scene->addPixmap(QPixmap("res.png"));
    }else{
        QMessageBox::about(this,"Error!","Wrong password or no generated
trees!\n You cannot see a tree!");
    }
}

BinTree.h:

#ifndef BINTREE_H
#define BINTREE_H
#include "libs.h"

class BinTree  {
    std::string data="";   //string of coded simbols
    int sum=0;        //summ of frequency
    BinTree* left=nullptr; //left tree
    BinTree* right= nullptr;    //right tree
    std::string code="";   //current code
public:
    std::string& Get_Code(){
            return this->code;
        }
        void Set_Code(std::string& str){
            this->code=str;
        }
        std::string& Get_Data(){
            return this->data;
        }
        BinTree(std::string& str){
            this->data=str;
        }
        ~BinTree(){
            if(left!=nullptr){
                delete left;
            }
            if(right!=nullptr){
                delete right;
            }
        }
        void SetSum(int s){
            this->sum=s;
        }
        int GetSum(){
            return this->sum;
        }
        void SetLeft(BinTree* tree){
            this->left=tree;
        }
        void SetRight(BinTree* tree){
            this->right=tree;
        }
        void Create_left(std::string& str) {
            this->left = new BinTree(str);
        }
```

```
        void Create_right(std::string& str){
            this->right = new BinTree(str);
        }
        void Set_Str(std::string& str){
            this->data=str;
        }
        BinTree* Get_left(){
            return this->left;
        }
        BinTree* Get_right(){
            return this->right;
        }
};

#endif // BINTREE_H
```

Façade.h:

```
#ifndef FACADE_H
#define FACADE_H
#include "libs.h"
#include "bintree.h"


struct simbol{  //struct for saving simbols and their frequencies
    int n=0;
    std::string c="";
    std::string code="";
};

class Facade{
    std::string task_string="";     //message that was generated
    std::string simbols="";     //array of simbols which has frequency
equal 1 and more
    int size=31;    //size of simbols array
    int index=0;//index for checking decoding
    simbol* arr;    //array of simbols and their frequencies
    BinTree* code_tree_=nullptr;    //Binary Tree
    std::vector<BinTree*> Haffman_tree; //all nodes of Haffman tree
    std::string str_answer="";  //right answer if it is message
    int int_answer=-1;  //right answer if it is integer
public:
    int file_num=1;
    int file_index=0;
    std::fstream file;
    BinTree* GetHaffman_();     //method returns head of the Haffman tree
    std::string coded_str="";
    std::string decoded_str="";
    int GetSize();
    simbol* GetArr();   //method for getting array of simbols and
    BinTree* GetTree(); //method for getting Binary Tree
    std::string GetTaskStr();   //method for getting message
    std::string GetSimbols();   //method for getting simbols array
    Facade(std::string message);
    ~Facade();
    QString BuildFrequencyStr();
```

```
    void SetAnwserInt(int a);
    int GetAnswerInt();     //method for remembering answer
    void CountFrequency(std::string message);  //method for counting
frequency of every single simbol
    void CreateTreeByFano(BinTree* tree);     //Creating Binary Tree with
Fano-Shennon method
    void CreateTreeByHaffman();   //Creating Binary Tree with Haffman
method
    void CreateHaffmanNodes();      //Creating array of nodes of Haffman
tree
    BinTree* MergeNodes(BinTree* left,BinTree* right);  //merging 2 node
to one
    void PrintTree(BinTree* tree);  //Printing Binary Tree
    void MakeHead();     //Method for making string which will be paled
to head of Binary Tree
    void Sort();    //This method sorts array of simbols by increasing
freauence
    void CodingByFano(); //Coding method
    void Decoding(BinTree* head, BinTree* fict); //Decoding method
    void CodingByHaffman(BinTree* head);
    static bool cmp(BinTree* t1, BinTree* t2);     //comparator for
vector sorting
    void BuildTreeFile();       //method builds file for Graphviz
    void BuildNode(BinTree* head);
};

#endif // FACADE_H
```

Façade.cpp:

```
#include "facade.h"

Facade::Facade(std::string message){
    this->task_string=message;
}

Facade::~Facade(){
    if(code_tree_!=nullptr){
        delete code_tree_;
    }
    if(arr!=nullptr){
        delete [] arr;
    }
}

void Facade::CountFrequency(std::string message){
    transform(message.begin(), message.end(), message.begin(),
::tolower);
    arr = new simbol[size];
        for(int i=0;i<message.length();i++){
            int ind=(char)(message[i])-97;
            if(isalpha(message[i])){
                if(arr[ind].n==0){
                    arr[ind].c=message[i];
                    arr[ind].n++;
                }else{
```

```
                    arr[ind].n++;
                }
            }
                if(message[i]=='.'){
                    arr[26].c=message[i];
                    arr[26].n++;
                }
                if(message[i]==','){
                    arr[27].c=message[i];
                    arr[27].n++;
                }
                if(message[i]=='?'){
                    arr[28].c=message[i];
                    arr[28].n++;
                }
                if(message[i]=='!'){
                    arr[29].c=message[i];
                    arr[29].n++;
                }
                if(message[i]==' '){
                    arr[30].c=message[i];
                    arr[30].n++;
                }
        }
}

void Facade::CreateTreeByFano(BinTree* tree){
    int curr_sum=0;
        int lsum=0;
        std::string ldata="";
        std::string rdata="";
        for(int i=0;i<(int)(tree->Get_Data().length());i++){
            for(int j=30;j>=0;j--){
                if(tree->Get_Data()[i]==arr[j].c[0]){
                    curr_sum+=arr[j].n;
                }
            }
        }
        tree->SetSum(curr_sum);
        for(int i=0;i<tree->Get_Data().length();i++){
            for(int j=0;j<31;j++){
                if(tree->Get_Data()[i]==(arr[j].c[0])){
                    if(lsum+arr[j].n<=curr_sum/2){
                        lsum+=arr[j].n;
                        ldata+=arr[j].c;
                    }else{
                        rdata+=arr[j].c;
                    }
                }
            }
        }
        std::string lcode=tree->Get_Code()+'0';
        std::string rcode=tree->Get_Code()+'1';
        if(ldata.length()>1){
            tree->Create_left(ldata);
            tree->Get_left()->Set_Code(lcode);
```

```
            CreateTreeByFano(tree->Get_left());
        }else if(ldata.length()==1){
            tree->Create_left(ldata);
            tree->Get_left()->Set_Code(lcode);
            tree->Get_left()->SetSum(lsum);
            for(int j=30;j>=0;j--){
                if(arr[j].c[0]==ldata[0]){
                    arr[j].code=lcode;
                }
            }
        }
        if(rdata.length()>1){
            tree->Create_right(rdata);
            tree->Get_right()->Set_Code(rcode);
            CreateTreeByFano(tree->Get_right());
        }else if(rdata.length()==1){
            tree->Create_right(rdata);
            tree->Get_right()->Set_Code(rcode);
            tree->Get_right()->SetSum(curr_sum-lsum);
            for(int j=30;j>=0;j--){
                if(arr[j].c[0]==rdata[0]){
                    arr[j].code=rcode;
                }
            }
        }
}

void Facade::PrintTree(BinTree* tree){
    std::cout<<tree->Get_Data()<<" "<<tree->GetSum()<<" code: "<<tree-
>Get_Code()<<"\n";
    if(tree->Get_left()){
        PrintTree(tree->Get_left());
    } else{
        std::cout<<"#\n";
    }
    if(tree->Get_right()){
        PrintTree(tree->Get_right());
    }else{
        std::cout<<"#\n";
    }
}

void Facade::Sort(){
    int left_index;
        int right_index;
        int left_bord;
        int mid_bord;
        int right_bord;
        for (int i = 1; i < size; i *= 2){
            for (int j = 0; j < size - i; j = j+2*i){
                left_index = 0;
                right_index = 0;
                left_bord = j;
                mid_bord = j + i;
                right_bord = j+2*i;
                if(right_bord >= size){
```

```
                    right_bord = size;
                }
                simbol* sorted_array = new simbol[right_bord -
left_bord];
                while (left_bord + left_index < mid_bord && mid_bord +
right_index < right_bord){
                    if (arr[left_bord + left_index].n < arr[mid_bord +
right_index].n){
                        sorted_array[left_index + right_index].n =
arr[left_bord + left_index].n;
                        sorted_array[left_index + right_index].c =
arr[left_bord + left_index].c;
                        left_index += 1;
                    }
                    else{

sorted_array[left_index+right_index].n=arr[mid_bord+right_index].n;

sorted_array[left_index+right_index].c=arr[mid_bord+right_index].c;
                        right_index += 1;
                    }
                }
                while (left_bord + left_index < mid_bord){
                    sorted_array[left_index + right_index].n =
arr[left_bord + left_index].n;
                    sorted_array[left_index + right_index].c =
arr[left_bord + left_index].c;
                    left_index += 1;
                }
                while (mid_bord + right_index < right_bord){
                    sorted_array[left_index + right_index].n =
arr[mid_bord + right_index].n;
                    sorted_array[left_index + right_index].c =
arr[mid_bord + right_index].c;
                    right_index += 1;
                }
                for (int k = 0; k < left_index + right_index; k++){
                    arr[left_bord + k].n = sorted_array[k].n;
                    arr[left_bord + k].c = sorted_array[k].c;
                }
                delete [] sorted_array;
            }
        }
}

void Facade::MakeHead(){
    simbols="";
    int ind=30;
    while(ind>=0){
        if(arr[ind].n>0){
              simbols+=arr[ind].c[0];
            }else{
                break;
            }
            ind--;
        }
```

```
    code_tree_= new BinTree(simbols);
}

void Facade::CodingByFano(){
        for(int i=0;i<task_string.length();i++){
            for(int j=30;j>=0;j--){
                if(arr[j].c[0]==task_string[i]){
                    coded_str+=arr[j].code;
                    continue;
                }
            }
        }
        SetAnwserInt(arr[30].code.length());
}

void Facade::CodingByHaffman(BinTree* head){
    std::string lcode=head->Get_Code()+"0";
    std::string rcode=head->Get_Code()+"1";
    int curr_sum=0;
    for(int i=0;i<(int)(head->Get_Data().length());i++){
        for(int j=30;j>=0;j--){
            if(head->Get_Data()[i]==arr[j].c[0]){
                curr_sum+=arr[j].n;
            }
        }
    }
    head->SetSum(curr_sum);
    if(head->Get_left() && head->Get_left()->Get_Code()==""){
        head->Get_left()->Set_Code(lcode);
        CodingByHaffman(head->Get_left());
    }
    if(head->Get_right()){
        head->Get_right()->Set_Code(rcode);
        CodingByHaffman(head->Get_right());
    }
    if(head->Get_Data().length()==1){
        for(int j=30;j>=0;j--){
            if(arr[j].c[0]==head->Get_Data()[0]){
                arr[j].code=head->Get_Code();
            }
        }
    }
    SetAnwserInt(arr[30].code.length());
}

void Facade::Decoding(BinTree *head, BinTree *fict){
    if (index <= coded_str.length()) {
            if (fict->Get_Data().length() == 1) {
                decoded_str += fict->Get_Data();
                if(head->Get_Data().length()==1){
                    index++;
                    if(coded_str.length()-index==1){
                        index++;
                    }
                }
                Decoding(head,head);
```

```
            }
            if(coded_str[index]=='0'){
                index++;
                Decoding(head,fict->Get_left());
            }else if(coded_str[index]=='1'){
                index++;
                Decoding(head,fict->Get_right());
            }
        }
}

void Facade::CreateHaffmanNodes(){
    BinTree* fict;
    for (int i=0;i<size;i++) {
        if(arr[i].n>0){
            fict=new BinTree(arr[i].c);
            fict->SetSum(arr[i].n);
            Haffman_tree.push_back(fict);
        }
    }
    std::reverse(Haffman_tree.begin(),Haffman_tree.end());
}

BinTree* Facade::MergeNodes(BinTree *left, BinTree *right){
    std::string str=left->Get_Data()+right->Get_Data();
    BinTree* head= new BinTree(str);

    head->SetLeft(left);
    head->SetRight(right);
    head->SetSum(left->GetSum()+right->GetSum());
    return head;
}

void Facade::CreateTreeByHaffman(){
    BinTree* head;
    while (Haffman_tree.size() >= 2){
         head=MergeNodes(Haffman_tree[Haffman_tree.size()-1],
Haffman_tree[Haffman_tree.size()-2]);
         Haffman_tree.pop_back();
         Haffman_tree.pop_back();
         Haffman_tree.push_back(head);
         std::sort(Haffman_tree.begin(), Haffman_tree.end(), this->cmp);
    }
}

QString Facade::BuildFrequencyStr(){
    QString str="";
    for (int i=0;i<size;i++) {
        if(arr[i].n>0){
            str = str + QString::fromStdString(arr[i].c) + "(" +
QString::number(arr[i].n) + ")\n";
        }
    }
    return str;
}
```

```
void Facade::BuildTreeFile(){
    std::string path="Tree";
    path+=".txt";
    file.open(path, std::fstream::in | std::fstream::out|
std::fstream::trunc);
    file<<"digraph Tree{\n";
    if(code_tree_!=nullptr){
        BuildNode(code_tree_);
    }else if(this->GetHaffman_()!=nullptr){
        BuildNode(Haffman_tree[0]);
    }
    file << "}";
    file.close();
    std::string command = "dot -Tpng " + path;
    command+=" -o res.png";
    system(command.c_str());
    file_num++;
    file_index=0;
}

void Facade::BuildNode(BinTree *head){
    if(head->Get_left()){
        file<<"\""<<head->Get_Data()<< ", ("<<head->GetSum()<<")\"";
        file<<" -> \""<< head->Get_left()->Get_Data()<< ", ("<<head-
>Get_left()->GetSum()<<")\";\n";
        BuildNode(head->Get_left());
    }else{
        file << file_index << " [shape=point];\n";
        file<<"\""<<head->Get_Data()<< ", ("<<head->GetSum()<<")\"";
        file<<" -> "<< file_index <<";\n";
        file_index++;
    }
    if(head->Get_right()){
        file<<"\""<<head->Get_Data()<< ", ("<<head->GetSum()<<")\"";
        file<<" -> \""<< head->Get_right()->Get_Data()<< ", ("<<head-
>Get_right()->GetSum()<<")\";\n";
        BuildNode(head->Get_right());
    }else{
        file << file_index << " [shape=point];\n";
        file<<"\""<<head->Get_Data()<< ", ("<<head->GetSum()<<")\"";
        file<<" -> "<< file_index <<";\n";
        file_index++;
    }
}

void Facade::SetAnwserInt(int a){
    this->int_answer=a;
}

int Facade::GetAnswerInt(){
    return this->int_answer;
}

BinTree* Facade::GetHaffman_(){
    return this->Haffman_tree[0];
}
```

```
bool Facade::cmp(BinTree *t1, BinTree *t2){
    return t1->GetSum() > t2->GetSum();
}

simbol* Facade::GetArr(){
    return this->arr;
}

std::string Facade::GetSimbols(){
    return this->simbols;
}

BinTree* Facade::GetTree(){
    return this->code_tree_;
}

std::string Facade::GetTaskStr(){
    return this->task_string;
}

int Facade::GetSize(){
    return this->size;
}
```

Asklog.h:

```
#ifndef ASKLOG_H
#define ASKLOG_H
#include "libs.h"

class Asklog : public QWidget{
    Q_OBJECT
public:
    explicit  Asklog(QWidget *parent = nullptr);
    QLineEdit* line;
    QLabel* label;
    QPushButton* ok;
signals:
    void myclose();
};

#endif // ASKLOG_H
```

Asklog.cpp:

```
#include "asklog.h"

Asklog::Asklog(QWidget *parent) : QWidget(parent)
{
    QVBoxLayout *ly = new QVBoxLayout();
    setFixedSize(200, 300);
    label = new QLabel("Enter a password:");
    line = new QLineEdit();
    ok= new QPushButton("&Ok",this);
    ly->addWidget(label);
```

```
    ly->addWidget(line);
    ly->addWidget(ok);
    setLayout(ly);
    connect(ok, &QPushButton::clicked, [=](){
            emit myclose();
    });
}
```

libs.h:

```
#ifndef LIBS_H
#define LIBS_H
#include <list>
#include <fstream>
#include <iostream>

#include <QDialog>
#include <QListWidget>
#include <QListWidgetItem>
#include <QLabel>
#include <QPushButton>
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QToolBar>
#include <QStatusBar>
#include <QKeyEvent>
#include <QMessageBox>
#include <QFile>
#include <QLineEdit>
#include <QTextStream>
#include <QCheckBox>
#include <QObject>

#include <QGraphicsView>

#include <QMainWindow>
#include <QGraphicsScene>
#include <QWidget>

#endif // LIBS_H
```

main.cpp:

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

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

    return a.exec();
}
```

ПРИЛОЖЕНИЕ Б

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ ПРОГРАММЫ

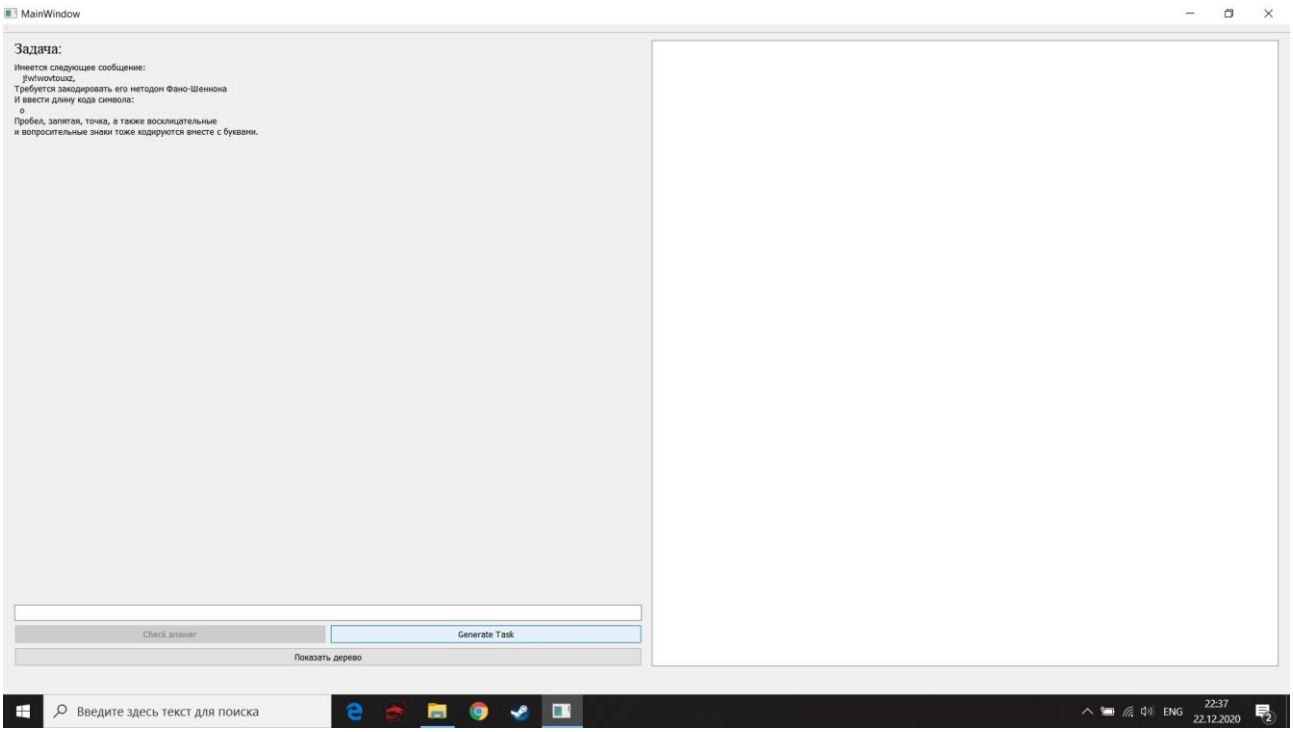

Рисунок 1 – Проверка генерации задания

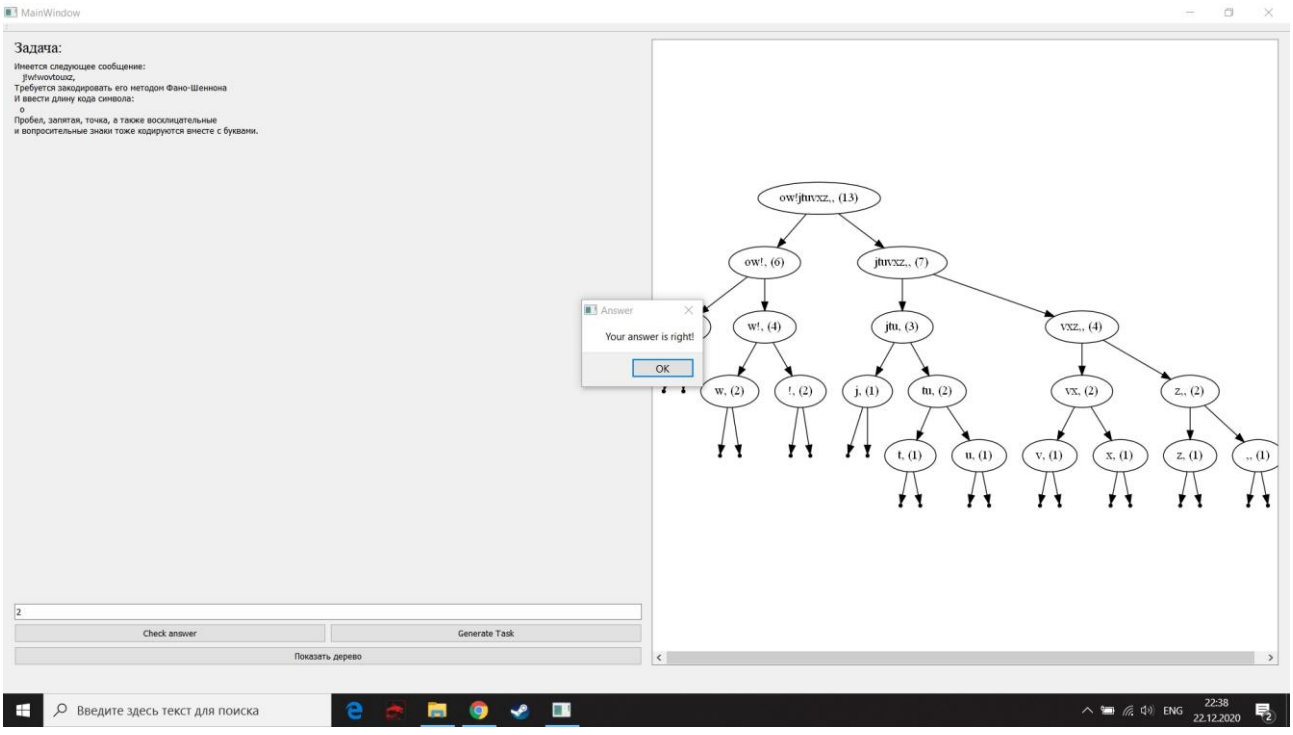

Рисунок 2 – Пример правильного ответа

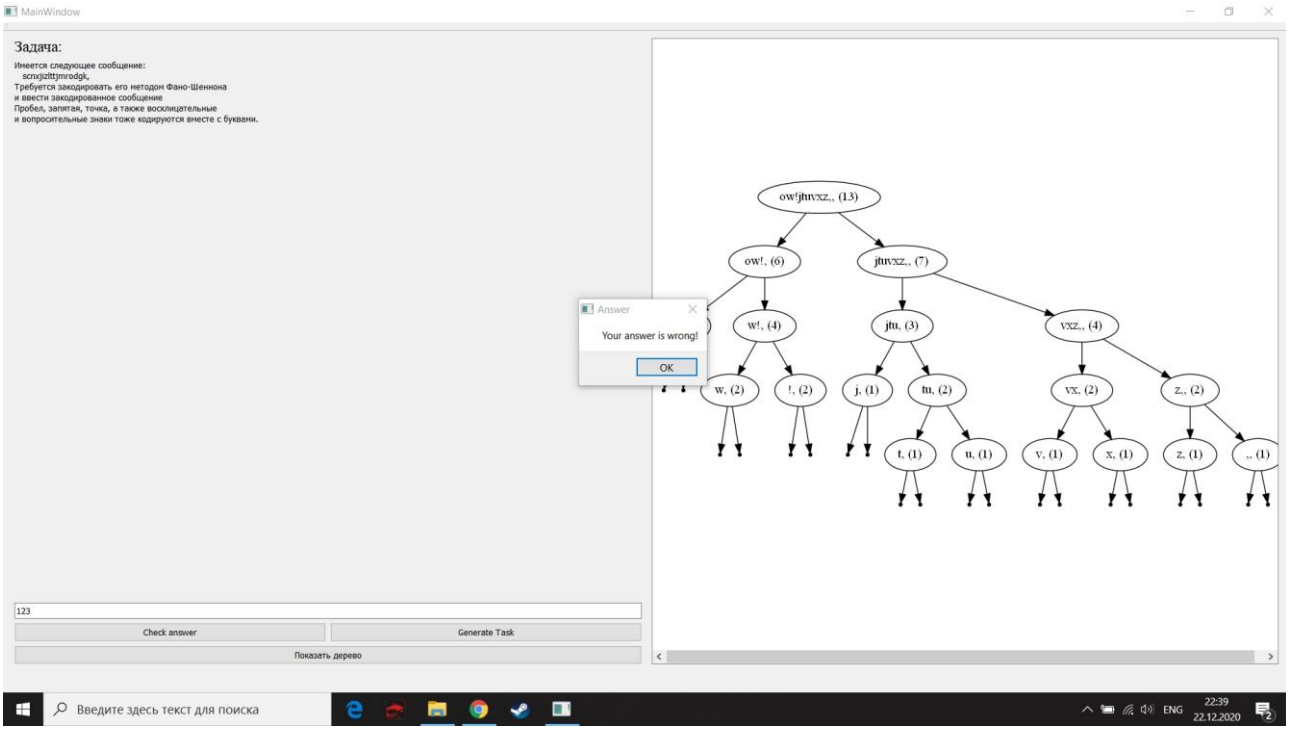

Рисунок 3 –Пример неправильного ответа на задание

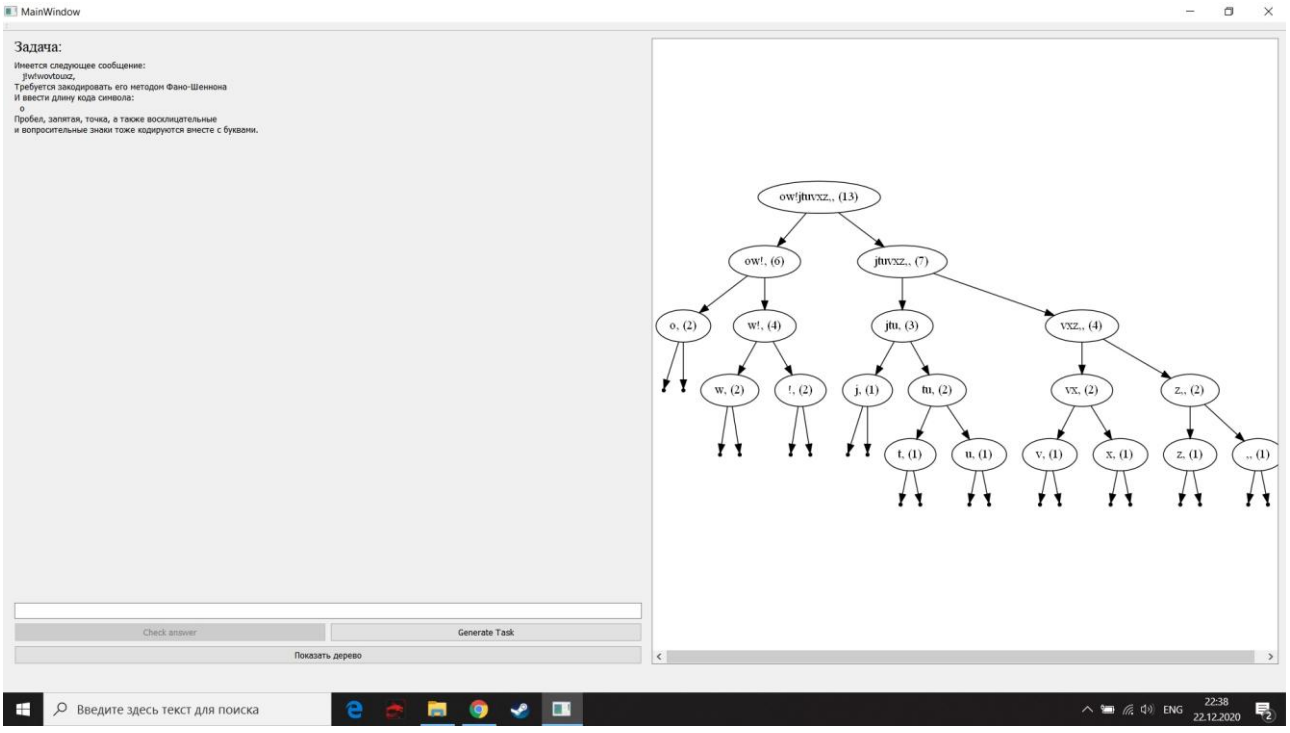

Рисунок 4 – Проверка отрисовки дерева

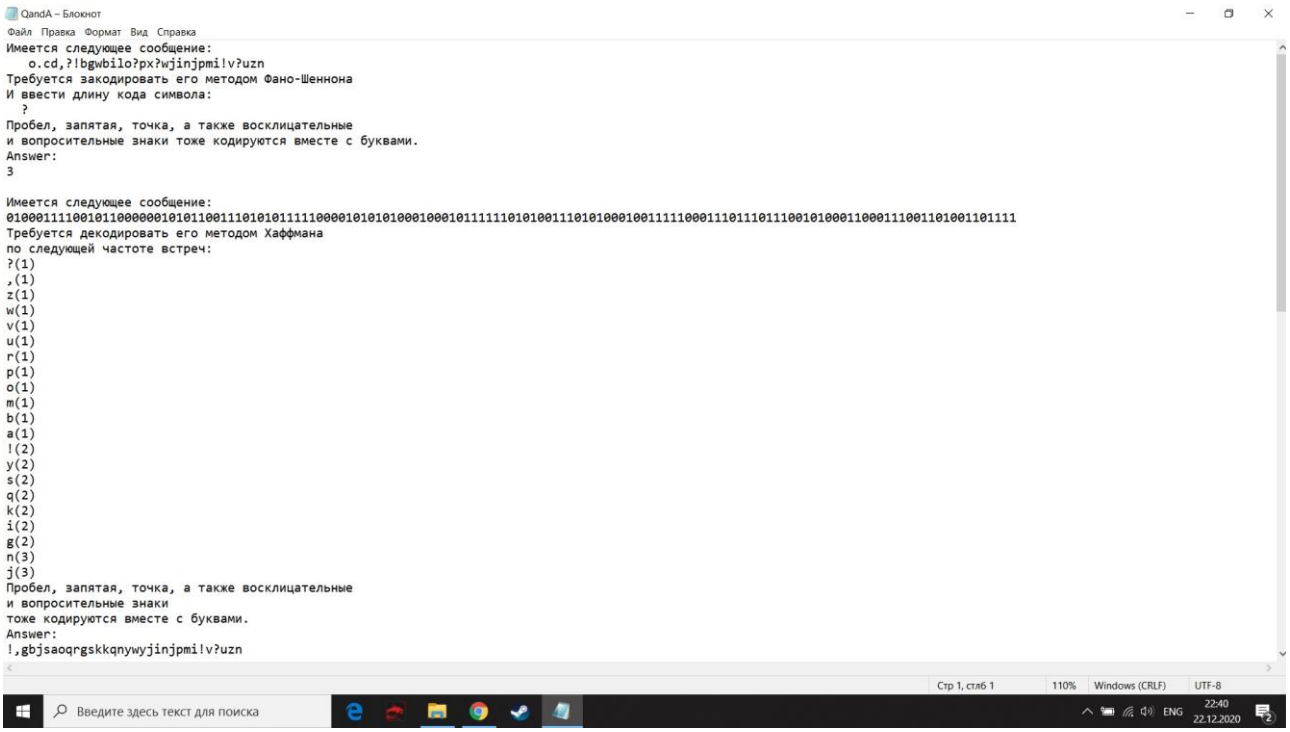

Рисунок 6 – Пример генерируемого файла с условиями и ответами на задачи