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: Рандомизированные дерамиды

Студент гр. 9303 _____ Микулик Д.П.

Преподаватель _____ Филатов Ар.Ю.

Санкт-Петербург
2020

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Микулик Д.П.

Группа 9303

Тема работы: Рандомизированные дерамиды поиска – вставка и исключение (текущий контроль)

Исходные данные:

Исходные данные генерируются автоматически (например, условие задачи).

Содержание пояснительной записки:

Аннотация

Введение

Основные теоретические положения.

Описание кода программы

Заключение

Список использованных источников.

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания: 06.11.2020

Дата сдачи реферата: 18.12.2020

Дата защиты реферата: 18.12.2020

Студент _____ Микулик Д.П.

Преподаватель _____ Филатов Ар.Ю.

АННОТАЦИЯ

Курсовая работа представляет собой программу, предназначенную для генерации заданий, связанных с вставкой и исключением элементов из декартова дерева с обеспечением базового функционала задания. Код программы написан на языке программирования C++, запуск программы подразумевается на операционных системах семейства Linux. 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	5
1.	Основные теоретические положения	6
1.1	Основные теоретические положения о декартовом дереве	6
1.2	Основные теоретические положения о реализованных функциях	6
2.	Описание интерфейса пользователя	7
2.1.	Описание главного окна программы	7
2.2.	Описание вспомогательных окон	8
2.3.	Описание создаваемого файла	8
	Заключение	10
	Список использованных источников	11
	Приложение А. Исходный код программы	12
	Приложение Б. Результаты тестирования программы.	21

ВВЕДЕНИЕ

Цель работы — разработка программы для генерации и проверки задач на вставку и исключение элементов в/из декартова дерева. 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 о работе с декартовыми деревьями.

2. Разработка программного кода в рамках полученного задания.

3. Написание программного кода.

4. Тестирование программного кода.

Полученное задание:

Реализация и экспериментальное машинное исследование рандомизированных дерамид поиска, а именно вставки и исключения элементов. Тип задания – текущий контроль.

1. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

1.1. Основные теоретические положения о декартовом дереве.

Декартово дерево - это структура данных, объединяющая в себе бинарное дерево поиска и бинарную кучу (отсюда и второе её название: treap (tree+heap) и дерамида (дерево+пирамида).

Более строго, это структура данных, которая хранит пары (X, Y) в виде бинарного дерева таким образом, что она является бинарным деревом поиска по x и бинарной пирамидой по y. Предполагая, что все X и все Y являются различными, получаем, что если некоторый элемент дерева содержит (X_0, Y_0), то у всех элементов в левом поддереве $X < X_0$, у всех элементов в правом поддереве $X > X_0$, а также и в левом, и в правом поддереве имеем: $Y < Y_0$.

Дерамиды были предложены Сиделем (Siedel) и Арагон (Aragon) в 1989 г.

1.2. Описание реализованных методов.

Для реализации операций понадобится реализовать две вспомогательные операции: `Split` и `Merge`.

`Split` – разделяет дерево T на два дерева L и R (которые являются возвращаемым значением) таким образом, что L содержит все элементы, меньшие по ключу X, а R содержит все элементы, большие X. Эта операция выполняется за $O(\log N)$. Реализация её довольно проста - очевидная рекурсия.

`Merge` – объединяет два поддерева T_1 и T_2, и возвращает это новое дерево. Эта операция также реализуется за $O(\log N)$. Она работает в предположении, что T_1 и T_2 обладают соответствующим порядком (все значения X в первом меньше значений X во втором). Таким образом, нам нужно объединить их так, чтобы не нарушить порядок по приоритетам Y. Для этого просто выбираем в качестве корня то дерево, у которого Y в корне больше, и рекурсивно вызываем себя от другого дерева и соответствующего сына выбранного дерева.

`Insert` – Разобьём наше дерево по ключу, который мы хотим добавить, то есть $\text{split}(T, k.x) \to \langle T1, T2 \rangle$. Сливаем первое дерево с новым элементом, то

есть $\text{merge}(T1,k) \to T1$. Сливаем получившиеся дерево со вторым, то есть $\text{merge}(T1,T2) \to T$.

`Remove` – Разобьём наше дерево по ключу, который мы хотим удалить, то есть $\text{split}(T,k.x) \to \langle T1,T2 \rangle$. Теперь отделяем от первого дерева элемент x, то есть самого левого ребёнка дерева $T2$. Сливаем первое дерево со вторым, то есть $\text{merge}(T1,T2) \to T$.

`Read` – Функция, генерирующая декартово дерево. Для определенности в дереве не генерируются одинаковые элементы. Функция read возвращает пару ключ-приоритет, эта пара является элементом, который будет удален, в случае генерации задания на удаление элемента из дерева.

`Visualize` – Метод, создающий картинку, визуализирующую текущее построенное дерево. Создает текстовый файл, описывающий структуру дерева, а также .png изображение, используя dot компилятор.

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 декартова дерева).

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

Форма содержит кнопки генерации задания, проверки ответа, выхода из программы, а также поле для ввода ответа на задание. Также, слева имеется

поле для показа условия – картинки с визуализированным сгенерированным декартовым деревом.

Также класс `MainWindow` содержит следующие методы:

`GenerateTask()` – генерирует задание одного из двух типов: на вставку нового элемента в дерево или исключение уже существующего в дереве элемента. Тип генерируемого задания выбирается случайным образом.

`BuildTree()` – генерирует и визуализирует декартово дерево.

`CheckAnswer()` – проверяет правильность введенного ответа. В случае правильного ответа, визуализирует дерево, являющееся ответом.

Для однозначности вводимого ответа дерево генерируется содержащим всегда различные элементы.

2.2. Описание вспомогательных окон.

Были реализованы два вспомогательных окна, которые показывают результат сравнения введенного ответа с правильным. Вызываются при нажатии на кнопку Check Answer. Также стоит отметить, что для корректной работы программы требуется предустановка GraphViz на ПК.

Для запуска на Ubuntu был создан bash-скрипт Coursework.sh, выполняющий предустановку покаетов Graphviz, если они отсутствуют, и выполняет запуск программы. Для корректного запуска приложения требуется наличие некоторых библиотек и плагинов Qt, поэтому папка с проектом их содержит.

2.3. Описание создаваемого файла.

При генерации задачи, ее условие выводится в приложении и, по требованию задания, в файл conditions.txt. Данный файл имеет следующую структуру:

Первый блок каждого задания содержит визуальное представление дерева, в виде уступчатого списка, который идет после строки «Дано следующее дерево». После выведенного дерева начинается второй блок: блок

условия задачи. В нем выводится сам задание, например, информация о том, какой именно элемент вставляется в дерево. Третий блок, наличие которого подразумевает каждое задание – блок правильного ответа. Здесь ответ содержится в удобной для вставки в проверяющее поле форме (т. е. Одна строка, в которой без пробелов идут сначала все вершины первого уровня, затем второго и так далее). Пример генерируемого файла смотреть 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 ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

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

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

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

MAINWINDOW.H:

```
#IFNDEF MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QGraphicsScene>
#include <QMessageBox>
#include <ctime>
#include "treap.h"

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    void BuildTree();
    ~MainWindow();
public slots:
    void GenerateTask();
    void CheckAnswer();

private:
    Ui::MainWindow *ui;
    QGraphicsScene* scene;
    QGraphicsPixmapItem* item;
    Treap* tree = nullptr;
    int n = 0;
    int prior = 0;
};
#endif // MAINWINDOW_H
```

TREAP.H:

```
#ifndef TREAP_H

#define TREAP_H

#include <iostream>
#include <string>
#include <cstdlib>
#include <memory>
#include <fstream>
#include <stack>
#include <algorithm>
#include <sstream>
#include <vector>
```

```
using namespace std;

/* Mikulik D.P., Course work
 * Variant 14
 * Type of the task: current control (including and excluding elements for the
Cartesian tree data structure).
 * December 2020.
 * All comments are written in English, because UTF-8 encoding is maintained not
in all text editors or IDE's.
 */

/* Class Node
 * Node class is a basic class for each elem in the treap.
 * It contains key value, priority of the current elem, size of a treap, and
pointers on left/right "sons" of the element.
 * Its constructor initializes new elem with random priority.
 * There were used smart pointers to avoid using complex destructor.
 */

class Node{
    public:
        int key;
        int prior;
        int size;
        shared_ptr<Node> left;
        shared_ptr<Node> right;
        Node(int key){
            this->key = key;
            left = right = nullptr;
            this->prior = rand()%100;
            size = 1;
        }
        Node(){}
};

/*
 * Pair defined here is a std::pair object, containing two Nodes.
 */

typedef pair<shared_ptr<Node>, shared_ptr<Node>> Pair;

/* Class Treap.
 * Class Treap is a base class used in the coursework.
 * It storages a pointer on root of the treap (data).
 * Also this class provides all basic methods to interact with the data
structure.
 */
class Treap{
protected:
    shared_ptr<Node> data = nullptr;
public:
    int size;
    /* Merge method.
     * Merge method is one of the fundamental methods of the Cartesian tree data
structure.
     * It merges two Cartesian trees.
     */
    shared_ptr<Node> merge(shared_ptr<Node> left, shared_ptr<Node> right);

    /*
```

```
     * Split method.
     * Split method is one of the fundamental methods of the Cartesian tree data
structure.
     * It splits one Cartesian tree into two ones by the given x key value.
     */
    Pair split(shared_ptr<Node> p, int x);

    /* Insert method.
     * Is one of two methods required by the task of the course work.
     * Inserts a new element of the given x value into the treap .
     */
    void insert(int x);

    /* Remove method.
     * Is the second required method.
     * Removes an element of the given key value from the treap .
     */
    void remove(int key);

    /* Dispose method.
     * Deletes the node that was given and its "sons".
     */
    void dispose(shared_ptr<Node> node);

    /* Print method.
     * Is a wrapper of the PrintInOrderTraversal method.
     * Prints the tree to the console.
     */
    void PrintInOrderTraversal(shared_ptr<Node> node, int k);
    void print();

    /* Visualize method
     * The main method of the 5th lab, creates .txt file with some dependencies.
     * After creating a file a dot compiler is called to process a .png file
containing the tree.
     */
    void visualize();

    /* Read method.
     * Reads data from the file of given name.
     * Creates a Cartesian tree using the given key values.
     */
    std::pair<int, int> read();
    int findElem(int x);
    int find(shared_ptr<Node> node, int x);
    std::string correctAnswer();

};

#endif // TREAP_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();
}
```

MAINWINDOW.CPP:

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

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    scene = new QGraphicsScene;
    connect(ui->exitButton, SIGNAL(clicked()), this, SLOT(close()));
    connect(ui->checkButton, SIGNAL(clicked()), this, SLOT(GenerateTask()));
    connect(ui->checkAnswerButton, SIGNAL(clicked()), this,
SLOT(CheckAnswer()));
}

void MainWindow::GenerateTask(){
    BuildTree();
    QString line("Задача: \n");
    int type = rand()%2;
    if (type){
        line += "Пусть дано некоторое декартово дерево.\n";
        line += "Из заданного дерева удаляется элемент ";
        line += QString::number(n);
        line += ".\n";
        line += "Требуется вывести новое дерево, полученное в результате
удаления из него элемента ";
        line += QString::number(n);
        line += ".\n";
        line += "Дерево вводится по следующему принципу:\nсначала вводится номер
вершины-корня,\n затем поочередно вершины сначала 2 уровня, потом третьего и
т.д.\n";
        line += "Номера вершин одного уровня выводятся в порядке слева направо.\
n";
        line += "В любом случае результатом будет строка,\n которая описывает
получившееся бинарное дерево.\n";
        tree->remove(n);
    }
    else{
        line += "Пусть дано некоторое декартово дерево.\n";
        line += "В заданное дерево вставляется элемент ";
        n = rand()%(tree->size + 1) + (tree->size * tree->size);
        tree->insert(n);
        prior = tree->findElem(n);
        line += QString::number(n) + "," + QString::number(prior);
        line += ".\n";
        line += "Требуется вывести новое дерево, полученное в результате вставки
в него элемента ";
        line += QString::number(n);
```

```
        line += ".\n";
        line += "Дерево вводится по следующему принципу:\nсначала вводится номер
вершины-корня,\n затем поочередно вершины сначала 2 уровня, потом третьего и
т.д.\n";
        line += "Номера вершин одного уровня выводятся в порядке слева направо.\
n";
        //line += "Все недостающие вершины на уровне требуется обозначить
исмолом #.\n";
        line += "В любом случае результатом будет строка,\n которая описывает
получившееся бинарное дерево.\n";
    }
    cout << tree->correctAnswer() << endl;
    ui->taskLabel->setText(line);
}

void MainWindow::BuildTree()
{
    if(tree)
        delete tree;
    srand(time(NULL));
    tree = new Treap;
    std::pair<int, int> ans = tree->read();
    n = ans.first;
    prior = ans.second;
    tree->print();
    tree->visualize();
    ui->view->setScene(scene);
    ui->view->setRenderHint(QPainter::Antialiasing);

    scene->clear();

    item = scene->addPixmap(QPixmap("res.png"));
}

void MainWindow::CheckAnswer(){
    if (tree){
        std::string answer = (ui->lineEdit->text()).toStdString();
        std::string correctAnswer = tree->correctAnswer();
        QMessageBox* msg = new QMessageBox(this);
        if (answer == correctAnswer){
            msg->setText("Your answer is right!");
            ui->view->setScene(scene);
            ui->view->setRenderHint(QPainter::Antialiasing);
            scene->clear();
            tree->visualize();
            item = scene->addPixmap(QPixmap("res.png"));
        }
        else{
            msg->setText("Your answer is incorrect!");
        }
        msg->exec();
    }
}

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

TREAP.CPP:

```
#include "treap.h"

/*
 * Split function is used to split string objects by delimiter.
 * It is custom, because there're no split function in C++ standard library.
 * A type of return value is void, because the given vector cont stroages the
splitted string.
 */
void linesplit(const string& str, vector<string>& cont, char delim){
    stringstream ss(str);
    string token;
    while(getline(ss, token, delim)){
        cont.push_back(token);
    }
}

shared_ptr<Node> Treap::merge(shared_ptr<Node> left, shared_ptr<Node> right){
    if(!left) return right;
    if(!right) return left;
    if (left->prior > right->prior){
        left->right = merge(left->right, right);
        return left;
    }
    else{
        right->left = merge(left, right->left);
        return right;
    }
}

Pair Treap::split(shared_ptr<Node> p, int x){
    if(!p)
        return {0,0};
    if (p->key <= x){
        Pair q = split(p->right, x);
        p->right = q.first;
        return {p, q.second};
    }
    else {
        Pair q = split(p->left, x);
        p->left = q.second;
        return {q.first, p};
    }
}

void Treap::insert(int x){
    Pair q = split(data, x);
    shared_ptr<Node> t(new Node(x));
    data = merge(q.first, merge(t, q.second));
    print();
}

void Treap::remove(int key) {
    Pair fst_pair, snd_pair;
    fst_pair = split(data, key-1);
    snd_pair = split(fst_pair.second, key);
    data = merge(fst_pair.first, snd_pair.second);
    dispose(snd_pair.first);
    print();
```

```
}

void Treap::dispose(shared_ptr<Node> node) {
    if (node == nullptr)
        return;

    dispose(node->left);
    dispose(node->right);
    node.reset();
}

void Treap::PrintInOrderTraversal(shared_ptr<Node> node, int k){
    if (node){
        PrintInOrderTraversal(node->left, k+1);
        for(int i = 0; i < k; i++){
            cout << " " ;
        }
        cout << "{" << node->key << ", " << node->prior << "}";
        cout << endl;
        PrintInOrderTraversal(node->right, k+1);
    }
    else{
        return;
    }
}

void Treap::print(){
    PrintInOrderTraversal(this->data, 0);
    cout << "\n";
}

void Treap::visualize(){
    stack<shared_ptr<Node>> st;
    string file_name = "test.txt";

    //cout << "Enter the name of a file-data storager: " << endl;
    //getline(cin, file_name);
    std::fstream fs;
    fs.open(file_name, std::fstream::in | std::fstream::out|
std::fstream::trunc);
    if(data){
        st.push(data);
    }
    fs << "digraph Tree{\n";
    int k = 0;
    while(!st.empty()){
        auto node = st.top();
        st.pop();
        if(node->left){
            st.push(node->left);
            fs << "\"" << node->key << ", " << node->prior << "\"";
            fs << " -> " << "\"" << node->left->key << ", " << node->left->prior
<< "\";\n";
        }else{
            fs << k << " [shape=point];\n";
            fs << "\"" << node->key << ", " << node->prior << "\"";
            fs << " -> "  << k << ";\n";
            k++;
        }
        if(node->right){
```

```
            st.push(node->right);
            fs << "\"" << node->key << ", " << node->prior << "\"";
            fs << " -> " << "\"" << node->right->key << ", " << node->right-
>prior << "\";\n";
        }else{
            fs << k << " [shape=point];\n";
            fs << "\"" << node->key << ", " << node->prior << "\"";
            fs << " -> " << k << ";\n";
            k++;
        }

    }
    fs << "}";
    fs.close();
    string command = "dot -Tpng " + file_name + " -o res.png";
    system(command.c_str());
}

std::pair<int, int> Treap::read(){
    int count = rand()%16 + 3;
    int ind = rand()%count;
    size = count;
    int key = 0;
    std::pair<int, int> ans = {0 , 0};
    for(int i = 0; i < count; i++){
        key = rand()%(count * (i + 1)) + (i * count);
        this->insert(key);
        if(i == ind){
            ans.first = key;
            ans.second = find(data, key);
        }
    }
    return ans;
}

int Treap::findElem(int x){
    return find(data, x);
}

int Treap::find(shared_ptr<Node> node, int x){
    if(node->key == x){
        return node->prior;
    } else if(node->key >= x){
        return find(node->left, x);
    } else {
        return find(node->right, x);
    }
}

std::string Treap::correctAnswer(){
    std::vector<shared_ptr<Node>> queue;
    std::string ans = "";
    size_t qStart = 0;
    if(data){
        queue.push_back(data);
        ans += std::to_string(data->key);
        shared_ptr<Node> v = nullptr;
        while(qStart < queue.size()){
            v = queue[qStart];
            qStart++;
            if (v->left){
```

```
                queue.push_back(v->left);
                ans += std::to_string(v->left->key);
            }
            if (v->right){
                queue.push_back(v->right);
                ans += std::to_string(v->right->key);
            }
        }
    }
    return ans;
}
```

ПРИЛОЖЕНИЕ Б

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ ПРОГРАММЫ

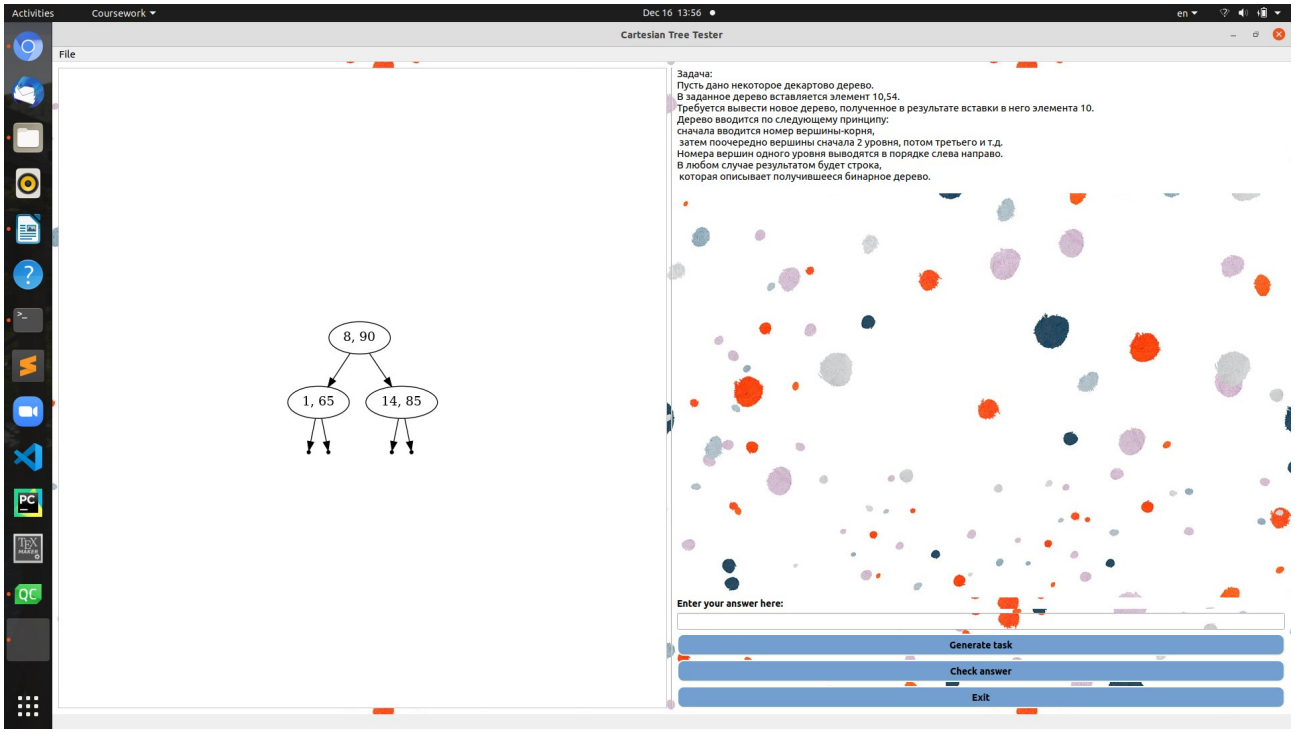
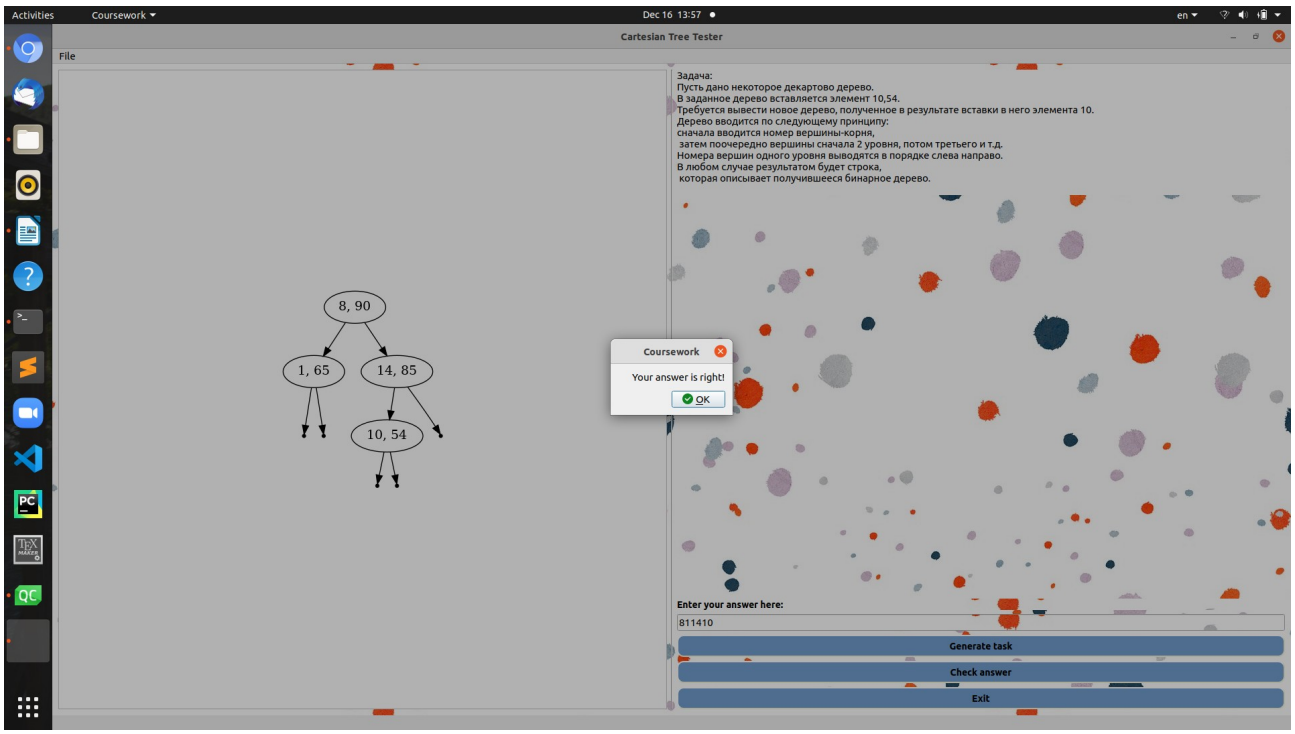


Рисунок 1 – Проверка генерации задания

Рисунок 2 – Пример правильного ответа



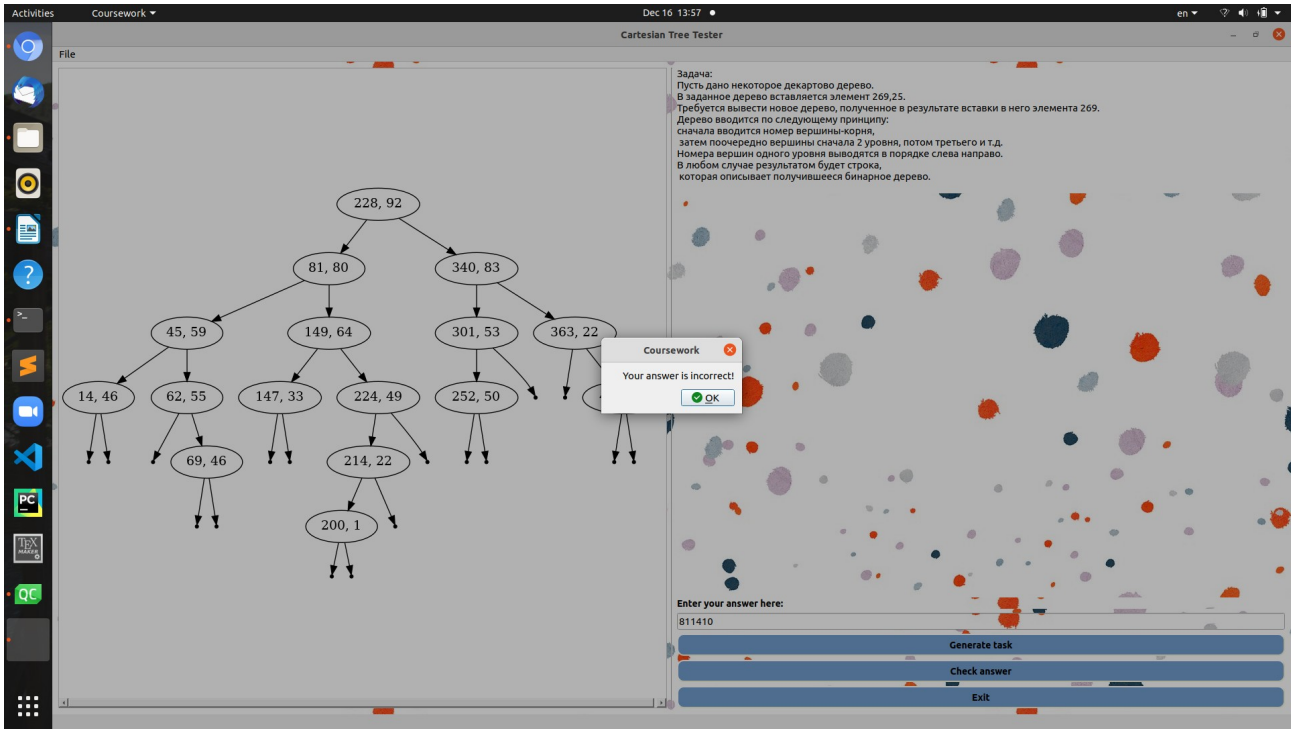


Рисунок 3 –Пример неправильного ответа на задание

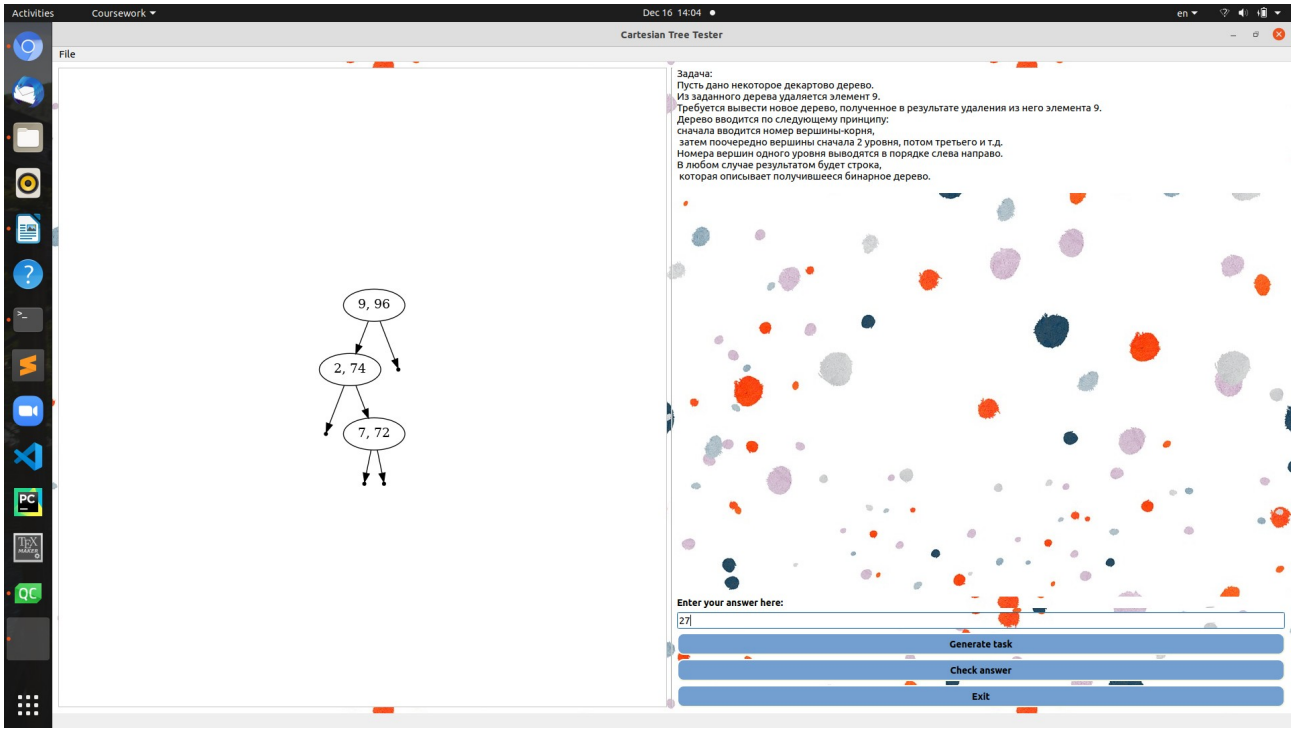


Рисунок 4 – Проверка генерации задания

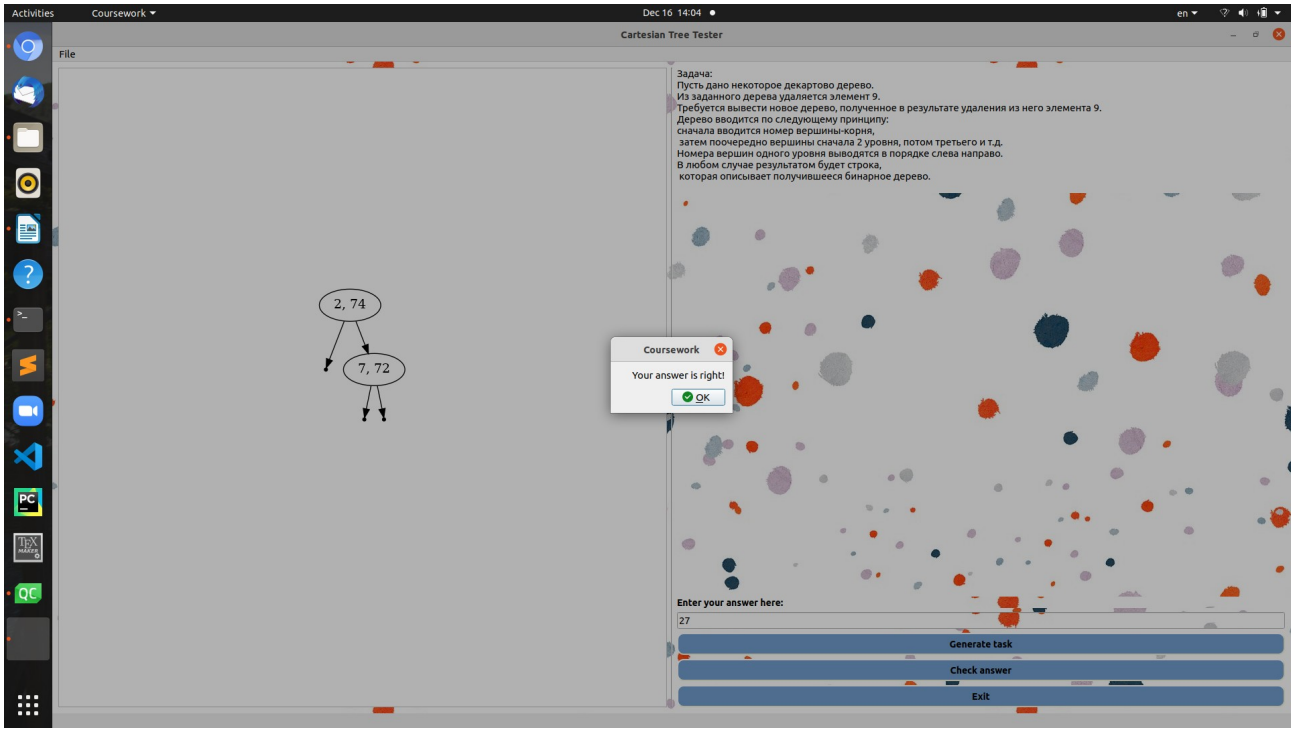


Рисунок 5 – Пример правильного ответа на задание

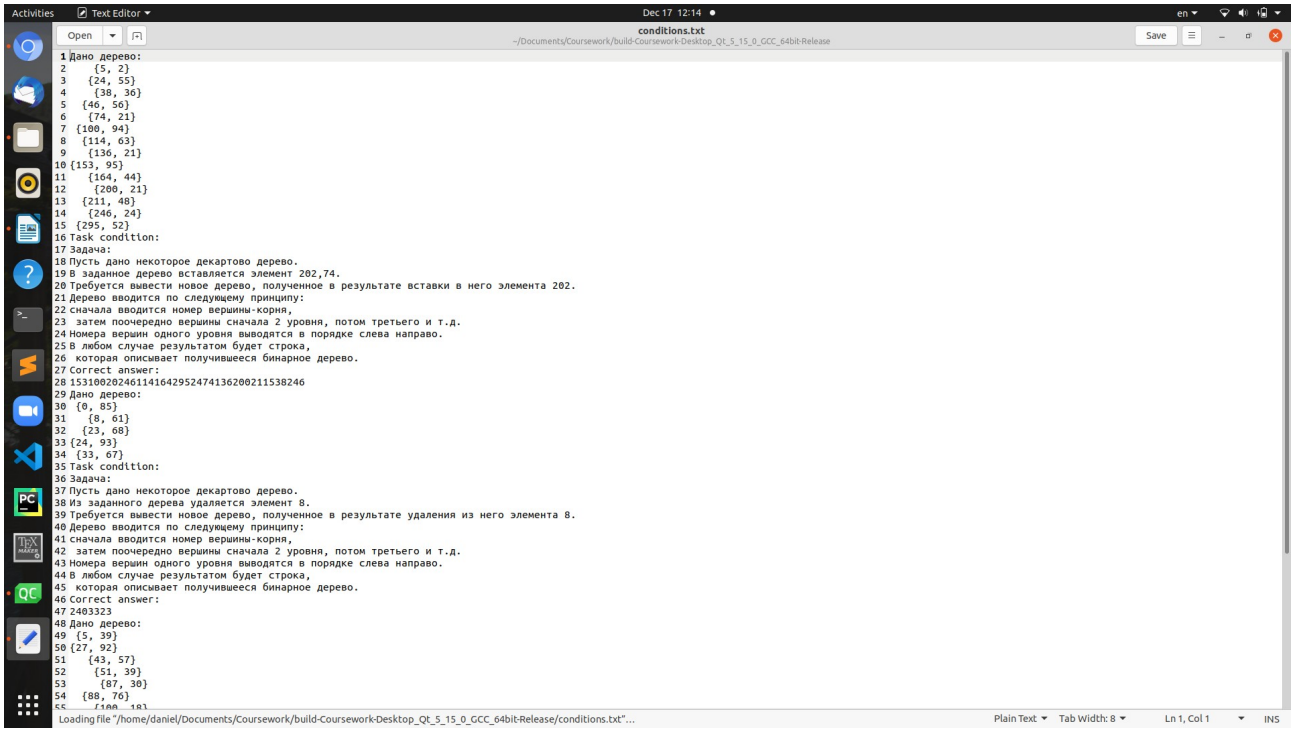


Рисунок 6 – Пример генерируемого файла с условиями и ответами на задачи