

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: AVL-деревья

Студент гр. 9303

Муратов Р.А.

Преподаватель

Филатов Ар.Ю.

Санкт-Петербург

2020

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Муратов Р.А.

Группа 9303

Тема работы: АВЛ-деревья – вставка и исключение. Текущий контроль.

Исходные данные:

Исходные данные генерируются автоматически (вопрос, ответ и граф).

Содержание пояснительной записки:

«Аннотация», «Содержание», «Введение», «Основные теоретические положения», «Описание программного кода», «Описание интерфейса пользователя», «Тестирование», «Программный код», «Заключение», «Список использованных источников».

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания: 06.11.2020

Дата сдачи реферата: 20.11.2020

Дата защиты реферата: 25.11.2020

Студент

Муратов Р.А.

Преподаватель

Филатов Ар.Ю.

АННОТАЦИЯ

Курсовая работа представляет собой программу с графическим интерфейсом для проведения текущего контроля среди студентов по теме «АВЛ-деревья – вставка и исключение». Программа может сгенерировать 10 вопросов (5 типовых вопроса * 2 варианта (вставка + исключение)). Программа разработана с помощью фреймворка Qt и языка программирования C++. Запустить программу можно только на ОС Windows 10. Исходный код, снимки экрана, которые демонстрируют процесс работы программы, и результаты тестирования представлены в приложениях.

СОДЕРЖАНИЕ

Введение	4
1. Основные теоретические положения	5
1.1. Балансировка	6
1.2. Алгоритм вставки вершины	8
1.3. Алгоритм удаления вершины	8
1.4. Оценка эффективности	10
2. Описание программного кода	11
2.1. Реализация АВЛ-дерева	11
2.2. Реализация проверяющей системы	13
3. Описание интерфейса пользователя	15
3.1. Общие сведения	15
3.2. Реализация графического интерфейса	15
3.3. Описание генерируемого файла	16
Заключение	17
Список использованных источников	18
Приложение А. Исходный код программы	19
Приложение Б. Результаты тестирования	51

ВВЕДЕНИЕ

Целью данной курсовой работы является создание программы для генерации заданий с ответами к ним для проведения текущего контроля среди студентов по теме «АВЛ-деревья – вставка и исключение». Задания и ответы должны выводиться в файл в удобной форме: тексты заданий должны быть готовы для передачи студентам, проходящим ТК; все задания должны касаться конкретных экземпляров структуры данных; ответы должны позволять удобную проверку правильности выполнения заданий.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Изучение теоретических основы АВЛ-деревьев
2. Реализация структуры данных «АВЛ-дерево» и соответствующего интерфейса
3. Реализация генератора вопросов и ответов
4. Тестирование программного кода

Также для более удобного взаимодействия с программой был реализован графический интерфейс.

1. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

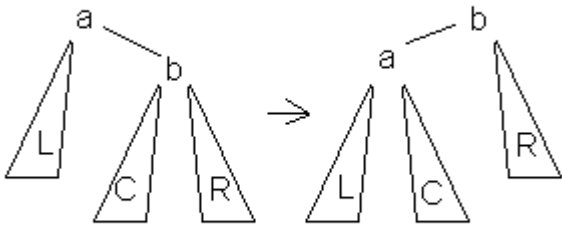
АВЛ-дерево — сбалансированное по высоте двоичное дерево поиска: для каждой его вершины высота её двух поддеревьев различается не более чем на 1.

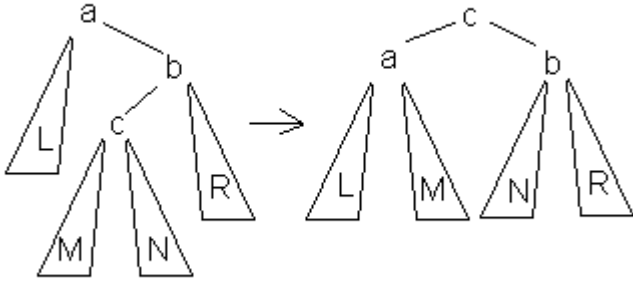
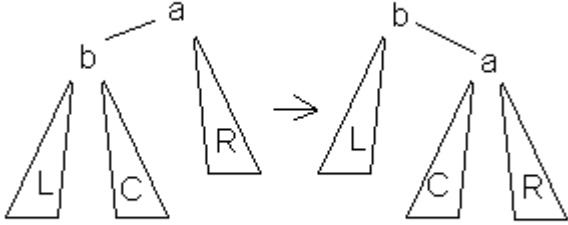
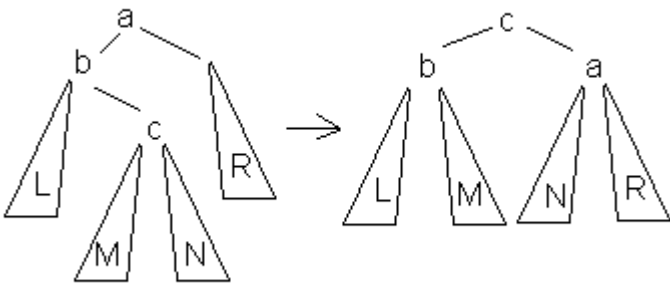
АВЛ — аббревиатура, образованная первыми буквами фамилий создателей (советских учёных) Георгия Максимовича Адельсон-Вельского и Евгения Михайловича Ландиса.

1.1. Балансировка

Относительно АВЛ-дерева балансировкой вершины называется операция, которая в случае разницы высот левого и правого поддеревьев $= 2$, изменяет связи предок-потомок в поддереве данной вершины так, что разница становится ≤ 1 , иначе ничего не меняет. Указанный результат получается вращениями поддерева данной вершины. Типы вращений представлены в табл. 1.

Таблица 1 – Типы вращений АВЛ-дерева

Малое левое вращение		Данное вращение используется тогда, когда (высота b - поддерева — высота L) $= 2$ и высота c - поддерева \leq высота R .
----------------------------	--	--

<p>Большое левое вращение</p>		<p>Данное вращение используется тогда, когда (высота b- поддерева — высота L) = 2 и высота c- поддерева > высота R.</p>
<p>Малое правое вращение</p>		<p>Данное вращение используется тогда, когда (высота b- поддерева — высота R) = 2 и высота C <= высота L.</p>
<p>Большое правое вращение</p>		<p>Данное вращение используется тогда, когда (высота b- поддерева — высота R) = 2 и высота c- поддерева > высота L.</p>

В каждом случае достаточно просто доказать то, что операция приводит к нужному результату и что полная высота уменьшается не более чем на 1 и не может увеличиться. Также можно заметить, что большое левое вращение — это композиция правого малого вращения и левого малого вращения. Из-за условия

балансированности высота дерева $O(\log(N))$, где N — количество вершин, поэтому добавление элемента требует $O(\log(N))$ операций.

1.2. Алгоритм вставки вершины

Показатель сбалансированности в дальнейшем будет интерпретироваться как разность между высотой правого и левого поддерев. Непосредственно при вставке (листу) присваивается нулевой баланс. Процесс включения вершины состоит из трёх частей (данный процесс описан Никлаусом Виртом в «Алгоритмы и структуры данных»[1]):

Прохода по пути поиска, пока не убедимся, что ключа в дереве нет.

Включения новой вершины в дерево и определения результирующих показателей балансировки.

«Отступления» назад по пути поиска и проверки в каждой вершине показателя сбалансированности. Если необходимо — балансировка.

Предположим, что процесс из левой ветви возвращается к родителю (рекурсия идёт назад), тогда возможны три случая (h_l — высота левого поддерев, h_r — высота правого поддерев):

1. $h_l < h_r$: выравнивается $h_l = h_r$. Ничего делать не нужно.
2. $h_l = h_r$: теперь левое поддерево будет больше на единицу, но балансировка пока не требуется.
3. $h_l > h_r$: теперь $h_l - h_r = 2$, — требуется балансировка.

В третьей ситуации требуется определить балансировку левого поддерев. Если левое поддерево этой вершины выше правого, то требуется большое правое вращение, иначе хватит малого правого. Аналогичные (симметричные) рассуждения можно привести и для включения в правое поддерево.

1.3. Алгоритм удаления вершины

Идея алгоритма следующая [2]: находим узел p с заданным ключом k (если не находим, то делать ничего не надо), в правом поддереве находим узел

\min с наименьшим ключом и заменяем удаляемый узел p на найденный узел \min (см. рис. 1).

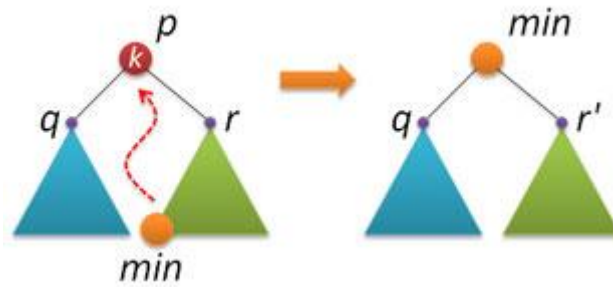


Рисунок 1 – Иллюстрация удаления ключа из АВЛ-дерева

При реализации возникает несколько нюансов. Прежде всего, если у найденный узел p не имеет правого поддерева, то по свойству АВЛ-дерева слева у этого узла может быть только один единственный дочерний узел (дерево высоты 1), либо узел p вообще лист. В обоих этих случаях надо просто удалить узел p и вернуть в качестве результата указатель на левый дочерний узел узла p .

Пусть теперь правое поддерево у p есть. Нужно найти минимальный ключ в этом поддереве. По свойству двоичного дерева поиска этот ключ находится в конце левой ветки, начиная от корня дерева.

Опять же, по свойству АВЛ-дерева у минимального элемента справа либо подвешен единственный узел, либо там пусто. В обоих случаях надо просто вернуть указатель на правый узел и по пути назад (при возвращении из рекурсии) выполнить балансировку.

Для удаления ключа сначала выполняется поиск нужного узла (те же действия, что и при вставке ключа). Как только ключ k найден, запоминаются корни q и r левого и правого поддеревьев узла p ; затем удаляется узел p ; если правое поддерево пустое, то возвращается указатель на левое поддерево; если правое поддерево не пустое, то сначала необходимо найти там минимальный элемент \min , потом извлечь его оттуда, затем слева к \min подвесить q , справа — то, что получилось из r , затем возвращается \min после его балансировки.

1.4. Оценка эффективности

Из теории известно, что высота AVL-дерева никогда не превысит высоту идеально сбалансированного дерева более, чем на 45 %. Для больших n имеет место оценка $1.04\log_2(n)$. Таким образом, выполнение основных операций требует порядка $\log_2(n)$ сравнений. Экспериментально выяснено, что одна балансировка приходится на каждые 2 включения и на каждые 5 исключений

2. ОПИСАНИЕ ПРОГРАММНОГО КОДА

2.1. Реализация АВЛ-дерева

Для реализации класса `AvlTree` также была написана структура `Node`, представляющая собой узел дерева. В ней хранятся:

- `key` – непосредственно само значение;
- `height` – высота дерева с корнем в данном узле;
- `left` – указатель на левое поддерево;
- `right` – указатель на правое поддерево.

Класс `AvlTree` имеет следующие поля (`T` – параметр шаблона, `NodePtr` = `std::shared_ptr<Node>`):

- `NodePtr head_` – указатель на голову дерева;
- `std::map<T, size_t> nodeCounter_` – счетчик ключей узлов (необходим для поддержки повторяющихся ключей)

В классе `AvlTree` реализованы основные методы `insert` (вставка элемента) и `remove` (удаление элемента). Также были реализованы следующие вспомогательные функции:

- `NodePtr insertAux(NodePtr node, const T& key)` – вспомогательный метод для `insert`, который рекурсивно ищет место в дереве для вставки элемента `key`;
- `NodePtr findMin(NodePtr node)` – метод для поиска узла с наименьшим ключом в дереве с корнем `node`;
- `NodePtr removeMin(NodePtr node)` – метод для удаления узла с минимальным ключом из дерева с корнем `node`;
- `NodePtr removeAux(NodePtr node, const T& key)` – метод для удаления `key` из дерева с корнем `node`, является вспомогательным методом для `remove`;
- `uint8_t height(NodePtr node)` – возвращает высоту дерева с корнем в `node`;

- `int8_t balanceFactor(NodePtr node)` – возвращает балансирующий фактор для узла `node`, то есть разность между высотами правого и левого поддеревьев данного узла. Это необходимо для балансировки;
- `void fixHeight(NodePtr node)` – данный метод делает значение поля `height` узла актуальным с учетом изменений в дереве;
- `NodePtr rotateLeft(NodePtr node)` – вращение влево вокруг узла `node`;
- `NodePtr rotateRight(NodePtr node)` – вращение вправо вокруг узла `node`;
- `NodePtr balance(NodePtr node)` – балансировка узла `node`.

Следующие поля и методы являются вспомогательными для удобной генерации вопросов и ответов (`RotPivot = std::pair<char, T>`):

- `std::vector<T> lastBalancedNodes_` - сбалансированные узлы после последней проведенной операции (вставка или удаление);
- `std::vector<RotPivot> lastRotations_` - направления вращения и узлы, вокруг которых они произошли после последней проведенной операции (вставка или удаление);
- `bool haveSameFrame(const AvlTree<T>& otherTree)` – проверка на совпадение каркасов текущего дерева и `otherTree`;
- `std::vector<T> getPrefixOrder() const` – получение элементов дерева в КЛП-порядке;
- `void printToFile(std::ofstream& outFile) const` - печать дерева в файл в виде уступчатого списка;
- `void printFrameToFile(std::ofstream& outFile) const` – печать каркаса дерева в файл в виде уступчатого списка;
- `void printDot(std::fstream& dotFile) const` - печать дерева (без количества повторений) в файл `dotFile` на языке DOT;

- `void printFrameDot(std::fstream& dotFile) const` - печать каркаса дерева (то есть без указания элементов) в файл `dotFile` на языке DOT.

2.2. Реализация проверяющей системы

`ICheckAnswer` – интерфейс для шаблона проектирования «Стратегия».

`FullMatch` – класс, реализующий интерфейс `ICheckAnswer`, метод `checkAnswer` которого проверяет на полное совпадение ответа пользователя с эталонным ответом.

`OneOf` – класс, реализующий интерфейс `ICheckAnswer`, метод `checkAnswer` которого проверяет, входит ли ответ пользователя в заданный набор допустимых ответов.

`QuestionAnswer` – класс, который содержит в себе вопрос (`std::string question_`), ответ на него (`std::string answer_`), стратегию проверки (`std::shared_ptr<ICheckStrategy> checkStrategy_`) и количество графов, связанных с вопросом (`int8_t graphNumber_`, на данный момент поддерживается вывод на экран 1-ого или 2-ух графов).

Основные методы класса `QuestionAnswer`:

- `std::pair<bool, int32_t> checkAnswer(std::string userAnswer)` - проверка, является ли правильным введенный пользователем ответ. Если нет, то указывается, с какого места ответ неправильный;
- `std::vector<std::string> getGraphFilePath() const` – пути до PNG-файлов с графами, данные файлы создаются с помощью утилиты `GraphViz`.

`QuestionGenerator` – класс, который с помощью одного метода `generateQA`, генерирует вопрос, соответствующее дерево (в дереве может быть от 5 до 15 элементов) и ответ. Также у класса имеется поле `qaFile_`,

который хранит указатель на файл, в который записываются дерево, вопрос и ответ на него.

В методе `generateQA` создается исходное дерево, хранящее элементы типа `char`, оно заполняется неповторяющимися символами (генератор псевдослучайных чисел `rand()`), затем с помощью генератора псевдослучайных чисел выбирается число от 1 до 10 и с помощью оператора `switch` выбирается вопрос. Затем вопрос, дерево и ответ записываются в файл `*qaFile_` и возвращается объект класса `QuestionAnswer`.

Типы генерируемых вопросов:

- Как будет выглядеть дерево при вставке/удалении элемента N? Перечислите элементы полученного дерева в КЛП порядке.
- Перечислите узлы, в которых нарушится баланс при вставке/удалении элемента N. Таких узлов может не оказаться.
- Перечислите последовательность поворотов поддеревьев и их корень при вставке/удалении элемента N. Поворотов может и не произойти.
- Как будет выглядеть дерево, если сначала вставить/удалить элемент N, а затем удалить/вставить элемент M? Перечислите элементы полученного дерева в КЛП порядке.
- Какой элемент необходимо вставить(удалить) в(из) дерево(а), чтобы его каркас совпадал с тем, который изображен рядом?

3. ОПИСАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

3.1. Общие сведения

При запуске приложения в центр окна выводится изображение графа, в нижней части окна имеется поле для ввода ответа и две кнопки: «Проверить» и «След. вопрос». Первая кнопка, как ни странно, проверяет ответ пользователя и выводит в status bar вердикт (также ответ можно проверить с помощью нажатия клавиши Enter). Если ответ пользователя правильный, то будет написано: «Ответ правильный!», иначе будет написано: «Ответ неверный: правильная часть Вашего ответа ...», где на месте троеточия будет указана правильная часть текущего ответа пользователя. Кнопка «След. вопрос» становится доступна только после правильного ответа пользователя.

Как только перед пользователем выводится очередной вопрос, в файл записываются дерево, вопрос и ответ на него. Подробнее о формате записи сказано в подразделе 3.3.

3.2. Реализация графического интерфейса

Класс `CurrentControl` – основной класс, наследник `QMainWindow`. В нем имеются следующие поля и методы:

- `Ui::CurrentControl *ui` – указатель на класс, сгенерированный moc по форме `currencontrol.ui`;
- `QGraphicsScene* gscene_` - умный указатель на графическую сцену, на которой отображаются деревья;
- `PixmapPtr pixmapFrame_` - умный указатель на изображение каркаса дерева;
- `GraphicsItemPtr gitemFrame_` - умный указатель на объект сцены, представляющий собой изображение каркаса дерева;
- `QuestionGeneratorPtr questionGen_` - умный указатель на генератор вопросов;

- `QuestionAnswerPtr qa_` - умный указатель на объект, содержащий вопрос и ответ.
- `void newQuestion()` – метод для генерации нового вопроса и вывода соответствующих данных на экран;
- `void checkAnswer()` – метод для проверки ответа пользователя на корректность;

3.3. Описание генерируемого файла

В файл после слов «Дано АВЛ-дерево:» или «Дано АВЛ-дерево и его каркас после некоторой операции:» после символа переноса строки выводятся АВЛ-дерево и его каркас (если это нужно для вопроса) в виде уступчатого списка. Если у узла два сына, то сначала указывается левый сын, затем правый, при отсутствии сына ставится минус «-» (например, если узел является листом, то ставится два минуса).

Затем после символа переноса строки ставится метка «Q:», после которой через пробел записывается вопрос и ставится символ переноса строки. Далее после метки «A:» записывается ответ на заданный вопрос. Затем записываются два символа перевода строки для разделения блоков и удобного чтения файла человеком.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была разработана программа с графическим интерфейсом, которая позволяет проводить текущий контроль среди студентов по теме «АВЛ-деревья – вставка и исключение». Также программа создает файл со сгенерированными деревьями, вопросами и ответами на заданную тему. Таким образом, результат полностью соответствует поставленной цели.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Вирт Н. Алгоритмы и структуры данных. — М.: Мир, 1989. — С. 272—286.
2. AVL-деревья: <https://habr.com/ru/post/150732/>
3. The C++ Resource Network: <https://www.cplusplus.com/>
4. Qt Documentation: <https://doc.qt.io/qt-5/>

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

avltree.h

```
#ifndef AVLTREE_H
#define AVLTREE_H

#include <iostream>
#include <fstream>
#include <string>
#include <algorithm>
#include <map>
#include <vector>

#include "node.h"

template<class T>
class AvlTree {
    // Умный указатель на узел
    using NodePtr = std::shared_ptr<Node<T>>;
    // Пара, в которой хранится информация о вращении и корне, вокруг
    // которого оно происходит
    using RotPivot = std::pair<char, T>;
public:
    AvlTree() = default;
    ~AvlTree() = default;

    // Создание нового дерева на основе otherTree
    AvlTree(const AvlTree<T>& otherTree);
    AvlTree<T>& operator=(const AvlTree<T>& otherTree);

    // Вставка элемента key в дерево
    void insert(const T& key);
    // Удаление элемента key из дерева
    void remove(const T& key);

    // Проверка совпадения каркасов деревьев
    bool haveSameFrame(const AvlTree<T>& otherTree) const;

    // Получение элементов дерева в КЛП-порядке
    std::vector<T> getPrefixOrder() const;

    // Геттеры для полей класса
```

```

        const std::map<T, size_t>& getNodeCounter() const { return
nodeCounter_; }
        const std::vector<T>& getLastBalancedNodes() const { return
lastBalancedNodes_; }
        const std::vector<RotPivot>& getLastRotations() const { return
lastRotations_; }

        // Печать элементов дерева в файл oufFile в ЛКП-порядке
void printInfix(std::ofstream& outFile) const;
        // Печать дерева в файл в виде уступчатого списка
void printToFile(std::ofstream& outFile) const;
        // Печать каркаса дерева в файл в виде уступчатого списка
void printFrameToFile(std::ofstream& outFile) const;
        // Печать дерева в консоль
void printToConsole() const;
        // Печать дерева (без количества повторений) в файл dotFile на
языке DOT
        void printDot(std::fstream& dotFile) const;
        // Печать дерева (с количеством повторений) в файл dotFile на
языке DOT
        void printDotWithCount(std::fstream& dotFile) const;
        // Печать каркаса дерева (то есть без указания элементов) в файл
dotFile на языке DOT
        void printFrameDot(std::fstream& dotFile) const;

protected:
        // Высота узла node
uint8_t height(NodePtr node);
        // Разность высот правого и левого поддеревьев узла node
int8_t balanceFactor(NodePtr node);
        // Восстановление актуальности поля height узла node
void fixHeight(NodePtr node);

        // Вращения налево и направо вокруг узла node
NodePtr rotateLeft(NodePtr node);
NodePtr rotateRight(NodePtr node);

        // Балансировка узла node
NodePtr balance(NodePtr node);

        // Вспомогательная функция для метода insert
NodePtr insertAux(NodePtr node, const T& key);

```

```

        // Вспомогательная функция для конструктора копирования и
оператора присваивания
        NodePtr copyAux(NodePtr otherNode);

        // Поиск узла с наименьшим ключом в дереве с корнем node
        NodePtr findMin(NodePtr node);
        // Удаление узла с минимальным ключом из дерева с корнем node
        NodePtr removeMin(NodePtr node);
        // Удаление ключа k из дерева с корнем node (вспомогательная
функция для метода remove)
        NodePtr removeAux(NodePtr node, const T& key);

        // Элементы дерева в КЛП-порядке
        std::vector<T> getPrefixOrderAux(NodePtr node, std::vector<T>&
vec) const;

        // Вспомогательная функция для метода haveSameFrame
        bool haveSameFrameAux(NodePtr thisNode, NodePtr otherNode) const;

        // Вспомогательная функция для метода printToFile
        void printToFileAux(NodePtr node, std::ofstream& outFile, uint8_t
step) const;
        // Вспомогательная функция для метода printFrameToFile
        void printFrameToFileAux(NodePtr node, std::ofstream& outFile,
uint8_t step) const;
        // Вспомогательная функция для метода printInfix
        void printInfixAux(NodePtr node, std::ofstream& outFile) const;
        // Вспомогательная функция для метода printToConsole
        void stepPrint(NodePtr node, uint8_t step) const;

        // Вспомогательные функции для методов printDot,
        // printDotWithCountAux и printFrameDotAux соответственно
        void printDotAux(NodePtr node, std::fstream& dotFile) const;
        void printDotWithCountAux(NodePtr node, std::fstream& dotFile)
const;
        void printFrameDotAux(NodePtr node, std::fstream& dotFile) const;

private:
        // Указатель на голову дерева
        NodePtr head_ = nullptr;

        // Счетчик количества повторении элемента в дереве (для
поддержания повторяющихся ключей)
        std::map<T, size_t> nodeCounter_;

```

```

        // Сбалансированные узлы после последней проведенной операции
        (вставка или удаление)
        std::vector<T> lastBalancedNodes_;

        // Направления вращения и узлы, вокруг которых они произошли
        после последней
        // проведенной операции (вставка или удаление)
        std::vector<RotPivot> lastRotations_;
    };

```

```

#include "avltree.cpp"

```

```

#endif // AVL_TREE_H

```

avltree.cpp

```

#ifndef BIN_TREE_CPP
#define BIN_TREE_CPP

#include "avltree.h"

template<class T>
AvlTree<T>::AvlTree(const AvlTree<T>& otherTree) {
    head_ = copyAux(otherTree.head_);

    nodeCounter_ = otherTree.nodeCounter_;
    lastBalancedNodes_ = otherTree.lastBalancedNodes_;
    lastRotations_ = otherTree.lastRotations_;
}

template<class T>
AvlTree<T>& AvlTree<T>::operator=(const AvlTree<T>& otherTree) {
    // Дерево не может присвоить саму себя
    if (this != &otherTree) {
        head_ = copyAux(otherTree.head_);

        nodeCounter_ = otherTree.nodeCounter_;
        lastBalancedNodes_ = otherTree.lastBalancedNodes_;
        lastRotations_ = otherTree.lastRotations_;
    }
    return *this;
}

template<class T>

```

```

void AvlTree<T>::insert(const T& key) {
    // Удаляем информацию, относящуюся к прошлому действию
    lastBalancedNodes_.clear();
    lastRotations_.clear();

    head_ = insertAux(head_, key);
}

template<class T>
void AvlTree<T>::remove(const T& key) {
    // Удаляем информацию, относящуюся к прошлому действию
    lastBalancedNodes_.clear();
    lastRotations_.clear();

    head_ = removeAux(head_, key);
}

template<class T>
bool AvlTree<T>::haveSameFrame(const AvlTree<T>& otherTree) const {
    return haveSameFrameAux(head_, otherTree.head_);
}

template<class T>
std::vector<T> AvlTree<T>::getPrefixOrder() const {
    std::vector<T> v;
    v = getPrefixOrderAux(head_, v);
    return v;
}

template<class T>
void AvlTree<T>::printInfix(std::ofstream& outFile) const {
    printInfixAux(head_, outFile);
}

template<class T>
void AvlTree<T>::printToFile(std::ofstream& outFile) const {
    printToFileAux(head_, outFile, 0);
}

template<class T>
void AvlTree<T>::printFrameToFile(std::ofstream& outFile) const {
    printFrameToFileAux(head_, outFile, 0);
}

```

```

template<class T>
void AvlTree<T>::printToConsole() const {
    stepPrint(head_, 0);
    std::cout << "-----\n";
}

template<class T>
void AvlTree<T>::printDot(std::fstream& dotFile) const {
    dotFile << "digraph AVL_tree {\n";

    // Здесь позже можно настроить узлы и ребра

    if (!head_)
        dotFile << "\n";
    else if (!head_->left && !head_->right)
        dotFile << "  n [label = \"" << head_->key << "\"]" << ";\n";
    else
        printDotAux(head_, dotFile);

    dotFile << "}\n";
}

template<class T>
void AvlTree<T>::printDotWithCount(std::fstream& dotFile) const {
    dotFile << "digraph AVL_tree {\n";

    // Здесь позже можно настроить узлы и ребра

    if (!head_)
        dotFile << "\n";
    else if (!head_->left && !head_->right)
        dotFile << "  n [label = \"" << head_->key << " (" <<
nodeCounter_.at(head_->key) << ")]" << ";\n";
    else
        printDotWithCountAux(head_, dotFile);

    dotFile << "}\n";
}

template<class T>
void AvlTree<T>::printFrameDot(std::fstream& dotFile) const {
    dotFile << "digraph AVL_tree {\n";

    // Здесь позже можно настроить узлы и ребра

```



```

        if (!head_)
            dotFile << "\n";
        else if (!head_->left && !head_->right)
            dotFile << "  n [label = \"#\"]" << ";\n";
        else
            printFrameDotAux(head_, dotFile);

        dotFile << "}\n";
    }

template<class T>
uint8_t AvlTree<T>::height(NodePtr node) {
    return node ? node->height : 0;
}

template<class T>
int8_t AvlTree<T>::balanceFactor(NodePtr node) {
    return height(node->right) - height(node->left);
}

template<class T>
void AvlTree<T>::fixHeight(NodePtr node) {
    node->height = std::max(height(node->left), height(node->right))
+ 1;
}

template<class T>
typename AvlTree<T>::NodePtr AvlTree<T>::rotateLeft(NodePtr node) {
    NodePtr newHead(node->right);
    node->right = newHead->left;
    newHead->left = node;
    fixHeight(node);
    fixHeight(newHead);
    return newHead;
}

template<class T>
typename AvlTree<T>::NodePtr AvlTree<T>::rotateRight(NodePtr node)
{
    NodePtr newHead(node->left);
    node->left = newHead->right;
    newHead->right = node;
    fixHeight(node);

```

```

        fixHeight(newHead);
        return newHead;
    }

template<class T>
typename AvlTree<T>::NodePtr AvlTree<T>::balance(NodePtr node) {
    fixHeight(node);
    if (balanceFactor(node) == 2) { // h(right) - h(left) == 2
        if (balanceFactor(node->right) < 0) {
            lastRotations_.emplace_back('r', node->right->key);
            node->right = rotateRight(node->right);
        }
        lastRotations_.emplace_back('l', node->key);
        lastBalancedNodes_.push_back(node->key);
        return rotateLeft(node);
    } else if (balanceFactor(node) == -2) { // h(right) - h(left) ==
-2
        if (balanceFactor(node->left) > 0) {
            lastRotations_.emplace_back('l', node->left->key);
            node->left = rotateLeft(node->left);
        }
        lastRotations_.emplace_back('r', node->key);
        lastBalancedNodes_.push_back(node->key);
        return rotateRight(node);
    }
    return node; // Балансировка не нужна
}

template<class T>
typename AvlTree<T>::NodePtr AvlTree<T>::insertAux(NodePtr node,
const T& key) {
    if (!node) {
        nodeCounter_[key] = 1;
        return std::make_shared<Node<T>>(key);
    }
    if (key < node->key)
        node->left = insertAux(node->left, key);
    else if (key > node->key)
        node->right = insertAux(node->right, key);
    else // Если узел уже имеется в дереве, то учитываем это
        ++nodeCounter_.at(key);
    return balance(node);
}

```

```

template<class T>
typename AvlTree<T>::NodePtr AvlTree<T>::copyAux(NodePtr otherNode)
{
    NodePtr node = std::make_shared<Node<T>>(otherNode->key,
otherNode->height);
    if (otherNode->left) {
        node->left = copyAux(otherNode->left);
    }
    if (otherNode->right) {
        node->right = copyAux(otherNode->right);
    }
    return node;
}

template<class T>
typename AvlTree<T>::NodePtr AvlTree<T>::findMin(NodePtr node) {
    while (node->left)
        node = node->left;
    return node;
}

template<class T>
typename AvlTree<T>::NodePtr AvlTree<T>::removeMin(NodePtr node) {
    if (!node->left) {
        return node->right;
    }
    node->left = removeMin(node->left);
    return balance(node);
}

template<class T>
typename AvlTree<T>::NodePtr AvlTree<T>::removeAux(NodePtr node,
const T& key) {
    if (!node)
        return nullptr;
    if (key < node->key) {
        node->left = removeAux(node->left, key);
    } else if (key > node->key) {
        node->right = removeAux(node->right, key);
    } else {
        if (--nodeCounter_.at(key) == 0) { // Удаляем узел из дерева
            только тогда, когда его счетчик обнулится
            if (!node->right)
                return node->left;
        }
    }
}

```

```

        NodePtr minElem = findMin(node->right);
        minElem->right = removeMin(node->right);
        minElem->left = node->left;
        return balance(minElem);
    }
}
return balance(node);
}

template<class T>
std::vector<T>    AvlTree<T>::getPrefixOrderAux(AvlTree<T>::NodePtr
node, std::vector<T>& vec) const {
    if (node) {
        vec.push_back(node->key);
        vec = getPrefixOrderAux(node->left, vec);
        vec = getPrefixOrderAux(node->right, vec);
    }
    return vec;
}

template<class T>
bool    AvlTree<T>::haveSameFrameAux(NodePtr    thisNode,    NodePtr
otherNode) const {
    if (thisNode && otherNode) {
        return haveSameFrameAux(thisNode->left, otherNode->left) &&
            haveSameFrameAux(thisNode->right, otherNode->right);
    } else if (thisNode || otherNode) {
        return false;
    } else { // thisNode == otherNode == nullptr
        return true;
    }
}

template<class T>
void    AvlTree<T>::printToFileAux(NodePtr    node,    std::ofstream&
outFile, uint8_t step) const {
    for (uint8_t i = 0; i < step; ++i)
        outFile << " ";
    if (node) {
        outFile << node->key << std::endl;
        printToFileAux(node->left, outFile, step + 1);
        printToFileAux(node->right, outFile, step + 1);
    } else {
        outFile << "-" << std::endl;
    }
}

```

```

    }
}

template<class T>
void AvlTree<T>::printFrameToFileAux(NodePtr node, std::ofstream&
outFile, uint8_t step) const {
    for (uint8_t i = 0; i < step; ++i)
        outFile << " ";
    if (node) {
        outFile << "#" << std::endl;
        printFrameToFileAux(node->left, outFile, step + 1);
        printFrameToFileAux(node->right, outFile, step + 1);
    } else {
        outFile << "-" << std::endl;
    }
}

template<class T>
void AvlTree<T>::printInfixAux(NodePtr node, std::ofstream&
outFile) const {
    if (node) {
        printInfixAux(node->left, outFile);
        outFile << node->key << " (" << nodeCounter_.at(node->key) <<
")" << " ";
        printInfixAux(node->right, outFile);
    }
}

template<class T>
void AvlTree<T>::stepPrint(NodePtr node, uint8_t step) const {
    for (uint8_t i = 0; i < step; ++i)
        std::cout << " ";
    if (node) {
        std::cout << node->key << " (" << nodeCounter_.at(node->key)
<< ")" << std::endl;
        stepPrint(node->left, step + 1);
        stepPrint(node->right, step + 1);
    } else {
        std::cout << "-" << std::endl;
    }
}

template<class T>

```

```

void AvlTree<T>::printDotAux(NodePtr node, std::fstream& dotFile)
const {
    static int counter = 0;
    if (!node) {
        dotFile << "  n" << ++counter << " [ shape = point ];\n";
        return;
    }

    std::tuple<int, int, int> indexes = {-1, -1, -1};
    dotFile << "  n" << ++counter << " [ label = " << "\"" << node->key << "\"" << " ];\n";

    std::get<0>(indexes) = counter;

    std::get<1>(indexes) = counter + 1;
    printDotAux(node->left, dotFile);

    std::get<2>(indexes) = counter + 1;
    printDotAux(node->right, dotFile);

    dotFile << "  n" << std::get<0>(indexes) << " -> {" <<
        ((std::get<1>(indexes) > 0) ? "n" +
std::to_string(std::get<1>(indexes)) : "") << " " <<
        ((std::get<2>(indexes) > 0) ? "n" +
std::to_string(std::get<2>(indexes)) : "") << "};\n";
    }

template<class T>
void AvlTree<T>::printDotWithCountAux(NodePtr node, std::fstream&
dotFile) const {
    static int counter = 0;
    if (!node) {
        dotFile << "  n" << ++counter << " [ shape = point ];\n";
        return;
    }

    std::tuple<int, int, int> indexes = {-1, -1, -1};
    dotFile << "  n" << ++counter << " [ label = " << "\"" << node->key << " (" << nodeCounter_.at(node->key) << ")\" << " ];\n";

    std::get<0>(indexes) = counter;

    std::get<1>(indexes) = counter + 1;
    printDotAux(node->left, dotFile);

```

```

        std::get<2>(indexes) = counter + 1;
        printDotAux(node->right, dotFile);

        dotFile << "  n" << std::get<0>(indexes) << " -> {" <<
            ((std::get<1>(indexes) > 0) ? "n" +
std::to_string(std::get<1>(indexes)) : "") << " " <<
            ((std::get<2>(indexes) > 0) ? "n" +
std::to_string(std::get<2>(indexes)) : "") << "};\n";
    }

    template<class T>
    void AvlTree<T>::printFrameDotAux(NodePtr node, std::fstream&
dotFile) const {
        static int counter = 0;
        if (!node) {
            dotFile << "  n" << ++counter << " [ shape = point ];\n";
            return;
        }

        std::tuple<int, int, int> indexes = {-1, -1, -1};
        dotFile << "  n" << ++counter << " [ label = " << "\"#\n";

        std::get<0>(indexes) = counter;

        std::get<1>(indexes) = counter + 1;
        printFrameDotAux(node->left, dotFile);

        std::get<2>(indexes) = counter + 1;
        printFrameDotAux(node->right, dotFile);

        dotFile << "  n" << std::get<0>(indexes) << " -> {" <<
            ((std::get<1>(indexes) > 0) ? "n" +
std::to_string(std::get<1>(indexes)) : "") << " " <<
            ((std::get<2>(indexes) > 0) ? "n" +
std::to_string(std::get<2>(indexes)) : "") << "};\n";
    }

#endif

```

currentcontrol.h

```

#ifndef CURRENTCONTROL_H
#define CURRENTCONTROL_H

```

```

#include <QMainWindow>
#include <QGraphicsScene>
#include <QGraphicsPixmapItem>
#include <QLabel>
#include <QKeyEvent>
#include <QDebug>

#include <memory>
#include <algorithm>
#include <cctype>

#include "questiongenerator.h"

QT_BEGIN_NAMESPACE
namespace Ui { class CurrentControl; }
QT_END_NAMESPACE

class CurrentControl : public QMainWindow {
    Q_OBJECT

    using QuestionGeneratorPtr = std::shared_ptr<QuestionGenerator>;
    using QuestionAnswerPtr = std::shared_ptr<QuestionAnswer>;
    using QPixmapPtr = std::shared_ptr<QPixmap>;
    using GraphicsItemPtr = std::shared_ptr<QGraphicsItem>;
public:
    CurrentControl(QWidget *parent = nullptr);
    ~CurrentControl();

    void keyPressEvent(QKeyEvent* event) override;

public slots:
    void newQuestion();
    void checkAnswer();

private:
    Ui::CurrentControl *ui;

    QGraphicsScene* gscene_ = new QGraphicsScene(this);
    QPixmapPtr pixmap_ = nullptr;
    QPixmapPtr pixmapFrame_ = nullptr;

    GraphicsItemPtr gitem_ = nullptr;
    GraphicsItemPtr gitemFrame_ = nullptr;

```



```

        QLabel* answerStatus_ = new QLabel("", nullptr);

        QuestionGeneratorPtr                questionGen_
std::make_shared<QuestionGenerator>();
        QuestionAnswerPtr qa_ = nullptr;
    };
#endif // CURRENTCONTROL_H

```

currentcontrol.cpp

```

#include "currentcontrol.h"
#include "ui_currentcontrol.h"

CurrentControl::CurrentControl(QWidget *parent)
    : QMainWindow(parent),
      ui(new Ui::CurrentControl) {
    ui->setupUi(this);

    this->showMaximized();

    ui->graphicsView->setScene(gscene_);

    ui->statusbar->addWidget(answerStatus_);

    connect(ui->checkButton, &QPushButton::clicked,
            this, &CurrentControl::checkAnswer);
    connect(ui->nextQuestionBtn, &QPushButton::clicked,
            this, &CurrentControl::newQuestion);

    newQuestion();
}

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

void CurrentControl::keyPressEvent(QKeyEvent* event) {
    if (event->key() == Qt::Key_Return) {
        ui->checkButton->click();
    }
}

void CurrentControl::newQuestion() {

```

```

        qa_ = std::make_shared<QuestionAnswer>(questionGen_-
>generateQA());

        // Нельзя перейти к следующему вопросу, пока нет правильного
        ответа на текущий
        ui->nextQuestionBtn->setEnabled(false);

        // Публикация вопроса
        ui->questionLabel->setText(qa_->getQuestion().c_str());

        // Возвращение к начальным значениям
        ui->answer->clear();
        answerStatus_->setText("");

        // Удаление изображения прошлого графа
        if (gitem_) {
            gscene_->removeItem(gitem_.get());
        }

        // Удаление изображения каркаса
        if (gitemFrame_) {
            gscene_->removeItem(gitemFrame_.get());
            gitemFrame_ = nullptr;
        }

        // Создание изображения нового графа
        pixmap_ = std::make_shared<QPixmap>(qa_-
>getGraphFilePath()[0].c_str());
        gitem_ = GraphicsItemPtr(gscene_->addPixmap(*pixmap_));

        switch (qa_->getGraphNumber()) {
        case 1: {
            gscene_->setSceneRect(0, 0, pixmap_->width(), pixmap_-
>height());
            // this->resize(2 * pixmap_->width(), 2 * pixmap_->height());
            break;
        }
        case 2: {
            pixmapFrame_ = std::make_shared<QPixmap>(qa_-
>getGraphFilePath()[1].c_str());
            gitemFrame_ = GraphicsItemPtr(gscene_-
>addPixmap(*pixmapFrame_));
            gitemFrame_->setPos(gitem_->pos() + QPointF(pixmap_->width() +
20, 0));
        }
    }

```

```

        gscene_>setSceneRect(0, 0, pixmap_>width() + pixmapFrame_>
>width() + 20, std::max(pixmap_>height(), pixmapFrame_>height()));
        // this->resize(pixmap_>width() + pixmapFrame_>width() + 100,
pixmap_>height() + pixmapFrame_>height());
        break;
    }
}

void CurrentControl::checkAnswer() {
    // Удаление старого статуса ответа
    answerStatus_>setText("");

    // Ответ, введенный пользователем
    std::string userAnswer = ui->answer->text().toStdString();

    // Проверка ответа
    auto status = qa_>checkAnswer(ui->answer->text().toStdString());
    if (!status.first) {
        answerStatus_>setText(QString::fromStdString("Ответ неверный:
правильная часть Вашего ответа \"" + userAnswer.substr(0, status.second)
+ "\""));
    } else {
        // Можно переходить к следующему вопросу
        ui->nextQuestionBtn->setEnabled(true);
        answerStatus_>setText("Ответ правильный!");
    }
    // qDebug() << __PRETTY_FUNCTION__ << "status.first = " <<
status.first;
}

```

fullmatch.h

```

#ifndef FULLMATCH_H
#define FULLMATCH_H

#include <algorithm>

#include "icheckstrategy.h"

// --- Паттерн "Стратегия" ---
// --- Проверка ответа пользователя на полное совпадение с
правильным ответом ---
class FullMatch : public ICheckStrategy {
public:

```

```

        FullMatch(std::string a);
        ~FullMatch() = default;

        std::pair<bool, int32_t> checkAnswer(std::string userAnswer)
override;

    private:
        std::string answer_;
    };

#endif // FULLMATCH_H

```

fullmatch.cpp

```

#include "fullmatch.h"

FullMatch::FullMatch(std::string a)
    : answer_(a) {}

std::pair<bool, int32_t> FullMatch::checkAnswer(std::string
userAnswer) {
    // Удаление пробельных символов
    userAnswer.erase(std::remove(userAnswer.begin(),
userAnswer.end(), ' '), userAnswer.end());

    // Правильная ли строка и индекс, начиная с которого строка
является неправильной
    std::pair<bool, int32_t> correctness;

    correctness.first = userAnswer == answer_;
    if (!correctness.first) {
        if (userAnswer.length() > answer_.length()) {
            const auto& mismatchPos = std::mismatch(answer_.begin(),
answer_.end(), userAnswer.begin());
            correctness.second = mismatchPos.first - answer_.begin();
        } else {
            const auto& mismatchPos = std::mismatch(userAnswer.begin(),
userAnswer.end(), answer_.begin());
            correctness.second = mismatchPos.second - answer_.begin();
        }
    }
    return correctness;
}

```

icheckstrategy.h

```
#ifndef ICHECKSTRATEGY_H
#define ICHECKSTRATEGY_H

#include <string>

// --- Интерфейс для паттерна "Стратегия" ---
class ICheckStrategy {
public:
    virtual ~ICheckStrategy() = default;

    // Функция проверяет, является ли правильным введенный ответ.
    // Если нет, то указывается, с какого места ответ неправильный.
    virtual std::pair<bool, int32_t> checkAnswer(std::string
userAnswer) = 0;
};

#endif // ICHECKSTRATEGY_H
```

node.h

```
#ifndef NODE_H
#define NODE_H

#include <cinttypes>
#include <memory>

template<class T>
struct Node {
    using NodePtr = std::shared_ptr<Node<T>>;

    Node(T key);
    Node(T key, uint8_t h);
    ~Node() = default;

    T key;
    uint8_t height = 1;
    NodePtr left = nullptr;
    NodePtr right = nullptr;
};

template<class T>
Node<T>::Node(T key)
    : key(key) {}
```

```

        template<class T>
        Node<T>::Node(T key, uint8_t h)
            : key(key), height(h) {}

#endif // NODE_H

oneof.h
#ifndef ONEOF_H
#define ONEOF_H

#include <algorithm>

#include "icheckstrategy.h"

// --- Паттерн "Стратегия" ---
// --- Проверка ответа пользователя на вхождение в набор правильных
ответов ---
class OneOf : public ICheckStrategy {
public:
    OneOf(std::string a);
    ~OneOf() = default;

    std::pair<bool, int32_t> checkAnswer(std::string userAnswer)
override;

private:
    std::string answer_;
};

#endif // ONEOF_H

```

```

oneof.cpp
#include "oneof.h"

OneOf::OneOf(std::string a)
    : answer_(a) {}

std::pair<bool, int32_t> OneOf::checkAnswer(std::string userAnswer)
{
    // Удаление пробельных символов
    userAnswer.erase(std::remove(userAnswer.begin(),
userAnswer.end(), ' '), userAnswer.end());

    // Правильный ли ответ (второй параметр всегда равен 0)

```

```

        std::pair<bool, int32_t> correctness;
        correctness.second = 0;

        // Учитывается только первый символ введенной пользователем
строки
        correctness.first = (answer_.find(userAnswer[0]) !=
std::string::npos);

        return correctness;
}

```

questionanswer.h

```

#ifndef QUESTIONANSWER_H
#define QUESTIONANSWER_H

#include "avltree.h"
#include "icheckstrategy.h"

#include <string>
#include <fstream>
#include <cstdlib>
#include <algorithm>

class QuestionAnswer {
public:
    QuestionAnswer(std::string q, std::string a,
                    std::shared_ptr<ICheckStrategy> checkStr,
                    int8_t graphNumber);
    QuestionAnswer() {}
    ~QuestionAnswer();

    // Функция проверяет, является ли правильным введенный ответ.
    // Если нет, то указывается, с какого места ответ неправильный.
    std::pair<bool, int32_t> checkAnswer(std::string userAnswer);

    std::vector<std::string> getGraphFilePath() const;

    int8_t getGraphNumber() { return graphNumber_; }

    const std::string& getQuestion() const { return question_; }
    const std::string& getAnswer() const { return answer_; }

private:
    std::string question_;

```

```

        std::string answer_;

        std::shared_ptr<ICheckStrategy> checkStrategy_ = nullptr;

        int8_t graphNumber_;
    };

#endif // QUESTIONANSWER_H

questionanswer.cpp
#include "questionanswer.h"

QuestionAnswer::QuestionAnswer(std::string q, std::string a,
                                std::shared_ptr<ICheckStrategy>
checkStr,
                                int8_t graphNumber)
    : question_(q),
      answer_(a),
      checkStrategy_(checkStr),
      graphNumber_(graphNumber) {}

QuestionAnswer::~~QuestionAnswer() {}

std::pair<bool, int32_t> QuestionAnswer::checkAnswer(std::string
userAnswer) {
    return checkStrategy_>checkAnswer(userAnswer);
}

std::vector<std::string> QuestionAnswer::getGraphFilePath() const {
    std::vector<std::string> paths;

    // Создание изображения для первого графа
    std::system("Graphviz\\bin\\dot -Tpng -o graph.png graph.gv");
    paths.push_back("graph.png");

    if (graphNumber_ == 2) {
        std::system("Graphviz\\bin\\dot -Tpng -o graph_frame.png
graph_frame.gv");
        paths.push_back("graph_frame.png");
    }
    return paths;
}

```


questiongenerator.h

```
#ifndef QUESTIONGENERATOR_H
#define QUESTIONGENERATOR_H

#include <QDebug>

#include <memory>
#include <cstdlib>
#include <ctime>
#include <vector>

#include "avltree.h"
#include "questionanswer.h"
#include "icheckstrategy.h"
#include "fullmatch.h"
#include "oneof.h"

class QuestionGenerator {
    using AvlTreePtr = std::shared_ptr<AvlTree<char>>;
public:
    QuestionGenerator();
    ~QuestionGenerator();

    QuestionAnswer generateQA();

private:
    std::ofstream* qaFile_ = nullptr;
};

#endif // QUESTIONGENERATOR_H
```

questiongenerator.cpp

```
#include "questiongenerator.h"

QuestionGenerator::QuestionGenerator() {
    // Инициализировать генератор текущим временем
    std::srand(std::time(nullptr));

    // Файл, в котором будут храниться вопросы и ответы
    qaFile_ = new std::ofstream("./qa.txt");
}

QuestionGenerator::~~QuestionGenerator() {
    if (qaFile_>is_open())
```

```

        qaFile_->close();
    delete qaFile_;
}

```

```

QuestionAnswer QuestionGenerator::generateQA() {

    // Создание дерева для вопроса
    AvlTree<char> avlTree;

    // Элементы дерева, которые в нем имеются
    std::vector<bool> elems(26);

    // Количество элементов в дереве выбирается рандомно (от 5 до 15)
    int n = 5 + rand() % 11;
    // Сгенерированный символ
    char c;
    for (int i = 0; i < n; ++i) {
        do { // Если элемент уже есть в дереве, то выбираем другой
            c = (97 + rand() % 25);
        } while (elems[c - 97]);
        elems[c - 97] = true;
        avlTree.insert(c);
    }

    // Создание файла с описанием графа
    std::fstream dotFile("./graph.gv", std::ios_base::out);

    // Сохранение исходного дерева в файл
    avlTree.printDot(dotFile);
    dotFile.close();

    // Дерево, которое будет получено после выполнения операции
    AvlTree<char> avlTreeNext(avlTree);

    // Вставляемый элемент
    char insertElem;
    do { // Если элемент уже есть в дереве, то выбираем другой
        insertElem = (97 + rand() % 25);
    } while (elems[insertElem - 97]);

    // Удаляемый элемент (из тех, которые имеются в дереве)
    char removeElem;
    do { // Если элемента нет в дереве, то выбираем другой
        removeElem = (97 + rand() % 25);
    } while (elems[removeElem - 97] == false);
}

```

```

    } while (!elems[removeElem - 97]);

    // Объект, в котором будет храниться формулировка вопроса и ответ
на него
    QuestionAnswer qa;

    // Строки с вопросом и ответом
    std::string question;
    std::string answer;

    // Рандомно выбирается тип вопроса (10 типов вопросов)
    switch (rand() % 10 + 1) {
    // Как будет выглядеть дерево при вставке элемента N?
    case 1: {
        avlTreeNext.insert(insertElem);

        // Формулировка вопроса
        question = "Как будет выглядеть дерево при вставке \'";
        question += insertElem;
        question += "\'?\\nПеречислите элементы полученного дерева в
КЛП-порядке.";

        // Формирование ответа
        for (auto& e : avlTreeNext.getPrefixOrder())
            answer += e;
        qa = QuestionAnswer(question, answer,
                             std::make_shared<FullMatch>(answer), 1);

        // Запись условия вопроса
        *qaFile_ << "Дано АВЛ-дерево:\\n";
        avlTree.printToFile(*qaFile_);

        break;
    }
    // Как будет выглядеть дерево при удалении элемента N?
    // Перечислите элементы полученного дерева в КЛП-порядке
    case 2: {
        avlTreeNext.remove(removeElem);

        // Формулировка вопроса
        question = "Как будет выглядеть дерево при удалении \'";
        question += removeElem;
        question += "\'?\\nПеречислите элементы полученного дерева в
КЛП-порядке.";

```

```

// Формирование ответа
for (auto& e : avlTreeNext.getPrefixOrder())
    answer += e;
qa = QuestionAnswer(question, answer,
                    std::make_shared<FullMatch>(answer), 1);

// Запись условия вопроса
*qaFile_ << "Дано AVL-дерево:\n";
avlTree.printToFile(*qaFile_);

break;
}
// Перечислите узлы, в которых нарушится баланс при вставке
элемента N.
case 3: {
    avlTreeNext.insert(insertElem);

    // Формулировка вопроса
    question = "Перечислите узлы, в которых нарушится баланс при
вставке элемента \'";
    question += insertElem;
    question += "\'?\\nУкажите эти элементы в порядке возрастания.
(Таких элементов может не оказаться)";

    // Формирование ответа
    auto lbn = avlTreeNext.getLastBalancedNodes();
    std::sort(lbn.begin(), lbn.end());

    for (auto& elem : lbn)
        answer.push_back(elem);

    qa = QuestionAnswer(question, answer,
                        std::make_shared<FullMatch>(answer), 1);

    // Запись условия вопроса
    *qaFile_ << "Дано AVL-дерево:\n";
    avlTree.printToFile(*qaFile_);

    break;
}
// Перечислите узлы, в которых нарушится баланс при удалении
элемента N.
case 4: {

```

```

avlTreeNext.remove(removeElem);

// Формулировка вопроса
question = "Перечислите узлы, в которых нарушится баланс при
удалении элемента \';
question += removeElem;
question += "\'?\\nУкажите эти элементы в порядке возрастания.
(Таких элементов может не оказаться)";

// Формирование ответа
auto lbn = avlTreeNext.getLastBalancedNodes();
std::sort(lbn.begin(), lbn.end());

for (auto& elem : lbn)
    answer.push_back(elem);

qa = QuestionAnswer(question, answer,
                    std::make_shared<FullMatch>(answer), 1);

// Запись условия вопроса
*qaFile_ << "Дано АВЛ-дерево:\\n";
avlTree.printToFile(*qaFile_);

break;
}
// Перечислите в хронологическом порядке последовательность
вращений поддеревьев и их корень при вставке элемента N.
// Примечание: вращении может и не быть.
case 5: {
    avlTreeNext.insert(insertElem);

    // Формулировка вопроса
    question = "Перечислите в хронологическом порядке
последовательность вращений поддеревьев и их корень при вставке \';
    question += insertElem;
    question += "\'.\\nФормат ответа: направление_вращения (l или r)
корень.\\n"
                "Например, если произошел большой поворот налево
вокруг узла p (q - его правый сын), то необходимо написать: r q l p.\\n"
                "Примечание: вращений может и не быть";

    // Формирование ответа
    for (auto& p : avlTreeNext.getLastRotations()) {
        answer.push_back(p.first);
    }
}

```

```

        answer.push_back(p.second);
    }
    qa = QuestionAnswer(question, answer,
                        std::make_shared<FullMatch>(answer), 1);

    // Запись условия вопроса
    *qaFile_ << "Дано AVL-дерево:\n";
    avlTree.printToFile(*qaFile_);

    break;
}
// Перечислите в хронологическом порядке последовательность
вращений поддеревьев и их корень при удалении элемента N.
// Примечание: вращения может и не быть.
case 6: {
    avlTreeNext.remove(removeElem);

    // Формулировка вопроса
    question = "Перечислите в хронологическом порядке
последовательность вращений поддеревьев и их корень при удалении \';
    question += removeElem;
    question += "\'.\nФормат ответа: направление_вращения (l или r)
корень.\n"
                "Например, если произошел большой поворот налево
вокруг узла p (q - его правый сын), то необходимо написать: r q l p.\n"
                "Примечание: вращений может и не быть";

    // Формирование ответа
    for (auto& p : avlTreeNext.getLastRotations()) {
        answer.push_back(p.first);
        answer.push_back(p.second);
    }
    qa = QuestionAnswer(question, answer,
                        std::make_shared<FullMatch>(answer), 1);

    // Запись условия вопроса
    *qaFile_ << "Дано AVL-дерево:\n";
    avlTree.printToFile(*qaFile_);

    break;
}
// Как будет выглядеть дерево, если сначала вставить элемент N, а
затем удалить элемент M?
// Перечислите элементы полученного дерева в КЛП-порядке.

```

```

case 7: {
    avlTree.insert(insertElem);
    avlTree.remove(removeElem);

    // Формулировка вопроса
    question = "Как будет выглядеть дерево, если сначала вставить
\'";

    question += insertElem;
    question += "\', а затем удалить \'";
    question += removeElem;
    question += "\'? \nПеречислите элементы полученного дерева в
КЛП-порядке.";

    // Формирование ответа
    for (auto& e : avlTreeNext.getPrefixOrder())
        answer += e;
    qa = QuestionAnswer(question, answer,
                        std::make_shared<FullMatch>(answer), 1);

    // Запись условия вопроса
    *qaFile_ << "Дано AVL-дерево: \n";
    avlTree.printToFile(*qaFile_);

    break;
}
// Как будет выглядеть дерево, если сначала удалить элемент N, а
затем вставить элемент M?
// Перечислите элементы полученного дерева в КЛП-порядке.
case 8: {
    avlTree.remove(removeElem);
    avlTree.insert(insertElem);

    // Формулировка вопроса
    question = "Как будет выглядеть дерево, если сначала удалить
\'";

    question += removeElem;
    question += "\', а затем вставить \'";
    question += insertElem;
    question += "\'? \nПеречислите элементы полученного дерева в
КЛП-порядке.";

    // Формирование ответа
    for (auto& e : avlTreeNext.getPrefixOrder())
        answer += e;

```

```

qa = QuestionAnswer(question, answer,
                     std::make_shared<FullMatch>(answer), 1);

// Запись условия вопроса
*qaFile_ << "Дано AVL-дерево:\n";
avlTree.printToFile(*qaFile_);

break;
}
// Какой элемент необходимо вставить в дерево, чтобы его каркас
совпадал с тем, который изображен рядом?
// Если несколько возможных элементов, введите любой из них (один
элемент).
case 9: {
    // Создание файла с описанием каркаса графа
    std::fstream frameDotFile("./graph_frame.gv",
std::ios_base::out);

    AvlTree<char> avlTreeFrame(avlTree);
    avlTreeFrame.insert(insertElem);

    elems[insertElem - 97] = true;

    // Сохранение каркаса нового дерева в файл
    avlTreeFrame.printFrameDot(frameDotFile);
    frameDotFile.close();

    // Формулировка вопроса
    question = "Какой элемент необходимо вставить в дерево, чтобы
его каркас совпадал с тем, который изображен рядом?\n"
               "Если несколько возможных элементов, введите любой
из них (один элемент).";

    // Формирование ответа
    answer.push_back(insertElem);

    for (char c = 'a'; c <= 'z'; ++c) {
        if (!elems[c - 97]) {
            avlTreeNext.insert(c);
            elems[c - 97] = true;

            // Если при вставке элемента получается дерево с тем же
каркасом, то запоминаем его
            if (avlTreeFrame.haveSameFrame(avlTreeNext))

```



```

        answer.push_back(c);

        // Возвращение исходного дерева
        avlTreeNext = avlTree;
    }
}

qa = QuestionAnswer(question, answer,
                    std::make_shared<OneOf>(answer), 2);

// Запись условия вопроса
*qaFile_ << "Дано АВЛ-дерево и его каркас после некоторой
операции:\n";
avlTree.printToFile(*qaFile_);
avlTreeFrame.printFrameToFile(*qaFile_);

break;
}
// Какой элемент необходимо удалить из дерева, чтобы его каркас
совпадал с тем, который изображен рядом?
// Если несколько возможных элементов, введите любой из них (один
элемент).
case 10: {
    // Создание файла с описанием каркаса графа
    std::fstream                                frameDotFile("./graph_frame.gv",
std::ios_base::out);

    AvlTree<char> avlTreeFrame(avlTree);
    avlTreeFrame.remove(removeElem);

    elems[removeElem - 97] = false;

    // Сохранение каркаса нового дерева в файл
    avlTreeFrame.printFrameDot(frameDotFile);
    frameDotFile.close();

    // Формулировка вопроса
    question = "Какой элемент необходимо удалить из дерева, чтобы
его каркас совпадал с тем, который изображен рядом?\n"
               "Если несколько возможных элементов, введите любой
из них (один элемент).";

    // Формирование ответа
    answer.push_back(removeElem);

```

```

        for (char c = 'a'; c <= 'z'; ++c) {
            if (elems[c - 97]) {
                avlTreeNext.remove(c);
                elems[c - 97] = false;

                // Если при удалении элемента получается дерево с тем же
каркасом, то запоминаем его
                if (avlTreeFrame.haveSameFrame(avlTreeNext))
                    answer.push_back(c);

                // Возвращение исходного дерева
                avlTreeNext = avlTree;
            }
        }

        qa = QuestionAnswer(question, answer,
                             std::make_shared<OneOf>(answer), 2);

        // Запись условия вопроса
        *qaFile_ << "Дано АВЛ-дерево и его каркас после некоторой
операции:\n";
        avlTree.printToFile(*qaFile_);
        avlTreeFrame.printFrameToFile(*qaFile_);

        break;
    }
}

// Запись вопроса и ответа в файл
*qaFile_ << "Q: " << question << std::endl;
*qaFile_ << "A: " << answer << '\n' << std::endl;

qDebug() << "Question: " << qa.getQuestion().c_str();
qDebug() << "Answer:" << qa.getAnswer().c_str();

return qa;
}

```

ПРИЛОЖЕНИЕ Б

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

На рис. 2 представлено окно программы при старте.

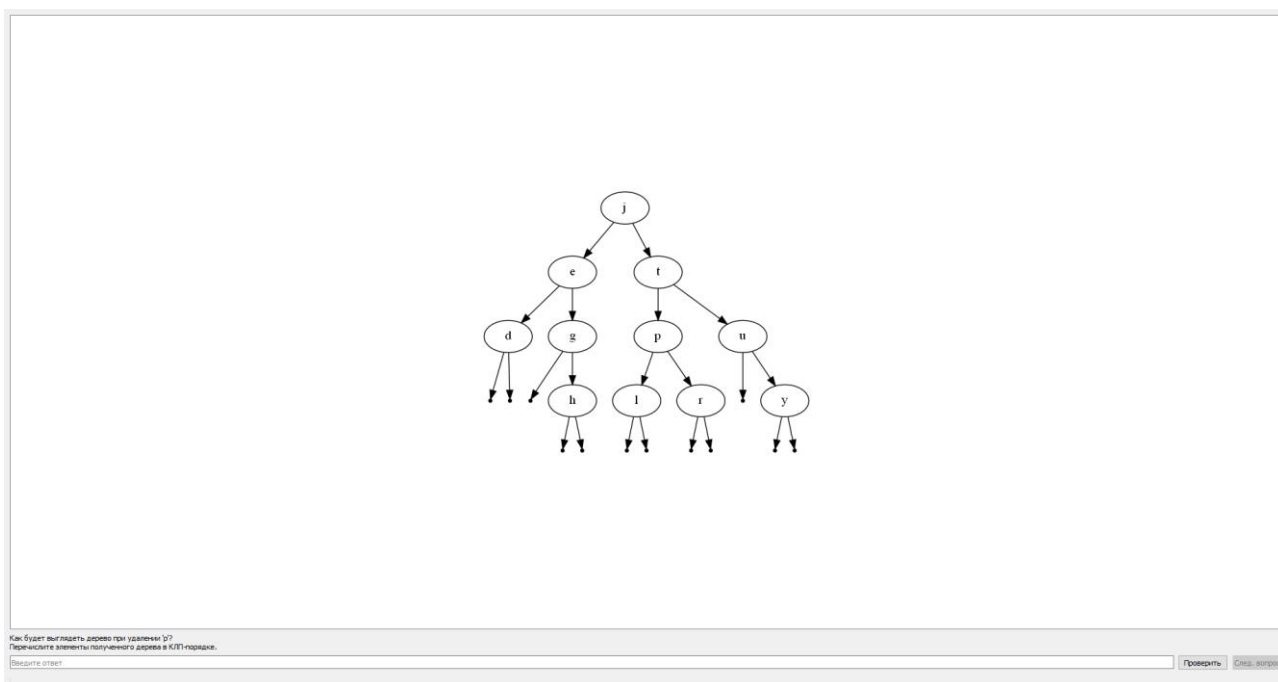


Рисунок 2 – Начало программы

На рис. 3 представлено окно программы при правильном ответе.

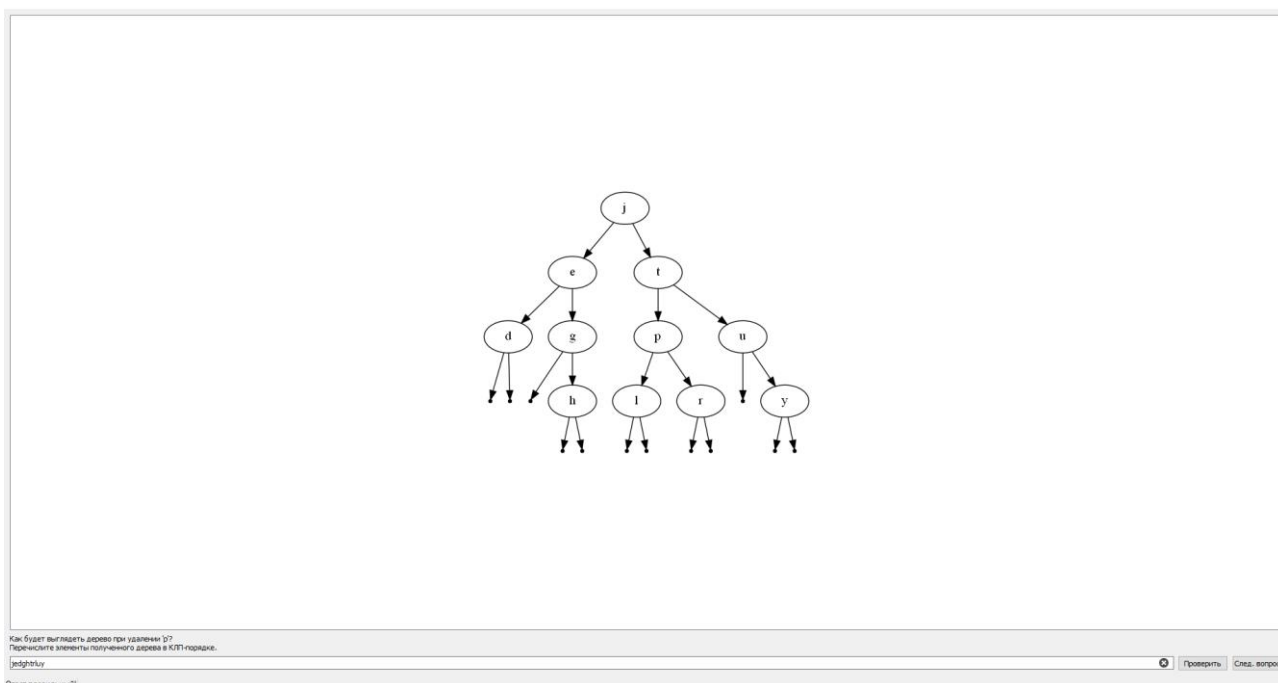


Рисунок 3 – Правильный ответ пользователя

На рис. 4 представлено окно программы при неправильном ответе пользователя.

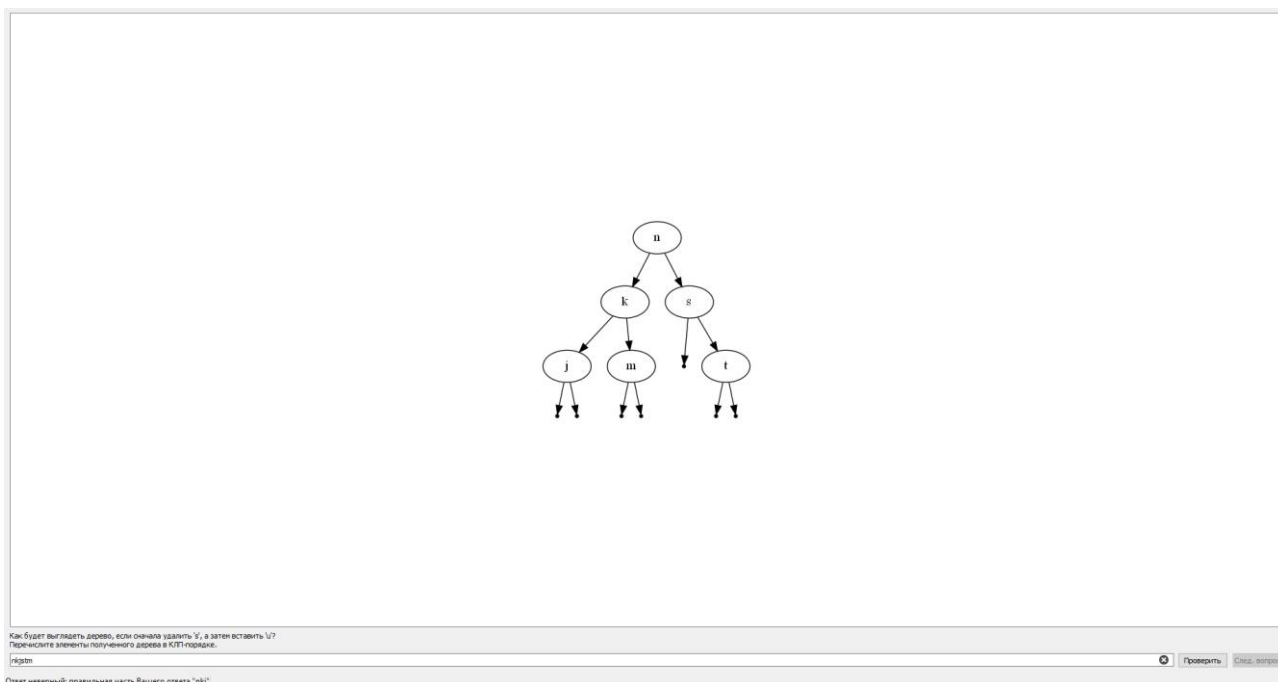


Рисунок 4 – Неправильный ответ пользователя

На рис. 5 представлено окно программы и генерируемый файл с описанием дерева, вопросом и ответом.

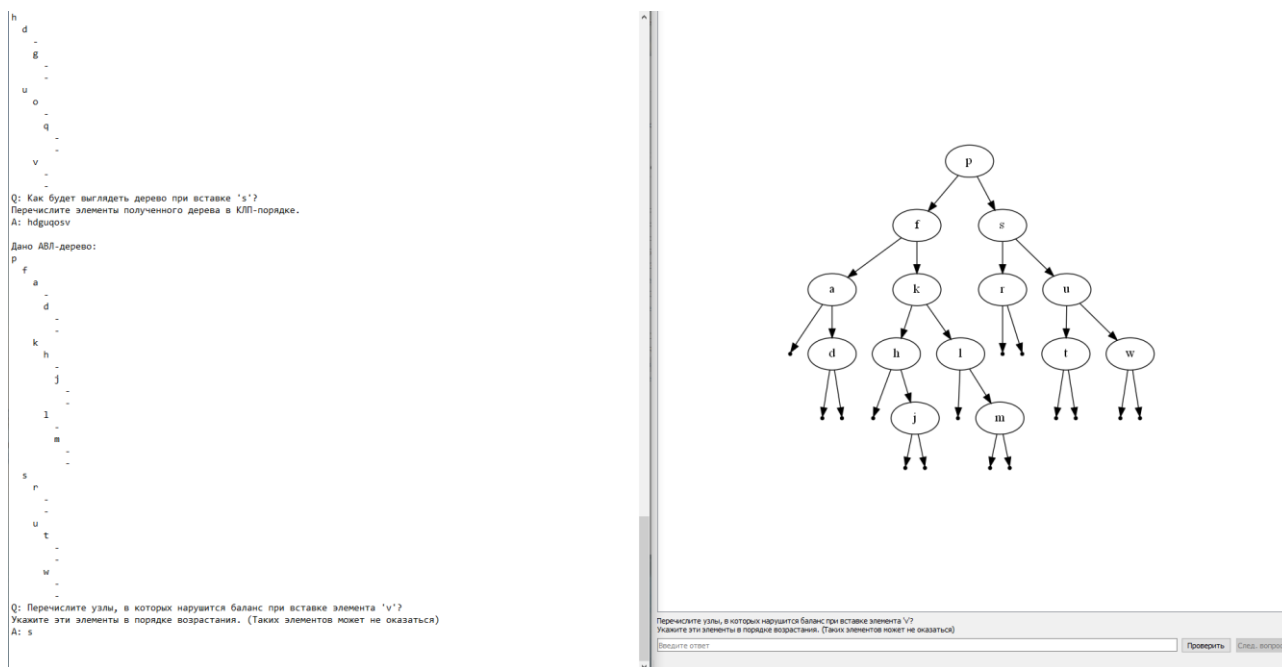


Рисунок 5 – Генерируемый файл и окно программы