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: Вставка и исключение элементов в АВЛ-деревьях – текущий контроль

Студентка гр. 9304 _____ Селезнёва А.В.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2020

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студентка Селезнёва А.В.

Группа 9304.

Тема работы: Вставка и исключение элементов в АВЛ-деревьях – текущий контроль.

Содержание пояснительной записки:
«Содержание», «Введение», «Формальная постановка задачи», «Описание алгоритмов», «Балансировка узлов», «Вставка ключей», «Удаление ключей», «Реализация программы», «Описание классов», «Описание функций», «Руководство пользователя», «Заключение», «Список использованных источников», «Приложение А – исходный код».

Предполагаемый объем пояснительной записки:
Не менее 20 страниц.

Дата выдачи задания: 23.11.2020

Дата сдачи реферата: 28.12.2020

Дата защиты реферата: 28.12.2020

Студентка _____ Селезнёва А.В.

Преподаватель _____ Филатов А.Ю.

АННОТАЦИЯ

В курсовой работе реализована программа, осуществляющая вставку и исключение элементов в структуре данных АВЛ-дерево.

Программа генерирует задания на вставку или удаление элементов в АВЛ-дереве. Создается два файла: в один из них записывается текст заданий, в другой – ответы на эти задания.

SUMMARY

In the course work, a program is implemented that inserts and excludes elements in the AVL-tree data structure.

The program generates tasks for inserting or deleting elements in the AVL tree. Two files are created: one of them records the text of tasks, the other – the answers to these tasks.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Требовалось написать программу, которая генерирует задания на вставку или удаление элементов в структуре данных АВЛ-дерево, выполняет их и записывает ответы на них в файл.

1. ФОРМАЛЬНАЯ ПОСТАНОВКА ЗАДАЧИ

Вариант 17.

АВЛ-деревья – вставка и исключение.

Текущий контроль. «Текущий контроль» – создание программы для генерации заданий с ответами к ним для проведения текущего контроля среди студентов. 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 (т.е. не должны быть вопросами по теории); ответы должны позволять удобную проверку правильности выполнения заданий.

2. ОПИСАНИЕ АЛГОРИТМОВ

2.1. Балансировка узлов

В процессе добавления или удаления узлов в АВЛ-дереве возможно возникновение ситуации, когда balance factor (разница между высотой правого и левого поддерева) некоторых узлов оказывается равными 2 или -2. Для исправления ситуации применяются повороты вокруг тех или иных узлов дерева.

Если balance factor равен -2, то проверяем левое поддерево. Если у левого поддерева проверяемого поддерева высота больше, чем у правого, то нам достаточно сделать один правый поворот, иначе нужно сделать последовательно сначала левый поворот для левого поддерева, а потом правый поворот для всего.

Если balance factor равен 2, то проверяем правое поддерево. Если у правого поддерева проверяемого поддерева высота больше, чем у левого, то нам достаточно сделать один левый поворот, иначе нужно сделать последовательно сначала правый поворот для правого поддерева, а потом левый поворот для всего.

2.2. Вставка ключей

Вставка нового ключа в АВЛ-дерево выполняется следующим образом: спускаемся вниз по дереву, выбирая правое или левое направление движения в зависимости от результата сравнения ключа в текущем узле и вставляемого ключа. После вставки нового ключа балансируем дерево, если это необходимо.

2.3. Удаление ключей

Удаление ключа из АВЛ-дерева выполняется следующим образом: находим узел p с заданным ключом k, в правом поддереве находим узел min с наименьшим ключом и заменяем удаляемый узел p на найденный узел min.

Если 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 – лист. В обоих этих случаях нужно просто

удалить узел *p* и вернуть в качестве результата указатель на левый дочерний узел узла *p*.

После удаления ключа выполняем балансировку, если это необходимо.

3. РЕАЛИЗАЦИЯ ПРОГРАММЫ

3.1. Описание классов

1. class Node – класс, описывающий узел дерева.

   Поля:

   std::shared_ptr<Node> left – указатель на левое поддерево;

   std::shared_ptr<Node> right – указатель на левое поддерево;

   int height – высота узла;

   int key – элемент, хранящийся в узле.

2. class AVL_Tree – класс, описывающий АВЛ-дерево.

   Поля:

   std::shared_ptr<Node> Head – указатель на корень дерева.

   Методы:

   void print_avl() – метод, который обходит дерево и выводит его на экран;

   std::shared_ptr<Node> get_head() – метод, возвращающий указатель на корень дерева;

   bool search_item(std::shared_ptr<Node> ptr, int k) – метод, который проверяет, присутствует ли элемент в дереве;

   std::shared_ptr<Node> create_AVL(std::vector<int> v) – метод для создания АВЛ-дерева;

   int bfactor(std::shared_ptr<Node> p) – метод, вычисляющий balance factor заданного узла;

   void fixheight(std::shared_ptr<Node> ptr) – метод, восстанавливающий корректное значение поля height;

   std::shared_ptr<Node> rotateright(std::shared_ptr<Node> p) – метод, осуществляющий правый поворот вокруг элемента p;

   std::shared_ptr<Node> rotateleft(std::shared_ptr<Node> q) – метод, осуществляющий левый поворот вокруг элемента q;

std::shared_ptr<Node> balance(std::shared_ptr<Node> ptr) – метод, осуществляющий балансировку узла ptr;

std::shared_ptr<Node> insert(std::shared_ptr<Node> p, int k) – метод для вставки элемента в дерево;

std::shared_ptr<Node> findmin(std::shared_ptr<Node> ptr) – метод для поиска узла с минимальным ключом в дереве;

std::shared_ptr<Node> removemin(std::shared_ptr<Node> p) – метод для удаления узла с минимальным ключом из дерева;

std::shared_ptr<Node> remove(std::shared_ptr<Node> p, int k) – метод для удаления ключа k из дерева.

3. class TK – класс для создания заданий текущего контроля.

   Поля:

   std::shared_ptr<AVL_Tree> tree – указатель на АВЛ-дерево;

   std::ofstream teacher – поток вывода в файл для проверяющего;

   std::ofstream student – поток вывода в файл для студента.

   Методы:

   void start_TR() – метод, считывающий основные данные и вызывающий генерацию выбранных заданий;

   void create_task_delete(std::vector<int>& v) – метод, создающий задания по удалению элемента из дерева;

   void create_task_insert(std::vector<int>& v) – метод, создающий задания по вставке элемента в дерево;

   void print_avl_teacher(std::shared_ptr<Node> ptr, int i = 0) – метод, записывающий ответы на задания в файл для проверяющего;

   void print_avl_student(std::shared_ptr<Node> ptr, int i = 0) – метод, записывающий задания в файл для студента.

3.2. Описание функций

int main() – вызывает метод класса ТК для генерации заданий текущего контроля.

4. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для запуска программы нужно прописать в терминале сначала команду make, а затем ./tk.

После запуска программы пользователю будет предложено ввести количество заданий для генерации. Это показано на рисунке 1:

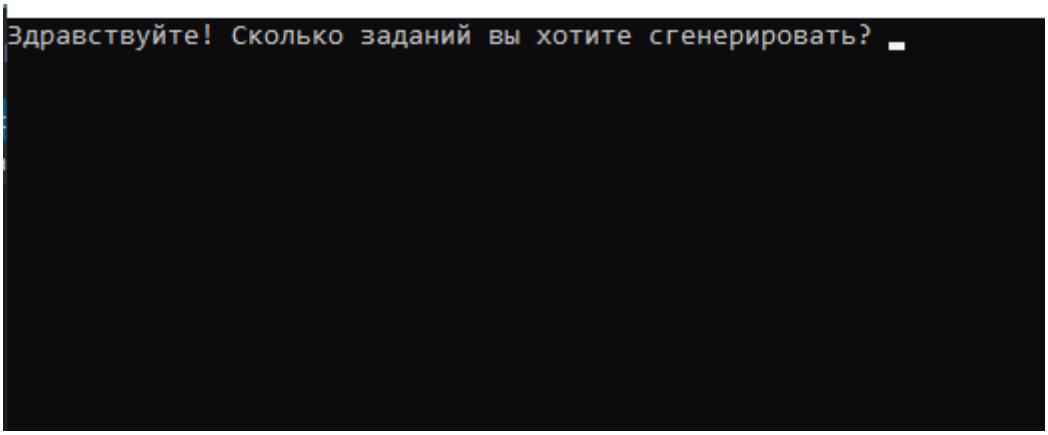

Рисунок 1 – Консоль при запуске программы

В данном случае было решено сгенерировать 3 задания. Далее пользователю будет предложено выбрать тип задания, а также ввести количество элементов в дереве, как показано на рисунке 2:

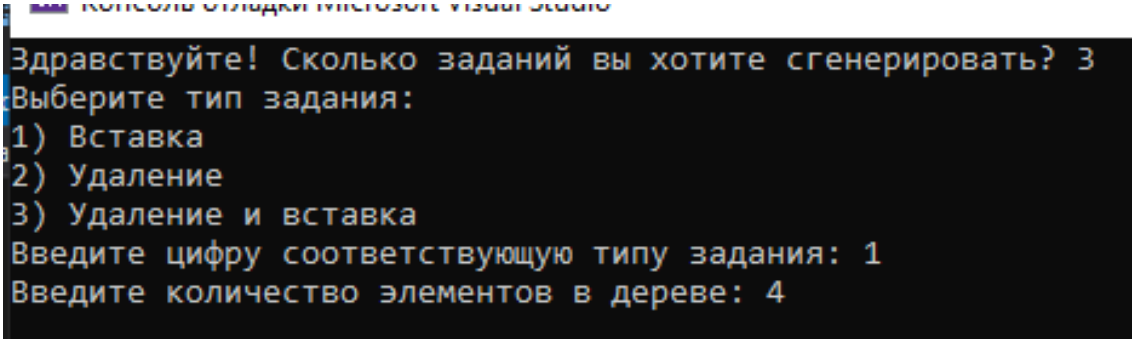

Рисунок 2 – Выбор типа задания

После этого создастся два файла: Student.txt и Teacher.txt. В первом из них находятся задания для студентов (рисунок 3), а во втором условия заданий и ответы на них для преподавателя (рисунок 4):

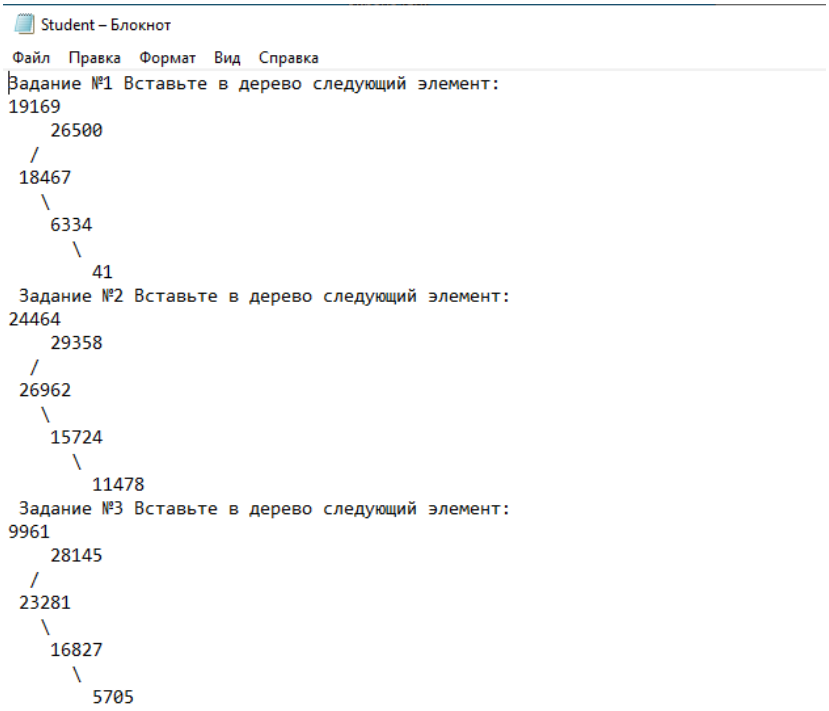

Рисунок 3 – Файл с заданиями для студента

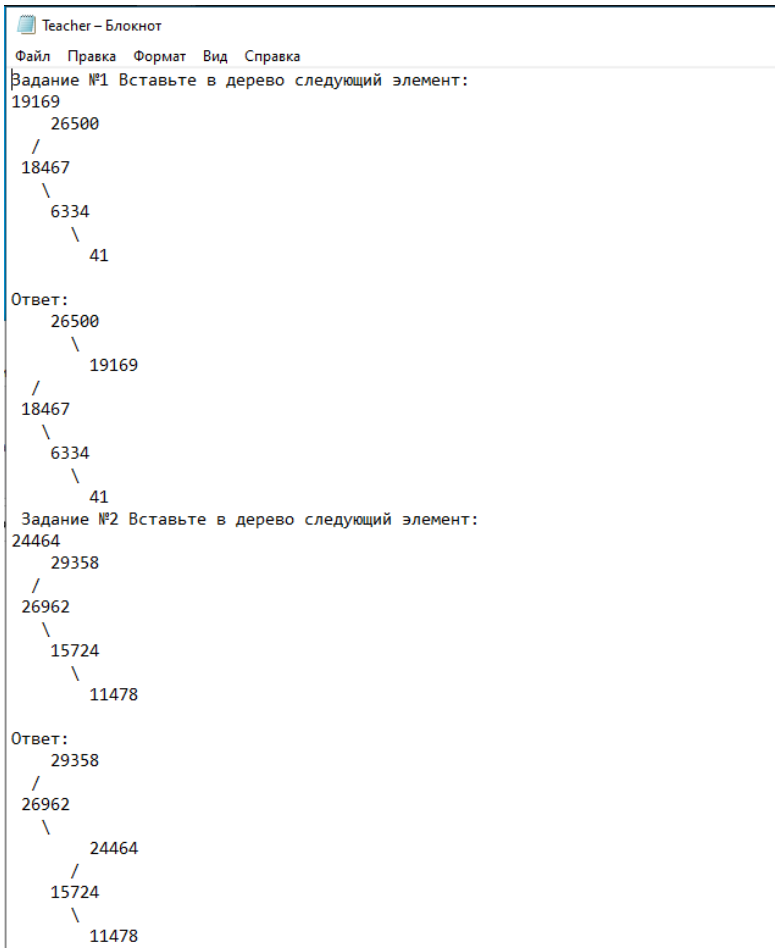

Рисунок 4 – Файл с ответами на задания для преподавателя

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была изучена и реализована структура данных АВЛ-дерево, а также операции вставки и удаления элементов для него.

Реализована программа на языке программирования C++, позволяющая генерировать задания для проведения текущего контроля.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. АВЛ-деревья // URL: https://habr.com/ru/post/150732/ (дата обращения: 27.12.2020);

2. Н. Вирт Алгоритмы и структуры данных. ДМК-Пресс, 2016 г.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

Название файла: avl.cpp

```
#include <iostream>
#include <memory>
#include <vector>
#include <algorithm>
#include <iomanip>
#include <fstream>
#include <string>

class AVL_Tree;
class TK;
class Node {
public:
     Node(int k): key(k) {
          height = 1;
     }
     Node(std::shared_ptr<Node> left, std::shared_ptr<Node>
right, int k) : left(left), right(right), key(k) {
          height = 1;
     }
private:
     friend class AVL_Tree;
     friend class TK;
     std::shared_ptr<Node> left = nullptr;
     std::shared_ptr<Node> right = nullptr;
     int height;
     int key;
};

class AVL_Tree {
public:
     AVL_Tree(std::vector<int> v) {
          std::sort(v.begin(), v.end());
          Head = this->create_AVL(v);
     }
     void print_avl() {
          print_node(this->Head);
     }
     std::shared_ptr<Node> get_head() {
          return this->Head;
     }
     bool search_item(std::shared_ptr<Node> ptr, int k) {
          if (ptr == nullptr) {
               return false;
          }
          else if (ptr->key > k) {
               return search_item(ptr->left, k);
```

```
                }
                else if (ptr->key < k) {
                    return search_item(ptr->right, k);
                }
                else {
                    return true;
                }
            }
            void inserting_elem(int k) {
                if (!search_item(this->Head, k)) {
                    insert(this->Head, k);
                    //std::cout << "insert\n";
                }
                else {
                    //std::cout << "The elem already exists\n";
                }
            }
            void delete_elem(int k) {
                if (search_item(this->Head, k)) {
                    Head = remove(this->Head, k);
                    //std::cout << "delete\n";
                }
                else {
                    //std::cout << "elem not found \n";
                }
            }

        private:
            std::shared_ptr<Node> Head;
            void print_node(std::shared_ptr<Node> ptr, int i = 0) {
                if (ptr != nullptr) {
                    if (ptr->right) {
                        print_node(ptr->right, i + 4);
                    }
                    if (i) {
                        std::cout << std::setw(i) << ' ';
                    }
                    if (ptr->right) {
                        std::cout << " /\n" << std::setw(i) << '
';
                    }
                    std::cout << ptr->key <<" " << ptr->height <<
"\n ";
                    if (ptr->left) {
                        std::cout << std::setw(i) << ' ' << "
\\\n";
                        print_node(ptr->left, i + 4);
                    }
                }
            }
            std::shared_ptr<Node> create_AVL(std::vector<int> v) {
                if (v.size() == 1) {
```

```
                    std::shared_ptr<Node> node =
std::make_shared<Node>(v[0]);
                    node->left = nullptr;
                    node->right = nullptr;
                    //node->height = 1;
                    return node;
               }
               else if (v.empty()) {
                    return nullptr;
               }
               else {
                    size_t i = v.size() / 2;
                    std::vector<int> v_left;
                    std::vector<int> v_right;
                    for (size_t z = 0; z < i; ++z) {
                         v_left.push_back(v.at(z));
                    }
                    for (size_t z = i+1 ; z < v.size(); ++z) {
                         v_right.push_back(v.at(z));
                    }
                    std::shared_ptr<Node> node =
std::make_shared<Node>(v[i]);
                    node->left = create_AVL(v_left);
                    node->right = create_AVL(v_right);
                    fixheight(node);
                    return node;
               }
          }
          int bfactor(std::shared_ptr<Node> p)
          {
               int lHeight = 0, rHeight = 0;
               if (p->left != nullptr) {
                    lHeight = p->left->height;
               }
               if (p->right != nullptr) {
                    rHeight = p->right->height;
               }
               //p->height = lHeight > rHeight ? lHeight + 1 :
rHeight + 1;
               return ( rHeight - lHeight);
          }

          void fixheight(std::shared_ptr<Node> ptr)
          {
               if (ptr->left) {
                    fixheight(ptr->left);
               }
               if (ptr->right) {
                    fixheight(ptr->right);
               }
               if (ptr->left == nullptr && ptr->right == nullptr) {
                    ptr->height = 1;
               }
```

```
                else if (ptr->right == nullptr) {
                    ptr->height = (ptr->left->height + 1);
                }
                else if (ptr->left == nullptr) {
                    ptr->height = (ptr->right->height + 1);
                }
                else {
                    ptr->height = ptr->left->height > ptr->right-
>height ? ptr->left->height : ptr->right->height;
                    ptr->height++;
                }

            }
            std::shared_ptr<Node> rotateright(std::shared_ptr<Node>
p) // правый поворот вокруг p
            {
                std::shared_ptr<Node> q = p->left;
                p->left = q->right;
                q->right = p;
                fixheight(p);
                fixheight(q);
                return q;
            }

            std::shared_ptr<Node> rotateleft(std::shared_ptr<Node> q)
// левый поворот вокруг q
            {
                std::shared_ptr<Node> p = q->right;
                q->right = p->left;
                p->left = q;
                fixheight(q);
                fixheight(p);
                return p;
            }

            std::shared_ptr<Node> balance(std::shared_ptr<Node> ptr)
// балансировка узла p
            {
                fixheight(ptr);
                if (bfactor(ptr) == 2)
                {
                    if (bfactor(ptr->right) < 0)
                        ptr->right = rotateright(ptr->right);
                    return rotateleft(ptr);

                }
                if (bfactor(ptr) == -2)
                {
                    if (bfactor(ptr->left) > 0)
                        ptr->left = rotateleft(ptr->left);
                    return rotateright(ptr);
                }
                return ptr; // балансировка не нужна
```

```
        }

        std::shared_ptr<Node> insert(std::shared_ptr<Node> p, int
k) // вставка ключа k в дерево с корнем p
        {
            if (p == nullptr)
                return std::make_shared<Node>(k);
            if (k < p->key)
                p->left = insert(p->left, k);
            else
                p->right = insert(p->right, k);
            return balance(p);
        }
        std::shared_ptr<Node> findmin(std::shared_ptr<Node> ptr)
{
            if(ptr == nullptr) {
                return nullptr;
            }
            return ptr->left ? findmin(ptr->left) : ptr;
        }
        std::shared_ptr<Node> removemin(std::shared_ptr<Node>
p){//, std::shared_ptr<Node> min) // удаление узла с минимальным
ключом из дерева p
            if (!p) {
                return nullptr;
            }
            else if (!p->left) {
                return p->right;
            }
            else {
                p->left = removemin(p->left);// , min);
                return balance(p);
            }
        }

        std::shared_ptr<Node> remove(std::shared_ptr<Node> p, int
k) // удаление ключа k из дерева p
        {
            if (k < p->key)
                p->left = remove(p->left, k);
            else if (k > p->key)
                p->right = remove(p->right, k);
            else //  k == p->key
            {
                if (p->right == nullptr) {
                    return p->left;
                }
                auto min = findmin(p->right);
                min->right = removemin(p->right);//, min);
                min->left = p->left;
                return balance(min);
            }
            return balance(p);
```

```
          }

     };

     class TK {
     public:
          TK() {
               teacher.open("Teacher.txt");
               student.open("Student.txt");
          }
          ~TK() {
               teacher.close();
               student.close();
          }
          void start_TR() {
               std::string q_task_ = "Здравствуйте! Сколько заданий
вы хотите сгенерировать? ";
               std::string type_task_ = "Выберите тип задания:\n1)
Вставка\n2) Удаление\n3) Удаление и вставка\nВведите цифру
соответствующую типу задания: ";
               std::string q_elem_in_tree_ = "Введите количество
элементов в дереве: ";
               std::string delete_str = " Удалите из дерева
следующий элемент: \n";
               std::string insert_str = " Вставьте в дерево
следующий элемент: \n";
               unsigned int q_task;
               unsigned int type_task;
               unsigned int q_elem_in_tree;
               std::cout << q_task_;
               std::cin >> q_task;
               std::cout << type_task_;
               std::cin >> type_task;
               std::cout << q_elem_in_tree_;
               std::cin >> q_elem_in_tree;
               if (q_task == 0) {
                    std::cout << "Количество заданий должно быть
больше 0";
               }
               else if (type_task != 3 && type_task != 1 &&
type_task != 2 ) {
                    std::cout << "Неопределенный тип задания";
               }

               else {


                    for (unsigned int i = 1; i <= q_task; ++i) {
                         std::vector<int> Avl;
                         for (unsigned int i = 0; i <
q_elem_in_tree;) {
                              int x;
```

```
                                   if (count(Avl.begin(), Avl.end(), x =
rand()) == 0) {
                                        Avl.push_back(x);
                                        ++i;
                                   }
                              }
                              tree = std::make_shared<AVL_Tree>(Avl);
                              if (type_task == 1 || (type_task == 3 &&
i%2 ==0)) {
                                   teacher << "Задание №" << i <<
insert_str;
                                   student << "Задание №" << i <<
insert_str;
                                   create_task_insert(Avl);

                              }
                              else if (type_task == 2 || (type_task == 3
&& i % 2 == 1)) {
                                   teacher << "Задание №" << i <<
delete_str;
                                   student << "Задание №" << i <<
delete_str;
                                   create_task_delete(Avl);
                              }
                              Avl.clear();
                         }

                    }

               }
          private:
               std::shared_ptr<AVL_Tree> tree;
               std::ofstream teacher;
               std::ofstream student;
               void create_task_delete(std::vector<int>& v) {
                    int k = (rand() % v.size());
                    teacher << v[k] << "\n";
                    student << v[k] << "\n";
                    this->print_avl_student(this->tree->get_head());
                    this->print_avl_teacher(this->tree->get_head());
                    this->tree->delete_elem(v[k]);
                    teacher << "\nОтвет:\n";
                    this->print_avl_teacher(this->tree->get_head());
               }
               void create_task_insert(std::vector<int>& v) {
                    int x;
                    do {
                         x = rand();
                    } while(count(v.begin(), v.end(), x) != 0);
                    teacher << x << "\n";
                    student << x << "\n";
                    this->print_avl_student(this->tree->get_head());
                    this->print_avl_teacher(this->tree->get_head());
```

```
            this->tree->inserting_elem(x);
            teacher << "\nОтвет:\n";
            this->print_avl_teacher(this->tree->get_head());

        }
        void print_avl_teacher(std::shared_ptr<Node> ptr, int i =
0) {
            if (ptr != nullptr) {
                if (ptr->right) {
                    print_avl_teacher(ptr->right, i + 4);
                }
                if (i) {
                    teacher << std::setw(i) << ' ';
                }
                if (ptr->right) {
                    teacher << " /\n" << std::setw(i) << ' ';
                }
                teacher << ptr->key << "\n ";
                if (ptr->left) {
                    teacher << std::setw(i) << ' ' << " \\\n";
                    print_avl_teacher(ptr->left, i + 4);
                }
            }
        }
        void print_avl_student(std::shared_ptr<Node> ptr, int i =
0) {
            if (ptr != nullptr) {
                if (ptr->right) {
                    print_avl_student(ptr->right, i + 4);
                }
                if (i) {
                    student << std::setw(i) << ' ';
                }
                if (ptr->right) {
                    student << " /\n" << std::setw(i) << ' ';
                }
                student << ptr->key << "\n ";
                if (ptr->left) {
                    student << std::setw(i) << ' ' << " \\\n";
                    print_avl_student(ptr->left, i + 4);
                }
            }
        }
    };


    int main() {
        setlocale(LC_ALL, "Russian");
        TK task;
        task.start_TR();
        return 0;
    }
```