

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 - вставка и исключение.
Текущий контроль.

Студент гр. 9381

Преподаватель

Птичкин С. А.

Фирсов М. А.

Санкт-Петербург

2020

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Птичкин С. А.

Группа 9381

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

Содержание пояснительной записки:

- титульный лист, лист задания, аннотация, содержание;
- формальная постановка задачи;
- описание алгоритма;
- описание структур данных и функций;
- описание интерфейса пользователя
- тестирование;
- программный код (в приложении);
- выводы.

Предполагаемый объём пояснительной записки:

Не менее 40 страниц.

Дата выдачи задания: 31.10.2020

Дата сдачи реферата: 15.10.2020

Дата защиты реферата: 16.10.2020

Студент Птичкин С. А.

Преподаватель Фирсов М. А.

АННОТАЦИЯ

В данной работе была создана программа, для выполнения текущего контроля по теме АФЛ-деревья - вставка/исключение. Была реализована структура данных, отражающая АВЛ-дерево и методы взаимодействия с ним. В программе был реализован пользовательский интерфейс, с помощью которого пользователь может создать тест вручную или предоставить генерацию программе. Был представлен исходный код и предоставлено тестирование программы.

SUMMARY

In this paper, a program was created to perform current control on the topic of AFL-trees-insert/exclude. A data structure was implemented that reflects the AVL tree and methods of interaction with it. The program has implemented a user interface that allows the user to create a test manually or provide generation to the program. The source code was presented and testing of the program was provided.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	5
1.ИСХОДНОЕ ЗАДАНИЕ	6
2.ВЫПОЛНЕНИЕ РАБОТЫ	6
2.1. Описание алгоритма.	6
2.2. Класс AVL_Tree.	7
2.3. Основные функции.	10
2.4. Описание интерфейса.	13
3.ТЕСТИРОВАНИЕ	17
ЗАКЛЮЧЕНИЕ.	23
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.	23
ПРИЛОЖЕНИЕ А.	24

ВВЕДЕНИЕ

Цель работы.

Разработка программы для генерации заданий с ответами к ним для проведения текущего контроля среди студентов по теме “АВЛ-деревья - вставка и исключение”.

Задачи.

- Изучение языка программирования C++;
- Изучение структуры данных “АВЛ-дерево”;
- Изучение алгоритмов обработки АВЛ-дерева;
- Написание исходного кода программы;
- Программирование генерации заданий для студентов по данной теме;
- Сборка программы;
- Тестирование программы;

Основные теоретические положения.

Двоичное дерево поиска — это двоичное дерево, для которого выполняются следующие дополнительные условия (свойства дерева поиска):

- Оба поддерева — левое и правое — являются двоичными деревьями поиска.
- У всех узлов левого поддерева произвольного узла X значения ключей данных меньше либо равны, нежели значение ключа данных самого узла X .
- У всех узлов правого поддерева произвольного узла X значения ключей данных больше либо равны, нежели значение ключа данных самого узла X .

Очевидно, данные в каждом узле должны обладать ключами, на которых определена операция сравнения меньше.

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

1.ИСХОДНОЕ ЗАДАНИЕ

Реализовать класс АВЛ-дерева и методы работы с ним. Написать программу, генерирующую задания для текущего контроля и ответы к ним. Реализовать пользовательский интерфейс с возможностью создания заданий вручную, генерации заданий программой и сохранения данных в файл.

2.ВЫПОЛНЕНИЕ РАБОТЫ

2.1. Описание алгоритма.

Алгоритм добавления элемента:

Вставка нового элемента в АВЛ-дерево выполняется так же, как это делается в простых деревьях поиска: спускаемся вниз по дереву, выбирая правое или левое направление движения в зависимости от результата сравнения ключа в текущем узле и вставляемого ключа. Единственное отличие заключается в том, что при возвращении из рекурсии (т.е. после того, как элемент добавлен либо в правое, либо в левое поддерево, и это дерево сбалансировано) выполняется балансировка текущего узла.

Балансировка дерева происходит, когда разница между высотами поддеревьев одного элемента становится равной 2. В таком случае, в зависимости от конфигурации, необходимо произвести серию вращений, как это показано на рис.1.

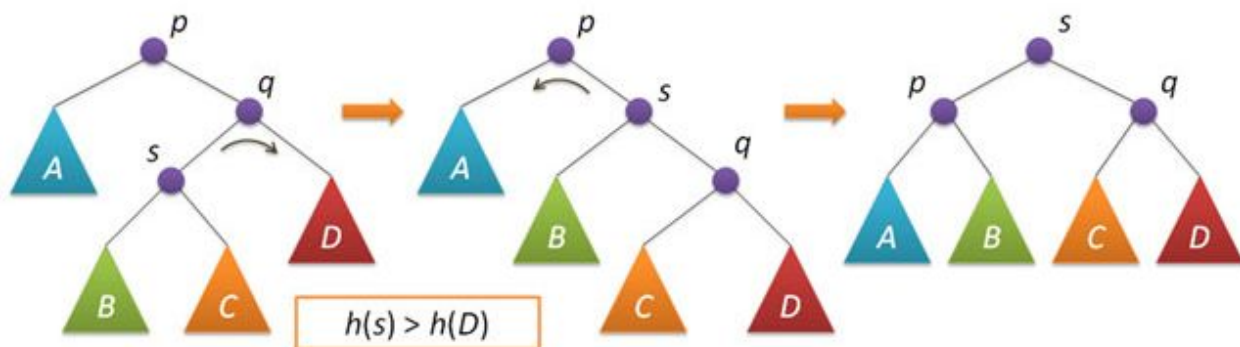


Рис.1 Балансировка дерева с помощью вращений

Алгоритм исключения элемента:

Начинаем рекурсивный поиск элемента с заданным ключём. При нахождении нужного элемента сохраняем его поддеревья, удаляем сам элемент, затем в правом поддереве находим минимальный элемент и заменяем удалённый на него. При каждом выходе из рекурсии ребалансируем дерево. На рис.2 наглядно показан процесс удаления элемента.

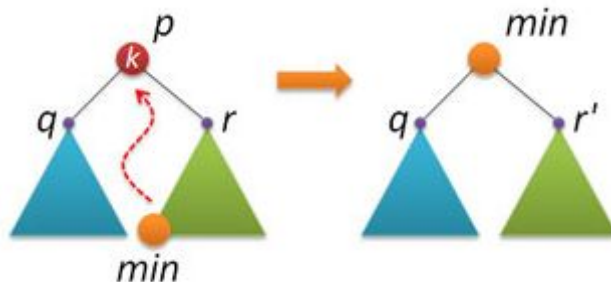


Рис.2 Удаление элемента

2.2. Класс AVL_Tree.

Объекты данного класса представляют собой AVL-дерево. У объекта есть приватные поля: `int height`, хранящий высоту дерева, `int key`, хранящий значение ключа, `AVL-Tree* left` и `AVL-Tree* right`, хранящие указатели на поддеревья. Также у класса определены некоторые методы для работы с ним.

Методы класса AVL-Tree:

1) AVL_Tree::AVL_Tree(int key)

Конструктор класса AVL-дерева. Получает на вход значение ключа k, поле height инициализируется единицей, поля left и right - нулевыми указателями.

2) bool AVL_Tree::find_key(int k)

Данный метод предназначен для проверки наличия в дереве элемента с заданным ключём. На вход передаётся сам ключ k. Функция рекурсивно обходит дерево и возвращает true при нахождении, false при отсутствии.

3) int AVL_Tree::get_height()

Данный метод предназначен для получения значения из поля height. Метод ничего не принимает и возвращает поле height объекта класса.

4) int AVL_Tree::bfactor()

Данный метод возвращает разницу между высотами поддеревьев у элемента дерева. Метод ничего не принимает на вход.

5) void AVL_Tree::fixheight()

Данный метод предназначен для обновления высоты у каждого элемента дерева. Метод ничего не принимает на вход и ничего не возвращает.

6) AVL_Tree* AVL_Tree::rotateright(ostream* stream)

Данный метод производит поворот вправо. На вход подаётся адрес потока вывода stream, в который выводится промежуточная информация, по умолчанию значение - nullptr. Метод возвращает указатель на элемент, полученный после поворота.

7) AVL_Tree* AVL_Tree::rotateleft(ostream* stream)

Данный метод производит поворот влево. На вход подаётся адрес потока вывода stream, в который выводится промежуточная информация, по умолчанию значение - nullptr. Метод возвращает указатель на элемент, полученный после поворота.

8) AVL_Tree* AVL_Tree::balance(ostream* stream)

Данный метод предназначен для выравнивания дерева по высоте после вставки/исключения элемента. На вход подаётся адрес потока вывода stream, которая в дальнейшем передаётся в методы левого и правого поворота. Функция возвращает указатель на элемент, относительно которого производилась балансировка.

9) AVL_Tree* AVL_Tree::insert(int k, ostream* stream)

Метод предназначен для вставки элемента в дерево. Метод принимает на вход ключ k и в зависимости от того меньше или больше он, чем значение текущего элемента вызывает рекурсивно этот же метод для своего левого или правого поддерева. Также на вход подаётся адрес потока вывода stream, в который выводится промежуточная информация, по умолчанию значение - nullptr. Дойдя до нулевого указателя происходит создание нового элемента. Функция сначала возвращает указатель на новый элемент, а затем возвращаемое значение функции балансировки.

10) AVL_Tree* AVL_Tree::findmin()

Данный метод рекурсивно обходит дерево и возвращает указатель на минимальный элемент дерева. Функция ничего не принимает на вход.

11) AVL_Tree* AVL_Tree::removemin()

Данный метод находит минимальный элемент дерева и удаляет его. Входные параметры отсутствуют. Метод возвращает указатель на правое поддерево.

12) AVL_Tree* AVL_Tree::remove(int k, ostream* stream)

Данный метод предназначен для удаления из дерева элемента с ключём k. Метод принимает на вход сам ключ k и адрес потока вывода stream, в который выводится промежуточная информация, по умолчанию значение - nullptr. Метод возвращает возвращаемое значение метода балансировки.

13) void AVL_Tree::destroy()

Метод предназначен для рекурсивного удаления дерева. Функция проходит всё дерево и очищает динамически выделенную память под элементы. Метод ничего не принимает и ничего не возвращает.

14) void AVL_Tree::print_tree(ostream* stream, int depth)

Данный метод предназначен для печати дерева в переданный поток ввода. Функция принимает указатель на поток stream и глубину рекурсивного погружения, которая отражает смещение элементов по высоте. Метод ничего не возвращает.

2.3. Основные функции.

Далее описаны функции непосредственно реализующие генерацию заданий для текущего контроля, а также пользовательский интерфейс.

1) int input_num(string message)

Функция принимает на вход сообщение, выводимое пользователю. Функция предназначена для корректного считывания числа из потока ввода. На вход принимается строка с сообщением пользователю, что ему делать. Объявляется переменная для записи числа и выделяется буфер на 10 символов. Затем из cin считывается 10 символов в буфер. Далее в цикле из данного буфера считывается число функцией sscanf. Пока функция не вернёт 1 - количество верно считанных аргументов, ввод не прекратится. Когда наконец число считается, оно возвращается функцией. Память, выделенная под буфер очищается.

2) void clear_memory(AVL_Tree mass_trees, int* mass_action, int count_of_trees)**

Данная функция предназначена для очистки памяти, выделенной под АЛВ-дерева и массив ключей преобразования. На вход подаются указатели на эти массивы(mass_trees mass_action) и количество созданных

деревьев(count_of_trees). В цикле последовательно запускается рекурсивный метод очистки каждого дерева, очищается память под сам массив указателей на деревья и массив ключей преобразований. Функция ничего не возвращает.

3) void save_tasks(AVL_Tree mass_trees, int* mass_action, int count_of_insert, int count_of_exceptions)**

Функция предназначена для сохранения сгенерированных заданий в файлы. На вход подаётся массив указателей на деревья(mass_trees), массив ключей преобразования(mass_action), количество вставок и исключений элементов(count_of_insert и count_of_exceptions). Сначала пользователь вводит имя файла для сохранения заданий. Файл открывается и туда записывается задание и соответствующее ему дерево. Затем пользователь вводит имя файла для сохранения ответов. Сначала производится преобразование дерева, для генерации ответа, при этом вся промежуточная информация вносится в файл. Затем печатается итоговое дерево, которое является ответом. Функция прекращает свою работу, ничего не возвращая.

4) AVL_Tree* tree_generator()

Данная функция предназначена для генерации случайных АВЛ-деревьев. Входных параметров нет. Объявляется указатель на дерево. Сначала при помощи функции rand() генерируется количество элементов дерева, затем в цикле, также при помощи rand(), генерируется необходимое число элементов, которые вставляются в дерево. Затем функция возвращает указатель на созданное дерево.

5) AVL_Tree* manual_generator()

Данная функция предназначена для создания АВЛ-дерева вручную пользователем. На вход функция ничего не принимает. С самого начала пользователь попадает в меню, где он может выбрать 1 из 3 действий:

вставить в дерево элемент, удалить элемент, законить создание дерева. В первых двух случаях требуется ввести ключ. Корректность ввода данных обеспечивается функцией `input_num()`. После каждого изменения дерево выводится в консоль. Функция завершает свою работу после введения команды 3 и возвращает указатель на созданное дерево.

6) **int user_interface()**

Данная функция реализует пользовательский интерфейс, а также вызывает все необходимые функции для генерации заданий текущего контроля. На вход функции подаётся её режим работы(mode). Значение 1, переменной mode означает рандомную генерацию заданий, значение 2 - генерацию вручную. В начале функция запрашивает у пользователя количество заданий на вставку и исключение элемента. Если оба некорректны - возвращает 1, что означает выход в главное меню. Иначе динамически выделяется память под массивы деревьев(`mass_trees`) и ключей преобразования(`mass_action`), нужного размера, которые нужны для задания. Затем в цикле последовательно вызывается функция генерации деревьев(вручную или рандомно) для вставки элемента. Сразу же после генерации дерева, с помощью функции происходит генерация(пользователем или рандомно) ключа для вставки(`rand_action`), который записывается в массив `mass_action`. Аналогичные действия происходят для деревьев для исключения элементов. Параллельно с этим в консоль выводятся промежуточные данные, сами задания и соответствующие деревья. В конце пользователь попадает в меню, где выбирает сохранить ли данные, продолжить или выйти из программы. Соответствующая функция вызывается. Функция возвращает 1, для выхода в главное меню, и 0 - для завершения программы.

7) **int main()**

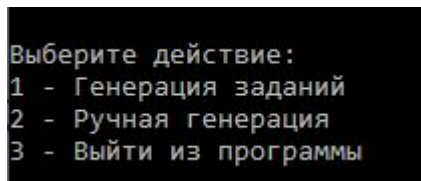
Функция не принимает никаких параметров. Данная функция

предназначена для стартового диалога с пользователем. Вначале инициализируется генератор случайных чисел, для рандомной генерации деревьев, затем идёт смена локализации на русскую. Далее идёт выбор начала работы или выхода из программы. За корректность введенных данных отвечает функция `input_num`. Ввод команды 1 вызывает функцию пользовательского интерфейса с входным параметром 1(рандомная генерация), команда 2, тоже самое, только с параметром 2(ручная генерация), команда 3 - выход из программы. Возвращаемое значение функции `user_interface()` определяет, будет ли продолжаться работа или программа завершится.

2.4. Описание интерфейса.

1) Стартовое меню.

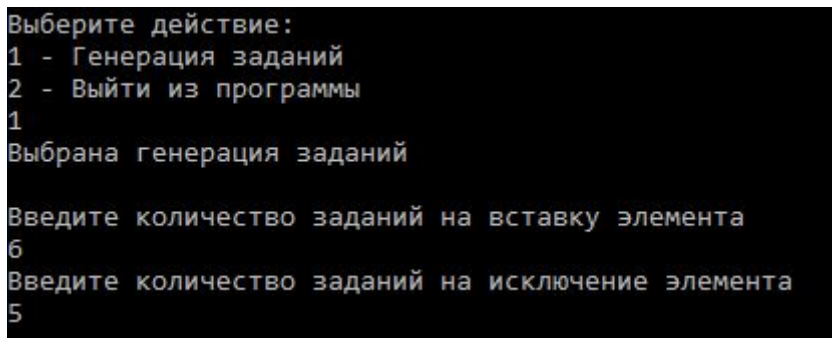
Здесь пользователь выбирает приступить ли к определённой генерации заданий или выйти из программы. Выбор осуществляется вводом соответствующей команды.



```
Выберите действие:  
1 - Генерация заданий  
2 - Ручная генерация  
3 - Выйти из программы
```

2) Выбор количества заданий.

После выбора в главном меню генерации заданий, у пользователя запрашивается количество заданий на вставку и удаление элемента.



```
Выберите действие:  
1 - Генерация заданий  
2 - Выйти из программы  
1  
Выбрана генерация заданий  
Введите количество заданий на вставку элемента  
6  
Введите количество заданий на исключение элемента  
5
```

3) Ручная генерация заданий.

В этом случае пользователь попадает в редактор деревьев. Для каждого задания пользователю придётся создавать дерево, а затем выбирать ключ для определённого задания. Функционал редактора определяется вставкой элемента в дерево, удаление элемента и завершение создания дерева.

```
Создание дерева для 1 задачи по вставке  
Введите команду:  
1 - Вставка элемента  
2 - Удаление элемента  
3 - Закончить создание дерева
```

После каждого действия дерево выводится в консоль. Выбор ключа осуществляется с помощью клавиатуры.

```
Текущее состояние дерева:  
3  
  2  
Введите команду:  
1 - Вставка элемента  
2 - Удаление элемента  
3 - Закончить создание дерева  
1  
Введите ключ  
4  
Вставка элемента 4  
Текущее состояние дерева:  
4  
3  
  2
```

После завершения создания дерева и введения ключа для задачи, на консоль выводится итоговое задание.

```
конец создания дерева  
Введите ключ элемента вставки, для генерации задания.  
1  
Итоговое задание 1: как будет выглядеть следующее дерево после вставки элемента 1 ?  
4  
3  
  2
```

4) Рандомная генерация заданий.

Пользователю выводятся сгенерированные задания вместе с промежуточными данными генерации.

```
Введите количество заданий на исключение элемента
1
-----
Промежуточные данные:
Вставка элемента 48
Вставка элемента 18
Вставка элемента 3
Производится поворот вправо

Задание 1: как будет выглядеть следующее дерево после вставки элемента 24 ?

    48
   / \
  18  3
```

5) Выбор дальнейших действий.

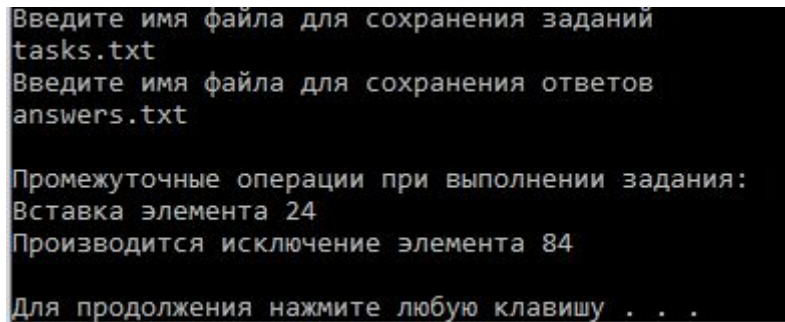
После генерации заданий пользователь попадает в меню выбора дальнейших действий. Введя соответствующую команду пользователь может сохранить данные или не сохранять, продолжить или завершить работу.

```
Нажмите ENTER, чтобы продолжить

Выберите дальнейшее действие:
1 - сохранить задания с ответами в файл и продолжить
2 - сохранить задания с ответами в файл и выйти
3 - продолжить без сохранения
4 - выйти без сохранения
```

6) Сохранение данных и выход из программы.

В случае выбора сохранения данных пользователем, ему предлагается ввести имена файлов сохранения заданий и ответов к ним. Также на консоль выводятся промежуточные операции, выполненные в ходе вычисления ответов. Выход из программы осуществляется после нажатия пользователем любой клавиши.



```
Введите имя файла для сохранения заданий
tasks.txt
Введите имя файла для сохранения ответов
answers.txt

Промежуточные операции при выполнении задания:
Вставка элемента 24
Производится исключение элемента 84

Для продолжения нажмите любую клавишу . . .
```


3.ТЕСТИРОВАНИЕ

1) Вставка элементов.

```
Вставка элемента 2
2
Вставка элемента 1
2
  1
Вставка элемента 3
  3
2
  1
Вставка элемента 4
    4
  3
2
  1
```

2) Поворот влево.

```
Вставка элемента 1
1
Вставка элемента 2
  2
1
Вставка элемента 3
Производится поворот влево
  3
2
  1
```

3) Поворот вправо.

```
Вставка элемента 3
3
Вставка элемента 2
3
  2
Вставка элемента 1
Производится поворот вправо
  3
2
  1
```

4) Двойной поворот(RL).

```
      5
     3
    2
   1
Вставка элемента 4
Производится поворот вправо
Производится поворот влево
      5
     4
    3
   2
  1
Для продолжения нажмите любую клавишу . . .
```

5) Двойной поворот(LR).

```
      5
     4
    3
   1
Вставка элемента 2
Производится поворот влево
Производится поворот вправо
      5
     4
    3
   2
  1
Для продолжения нажмите любую клавишу
```

6) Удаление листа.

```
Вставка элемента 2
2
Вставка элемента 1
2
  1
Вставка элемента 3
  3
2
  1
Производится исключение элемента 1
  3
2
Для продолжения нажмите любую клавишу . . .
```

7) Удаление корня.

```
Вставка элемента 2
2
Вставка элемента 1
2
  1
Вставка элемента 3
  3
2
  1
Производится исключение элемента 2
3
  1
Для продолжения нажмите любую клавишу
```

8) Некорректный ввод команды.

```
Выберите действие:
1 - Генерация заданий
2 - Выйти из программы
йцва
Ввод некорректный!

Выберите действие:
1 - Генерация заданий
2 - Выйти из программы
```

9) Некорректное количество заданий.

```
Выбрана генерация заданий

Введите количество заданий на вставку элемента
-1
Введите количество заданий на исключение элемента
0

Выберите действие:
1 - Генерация заданий
2 - Выйти из программы
```

10) Ввод отсутствующей команды.

```
Выберите дальнейшее действие:
1 - сохранить задания с ответами в файл и продолжить
2 - сохранить задания с ответами в файл и выйти
3 - продолжить без сохранения
4 - выйти без сохранения
5
Команда не распознана!

Выберите дальнейшее действие:
1 - сохранить задания с ответами в файл и продолжить
2 - сохранить задания с ответами в файл и выйти
3 - продолжить без сохранения
4 - выйти без сохранения
```

11) Ручное создание заданий.

```
Введите команду:
1 - Вставка элемента
2 - Удаление элемента
3 - Закончить создание дерева

1
Введите ключ
3
Вставка элемента 3
Текущее состояние дерева:
  3
2
  1

Введите команду:
1 - Вставка элемента
2 - Удаление элемента
3 - Закончить создание дерева

3
Конец создания дерева
Введите ключ элемента вставки, для генерации задания.
4

Итоговое задание 1: как будет выглядеть следующее дерево после вставки элемента 4 ?

  3
2
  1
```

12) Генерация заданий.

```
Промежуточные данные:
Вставка элемента 37
Вставка элемента 95
Вставка элемента 87
Производится поворот вправо
Производится поворот влево
Вставка элемента 56
Вставка элемента 38
Производится поворот вправо
Производится поворот влево
Вставка элемента 58
Производится поворот влево
Производится поворот вправо

Задание 1: как будет выглядеть следующее дерево после вставки элемента 2 ?

    95
   87
    58
56
   38
    37

Промежуточные данные:
Вставка элемента 89
Вставка элемента 91
Вставка элемента 46
Вставка элемента 49
Вставка элемента 10

Задание 2: как будет выглядеть следующее дерево после исключения элемента 10 ?

    91
   89
    49
   46
    10

Нажмите ENTER, чтобы продолжить
```

13) Сохранение заданий в файл.

Текущий контроль. Задания.

Задание 1: как будет выглядеть следующее дерево после вставки элемента 2 ?

```
      95
     /
    87
   /
  58
 /
56
/
38
/
37
```

Задание 2: как будет выглядеть следующее дерево после исключения элемента 10 ?

```
      91
     /
    89
   /
  49
 /
46
/
10
```

14) Сохранение ответов в файл.

Текущий контроль. Ответы.

Задание 1.

Промежуточные операции при выполнении задания:

Вставка элемента 2

Производится поворот вправо

Ответ:

```
      95
     /
    87
   /
  58
 /
56
/
38
/
37
/
2
```

Задание 2.

Промежуточные операции при выполнении задания:

Производится исключение элемента 10

Ответ:

```
      91
     /
    89
   /
  49
 /
46
```

ЗАКЛЮЧЕНИЕ.

Была написана программа, генерирующая задания для текущего контроля по вставке/исключению элементов АЛВ-деревьев. Также был написан удобный пользовательский интерфейс с возможностью сохранить задания и ответы в файл. Программа была протестирована, были обработаны все исключительные ситуации. Программный код можно посмотреть в приложении А.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.

1. Описание АЛВ-дерева и алгоритмов работы с ним // URL: <https://habr.com/ru/post/150732/>
2. Теоретическая информация об АВЛ-деревьях // URL: <https://ru.wikipedia.org/wiki/АВЛ-дерево>

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл AiSD_curswork.cpp:

```
#include "stdafx.h"
#include <string>
#include <iostream>
#include <ctime>
#include "AVL_Tree.h"
using namespace std;

int input_num(string message) { //функция корректного считывания числа
    int num = 0;
    cout << message << '\n';
    char* input = new char[10];
    fgets(input, 10, stdin);
    while (sscanf_s(input, "%d", &num) != 1) {
        cout << "Ввод некорректный!\n" << message << '\n';
        fgets(input, 10, stdin);
    }
    delete[] input;
    return num;
}

void clear_memory(AVL_Tree** mass_trees, int* mass_action, int count_of_trees)
{ //очистка памяти
    for (int i = 0; i < count_of_trees; i++) {
        mass_trees[i]->destroy();
    }
    delete mass_action;
}

void save_tasks(AVL_Tree** mass_trees, int* mass_action, int count_of_insert, int
count_of_exceptions) { //сохранение данных в файл
    char* file_name = new char[256];
    string insert_str = ": как будет выглядеть следующее дерево после вставки
элемента ";
    string except_str = ": как будет выглядеть следующее дерево после исключения
элемента ";
    cout << "Введите имя файла для сохранения заданий\n";
    cin >> file_name;
    getchar(); //вытаскиваем символ переноса строки из потока
    fstream tasks_file;
    tasks_file.open(file_name, fstream::out); //открытие или создание файла на
запись заданий
    memset(file_name, '\0', 256);
    tasks_file << "                                Текущий контроль.
Задания.\n\n";
```



```

        for (int i = 0; i < count_of_insert; i++) {
            tasks_file << "Задание " << i + 1 << insert_str << mass_action[i] << "
?\n\n"; //запись вопроса
            mass_trees[i]->print_tree(&tasks_file);          //запись дерева
            tasks_file << '\n';
        }
        for (int i = count_of_insert; i < count_of_exceptions+ count_of_insert; i++)
        {
            tasks_file << "Задание " << i + 1 << except_str << mass_action[i] << "
?\n\n"; //запись вопроса
            mass_trees[i]->print_tree(&tasks_file);          //запись дерева
            tasks_file << '\n';
        }
        tasks_file.close();
        cout << "Введите имя файла для сохранения ответов\n";
        cin >> file_name;
        getchar(); //вытаскиваем символ переноса строки из потока
        fstream answer_file;
        answer_file.open(file_name, fstream::out);          //открытие или создание файла
на запись заданий
        memset(file_name, '\0', 256);
        answer_file << "                                     Текущий контроль.
Ответы.\n\n";
        cout << "\nПромежуточные операции при выполнении задания:\n";
        for (int i = 0; i < count_of_insert; i++) {
            answer_file << "Задание " << i + 1 << ".\nПромежуточные операции при
выполнении задания:\n";
            mass_trees[i]->insert(mass_action[i], &answer_file); //вставка
сгенерированного элемента
            answer_file << "\nОтвет:\n";
            mass_trees[i]->print_tree(&answer_file);          //запись дерева
            answer_file << '\n';
        }
        for (int i = count_of_insert; i < count_of_exceptions + count_of_insert;
i++) {
            answer_file << "Задание " << i + 1 << ".\nПромежуточные операции при
выполнении задания:\n";
            mass_trees[i] = mass_trees[i]->remove(mass_action[i], &answer_file);
//исключение сгенерированного элемента
            answer_file << "\nОтвет:\n";
            mass_trees[i]->print_tree(&answer_file);          //запись дерева
            answer_file << '\n';
        }
        cout << '\n';
        delete file_name;
        answer_file.close();
    }

    AVL_Tree* tree_generator() {
        int new_key;

```

```

        int count_of_elem = rand() % 8 + 3;    //генерируем количество элементов
дерева
        AVL_Tree* rand_tree = nullptr;
        for (int i = 0; i < count_of_elem; i++) {
            new_key = rand() % 100 + 1;
            while (rand_tree->find_key(new_key)) {
                new_key = rand() % 100 + 1;    //генерируем пока элемент уже
есть в дереве
            }
            rand_tree = rand_tree->insert(new_key);    //вставляем новый элемент
        }
        return rand_tree;
    }

AVL_Tree* manual_generator() {
    string inp_num_dialog = "Введите ключ";
    string question = "\nВведите команду:\n1 - Вставка элемента\n2 - Удаление
элемента\n3 - Закончить создание дерева\n ";
    AVL_Tree* my_tree = nullptr;
    while (1) { //меню создания дерева
        switch (input_num(question))
        {
            case 1: my_tree = my_tree->insert(input_num(inp_num_dialog)); cout <<
"Текущее состояние дерева:\n"; my_tree->print_tree(&cout); break; //вставка
элемента

            case 2: my_tree = my_tree->remove(input_num(inp_num_dialog)); cout <<
"Текущее состояние дерева:\n"; my_tree->print_tree(&cout); break; //удаление
элемента

            case 3: cout << "Конец создания дерева\n"; return my_tree; break;
//завершение создания дерева
            default:
                cout << "Команда не распознана!\n"; break;
        }
    }
}

int user_interface(int mode) {
    string insert_mes = "Введите ключ элемента вставки, для генерации задания.";
    string except_mes = "Введите ключ элемента исключения, для генерации
задания.";
    string message = "Введите количество заданий на вставку элемента";
    string dialog_text = "\nВыберите дальнейшее действие:\n1 - сохранить задания
с ответами в файл и продолжить\n2 - сохранить задания с ответами в файл и выйти\n3
- продолжить без сохранения\n4 - выйти без сохранения";
    int count_of_insert = input_num(message);
    message = "Введите количество заданий на исключение элемента";
    int count_of_exceptions = input_num(message);
    if ((count_of_insert < 1) && (count_of_exceptions < 1)) { //проверка на
некорректное количество заданий
        return 1;
    }
}

```

```

    AVL_Tree** mass_trees = new AVL_Tree*[count_of_insert +
count_of_exceptions];
    int * mass_action = new int[count_of_insert + count_of_exceptions];
    int count_of_trees = count_of_insert + count_of_exceptions;
    int action;
    cout <<
"-----\
n";
    for (int i = 0; i < count_of_insert; i++) {
        if (mode == 1) {
            cout << "Промежуточные данные:\n";
            mass_trees[i] = tree_generator(); //генерация случайного дерева
            action = rand() % 100 + 1;
            while (mass_trees[i]->find_key(action)) {
                action = rand() % 100 + 1;    //генерация случайного ключа
для вставки
            }
        }
        if (mode == 2) {
            cout << "Создание дерева для " << i + 1 << " задачи по
вставке\n";
            mass_trees[i] = manual_generator();
            action = input_num(insert_mes);
        }
        mass_action[i] = action;    //заполняем массив преобразований
        cout << "\nИтоговое задание " << i+1 << ": как будет выглядеть
следующее дерево после вставки элемента "<< mass_action[i]<<" ?\n\n";
        mass_trees[i]->print_tree(&cout);
        cout << '\n';
    }
    for (int i = count_of_insert; i < count_of_trees; i++) {
        if (mode == 1) {
            cout << "Промежуточные данные:\n";
            mass_trees[i] = tree_generator();    //генерация случайного
дерева
            action = rand() % 100 + 1;
            while (!mass_trees[i]->find_key(action)) {
                action = rand() % 100 + 1;    //генерация случайного ключа
для исключения
            }
        }
        if (mode == 2) {
            cout << "Создание дерева для " << i + 1 << " задачи по
исключению\n";
            mass_trees[i] = manual_generator();
            action = input_num(except_mes);
        }
        mass_action[i] = action;    //заполняем массив преобразований
        cout << "\nИтоговое задание " << i + 1 << ": как будет выглядеть
следующее дерево после исключения элемента " << mass_action[i] << " ?\n\n";
    }
}

```

```

        mass_trees[i]->print_tree(&cout);
        cout << '\n';
    }
    cout << "Нажмите ENTER, чтобы продолжить";
    getchar();
    while (1) {
        switch (input_num(dialog_text)) { //выбор дальнейших действий
пользователем
            case 1: save_tasks(mass_trees, mass_action, count_of_insert,
count_of_exceptions); clear_memory(mass_trees, mass_action, count_of_trees);
return 1; break;//сохранение и очистка данных
            case 2: save_tasks(mass_trees, mass_action, count_of_insert,
count_of_exceptions); clear_memory(mass_trees, mass_action, count_of_trees);
return 0; break;
            case 3: clear_memory(mass_trees, mass_action, count_of_trees); return
1; break;
            case 4: clear_memory(mass_trees, mass_action, count_of_trees); return
0; break;
            default: cout << "Команда не распознана!\n"; break;
        }
    }
}

int main()
{
    srand(time(0)); //инициализация генератора случайных чисел
    setlocale(LC_ALL, "rus");
    string start_dialog = "\nВыберите действие:\n1 - Генерация заданий\n2 -
Ручная генерация\n3 - Выйти из программы";
    while (1) {
        switch (input_num(start_dialog)) {
            case 1:
                cout << "Выбрана randomная генерация заданий\n\n";
                if (!user_interface(1)) {
                    system("pause");
                    return 0;
                }
                break;
            case 2:
                cout << "Выбрана ручная генерация заданий\n\n";
                if (!user_interface(2)) {
                    system("pause");
                    return 0;
                }
                break;
            case 3:
                cout << "Выход из программы\n";
                return 0;
                break;
            default:

```

```

        cout << "Ответ некорректный!\n\n";
    }
}

```

Файл AVL_Tree.h:

```

#include "stdafx.h"
#include <iostream>
#include <fstream>
using namespace std;

class AVL_Tree {
private:
    int height;
    int key;
    AVL_Tree* left;
    AVL_Tree* right;
public:
    void destroy();
    AVL_Tree(int key);
    bool find_key(int k);
    void print_tree(ostream* stream, int depth = 0);
    int get_height();
    int bfactor();
    void fixheight();
    AVL_Tree* rotateright(ostream* stream = nullptr);
    AVL_Tree* rotateleft(ostream* stream = nullptr);
    AVL_Tree* balance(ostream* stream = nullptr);
    AVL_Tree* insert(int k, ostream* stream = nullptr);
    AVL_Tree* findmin();
    AVL_Tree* removemin();
    AVL_Tree* remove(int k, ostream* stream = nullptr);
};

```

Файл AVL_Tree.cpp:

```

#include "stdafx.h"
#include "AVL_Tree.h"

AVL_Tree::AVL_Tree(int key) { //конструктор
    this->height = 1;
    this->key = key;
    left = nullptr;
    right = nullptr;
}

bool AVL_Tree::find_key(int k) { //метод нахождения элемента с ключём k
    if (this == nullptr) {
        return 0;
    }
}

```

```

    }
    if (key == k) {
        return true;
    }
    if ((left->find_key(k)) || (right->find_key(k))) { //рекурсивный вызов для
поддеревьев
        return true;
    }
    return false;
}

int AVL_Tree::get_height() { //возвращает значение высоты дерева
    if (this != nullptr) {
        return height;
    }
    return 0;
}

int AVL_Tree::bfactor() { //разница между высотами поддеревьев
    return right->get_height() - left->get_height();
}

void AVL_Tree::fixheight() { //обновляет высоту после вставки/удаления элемента
    int hl = left->get_height();
    int hr = right->get_height();
    if (hl > hr) {
        height = hl + 1;
    }
    else {
        height = hr + 1;
    }
}

AVL_Tree* AVL_Tree::rotateright(ostream* stream) { //поворот вправо
    if (stream != nullptr)
        *stream << "Производится поворот вправо\n";
    cout << "Производится поворот вправо\n";
    AVL_Tree* q = left;
    left = q->right;
    q->right = this;
    fixheight();
    q->fixheight();
    return q;
}

AVL_Tree* AVL_Tree::rotateleft(ostream* stream) { //поворот влево
    if (stream != nullptr)
        *stream << "Производится поворот влево\n";

```

```

        cout << "Производится поворот влево\n";
        AVL_Tree* q = right;
        right = q->left;
        q->left = this;
        fixheight();
        q->fixheight();
        return q;
    }

    AVL_Tree* AVL_Tree::balance(ostream* stream) { //ребалансировка дерева
        fixheight();
        if (bfactor() == 2)
        {
            if (right->bfactor() < 0)
                right = right->rotateright(stream);
            return rotateleft(stream);
        }
        if (bfactor() == -2)
        {
            if (left->bfactor() > 0)
                left = left->rotateleft(stream);
            return rotateright(stream);
        }
        return this;
    }

    AVL_Tree* AVL_Tree::insert(int k, ostream* stream) { //вставка элемента с ключём k
        if (!this) {
            cout << "Вставка элемента "<<k<<'\n';
            if (stream != nullptr)
                *stream << "Вставка элемента " << k << '\n';
            return new AVL_Tree(k);
        }
        if (k < key) {
            left = left->insert(k, stream); //рекурсивный вызов для поддеревьев
        }
        if (k > key) {
            right = right->insert(k, stream); //рекурсивный вызов для поддеревьев
        }
        return balance(stream);
    }

    AVL_Tree* AVL_Tree::findmin() { //нахождение элемента с минимальным ключём
        if (left != nullptr) {
            return left->findmin();
        }
        return this;
    }
}

```

```

AVL_Tree* AVL_Tree::removemin() { //удаление элемента с минимальным ключём
    if (left == nullptr) {
        return right;
    }
    left = left->removemin();
    return balance();
}

AVL_Tree* AVL_Tree::remove(int k, ostream* stream){ //исключение элемента с ключём
k
    if (!this) return nullptr;
    if (k < key)
        left = left->remove(k, stream); //рекурсивный вызов для поддеревьев
    else if (k > key)
        right = right->remove(k, stream); //рекурсивный вызов для поддеревьев
    else
    {
        AVL_Tree* q = left;
        AVL_Tree* r = right;
        cout << "Производится исключение элемента "<<k<<'\n';
        if(stream!=nullptr)
            *stream << "Производится исключение элемента " << k << '\n';
        delete this;
        if (!r) return q;
        AVL_Tree* min = r->findmin();
        min->right = r->removemin();
        min->left = q;
        return min->balance(stream);
    }
    return balance(stream);
}

void AVL_Tree::destroy() { //рекурсивная очистка дерева
    if (this != nullptr) {
        left->destroy();
        right->destroy();
        delete this;
    }
}

void AVL_Tree::print_tree(ostream* stream, int depth) { //печать дерева в поток
    if (this!=nullptr) {
        if (right != nullptr) {
            right->print_tree(stream, depth + 1); //рекурсивный вызов для
            поддеревьев
        }
        for (int i = 0; i < depth; i++) *stream << "    "; //число пробелов
        //зависит от высоты
    }
}

```



```

        *stream << key << endl;
        if (left != nullptr) {
            left->print_tree(stream, depth + 1); //рекурсивный вызов для
поддеревьев
        }
    }
    else {
        *stream << "Пустое дерево\n";
    }
}

```