Авл дерево   
двоичное дерево поиска, отбалансированное, т.е. разница в высоте левого и правого поддерева любого узла не превышает 1.  
**Двоичное дерево поиска** — 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*, нежели значение ключа данных самого узла X.

**Как работает код:**

**1. Класс**Student

* Хранит данные о студенте: имя, номер группы, курс, возраст и средний балл.
* Используется как объект, который будет помещаться в узлы дерева.

**2. Класс**AVLNode

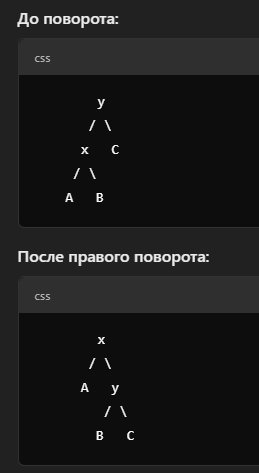
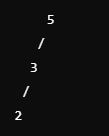
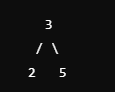
* Узел AVL-дерева:
  + Содержит объект Student.
  + Хранит ссылки на левое (left) и правое (right) поддеревья.
  + Поддерживает свойство height (высота узла), которое нужно для балансировки.

**3. Класс**AVLTree

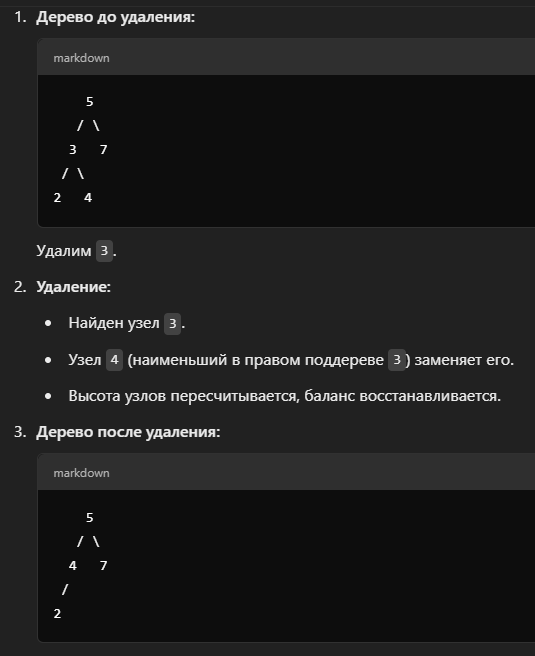
* Реализует AVL-дерево, где каждая операция (вставка, удаление) сохраняет балансировку:
  + **Балансировка:** Разница в высоте левого и правого поддерева любого узла не превышает 1.

**Основные функции:**

**Вспомогательные методы:**

* \_height(node) — Возвращает высоту узла (или 0, если узел пустой).
* \_balance\_factor(node) — Вычисляет баланс-фактор (разницу высот поддеревьев).
* \_rotate\_left(node)**/**\_rotate\_right(node) — Выполняют повороты для восстановления баланса.
* До:  после поворота: 

**Операции:**

1. insert(student):
   * Добавляет новый объект Student в дерево, соблюдая порядок.
   * Балансирует дерево с помощью поворотов (если необходимо).
2. delete(average\_grade):
   * Удаляет узел по значению среднего балла студента.
   * Находит минимальный элемент в правом поддереве для замены (если у узла есть оба потомка).
   * Балансирует дерево после удаления.  
     
3. search(average\_grade):
   * Рекурсивно ищет узел по среднему баллу.
   * Возвращает True, если найден, иначе False.
4. search\_via\_inorder(average\_grade):
   * Выполняет поиск, обходя дерево в порядке "лево-корень-право" (инфиксный обход).
5. save\_to\_file(filename)**/**load\_from\_file(filename):
   * Сохраняет дерево в файл построчно.
   * Загружает дерево из файла, восстанавливая структуру.

**Для чего это нужно:**

1. **Эффективное хранение данных:** AVL-дерево поддерживает упорядоченность элементов, что позволяет быстро выполнять операции поиска, добавления и удаления.
2. **Балансировка:** Обеспечивает время выполнения операций O(log n), даже при большом количестве данных.
3. **Работа с реальными объектами:** Узлы дерева содержат данные о студентах, что может использоваться, например, для работы с базами данных студентов в образовательных учреждениях.

**Максимальная куча** — это структура данных, в которой любой узел всегда больше дочернего узла, а ключ корневого узла является наибольшим среди всех остальных узлов.

**Класс**Student**:**

* Описывает данные студента: ФИО, группа, курс, возраст, средний балл.
* Метод \_\_repr\_\_ возвращает строковое представление объекта для удобного отображения.

**Класс**Node**:**

* Узел двусвязного списка, содержащий объект студента.
* Имеет ссылки на предыдущий (prev) и следующий (next) узлы.

**Класс**MaxHeap**:**

* Управляет узлами, организуя их в структуру максимальной кучи.
* Использует двусвязный список для хранения элементов.

**Ключевые методы:**

**1.**insert**:**

Добавляет нового студента в кучу.

* Создаёт новый узел (Node) с переданным студентом.
* Если куча пуста, устанавливает этот узел как голову (head) и хвост (tail).
* Иначе добавляет узел в конец списка и "поднимает" его вверх по правилу кучи (\_heapify\_up), чтобы структура оставалась корректной.

**2.**\_heapify\_up**:**

Поднимает добавленный элемент вверх для восстановления свойств кучи.

* Сравнивает текущий элемент с его родительским узлом:
  + Если элемент больше родителя, они меняются местами.
  + Повторяет процесс, пока элемент не окажется в правильном месте или не достигнет вершины.

**3.**extract\_max**:**

Извлекает студента с максимальным средним баллом (голову кучи).

* Если куча пуста, возвращает None.
* Удаляет голову (head) и уменьшает размер кучи.
* Если есть элементы, перемещает следующий узел на место головы и восстанавливает структуру через \_heapify\_down.

**4.**\_heapify\_down**:**

Восстанавливает свойства кучи после извлечения максимального элемента.

* Опускает текущий узел вниз, сравнивая с его потомками:
  + Если один из потомков больше текущего узла, меняет их местами.
  + Повторяет процесс, пока элемент не окажется на правильной позиции.

**5.**search**:**

Ищет студента по среднему баллу.

* Последовательно перебирает все узлы списка, сравнивая их средний балл с заданным.

**6.**save\_to\_file**и**load\_from\_file**:**

* Сохраняет студентов в файл в виде строк, разделённых запятыми.
* Загружает студентов из файла, добавляя их в кучу.