16. Tree rotations, AVL trees, inserting/finding/removing an element from an AVL tree.

Tree rotations, AVL trees.

Определения:
1. Уровнем дерева называется число узлов в самом длинном пути от корня до листа

2. Правой высотой узла называется уровень его правого поддерева , аналогично для левого.

3. Дерево называется сбалансированным, если для каждого узла разность правой и левой высоты <=1. Эта разность называется фактором балансировки.

AVL-дерево (дерево Адельсона-Вельского и Лэндиса) — это самобалансирующееся бинарное дерево поиска. Оно гарантирует, что разница высот между левым и правым поддеревом любого узла не превышает 1. Это обеспечивает O(logn) сложность для операций поиска, вставки и удаления. При вставке или удалении узлов выполняются повороты для восстановления баланса.

Добавление элемента в AVL-дерево выполняется следующим образом:

1. Элемент добавляется так же, как в бинарное дерево поиска (BST).
2. При возврате вверх по дереву проверяется фактор балансировки каждого узла. Если фактор балансировки для какого-либо узла x становится ≥2 или ≤−2, то балансировка нарушена, и выполняется одна из следующих операций:
   - **Случай 1:** Если добавленный элемент находится в левом поддереве левого узла x , выполняется **правый поворот (right rotation)** для узла x.
   - **Случай 2:** Если добавленный элемент находится в правом поддереве правого узла x, выполняется **левый поворот (left rotation)** для узла x
   - **Случай 3:** Если добавленный элемент находится в правом поддереве левого узла x , выполняется **лево-правый поворот (left-right rotation)**.
   - **Случай 4:** Если добавленный элемент находится в левом поддереве правого узла x, выполняется **право-левый поворот (right-left rotation)**.

Чтобы эффективно вычислять фактор балансировки каждого узла, в структуре узла, помимо значений value, left, right, также хранится высота дерева, исходящего из данного узла. При возврате вверх по дереву значение высоты обновляется.

1. Пример для правого поворота

```
        30                     20
       /                      /  \
     20         →           10    30
    /
  10
```

- Добавляем узел 10, из-за чего узел 30 становится несбалансированным (фактор баланса узла 30 = 2).
- Выполняем **правый поворот** на узле 30.

Шаги:

1. Узел 20 становится новым корнем.
2. Узел 30 становится правым потомком узла 20.
3. Узел 10 остается левым потомком узла 20.

2. Пример для левого поворота

Изначальное дерево:

```
        10                     20
          \                   /  \
           20      →        10    30
             \
              30
```

- Добавляем узел 30, из-за чего узел 10 становится несбалансированным (фактор баланса узла 10 = -2).
- Выполняем **левый поворот** на узле 10.

Шаги:

1. Узел 20 становится новым корнем.
2. Узел 10 становится левым потомком узла 20.
3. Узел 30 остается правым потомком узла 20.

3. Пример для лево-правого поворота:

```
        30                     30                     20
       /                      /                      /  \
     10         →           20          →          10    30
       \                   /
        20               10
```

- Добавляем узел 20, из-за чего узел 30 становится несбалансированным (фактор баланса узла 30 = 2), а узел 10 имеет фактор баланса = -1.
- Выполняем **лево-правый поворот**.

1. Выполняем **левый поворот** на узле 10:
2. Выполняем **правый поворот** на узле 30:

4. Пример для право-левого поворота

```
 10                       10                       20
   \                        \                     /  \
    30          →            20         →       10    30
   /                           \
 20                             30
```

- Добавляем узел 20, из-за чего узел 10 становится несбалансированным (фактор баланса узла 10 = -2), а узел 30 имеет фактор баланса = 1.
- Выполняем **право-левый поворот**.

1. Выполняется **правый поворот** на узле 30.
2. Выполняется **левый поворот** на узле 10.

УВЕРЖДЕНИЕ: Пусть n_L - мин. Число узлов в AVL дереве с уровнем L, тогда

$n_L = F_{L+2} - 1$, где F_k — k-ое число Фибоначчи с начальным значением $F_1 = F_2 = 1$

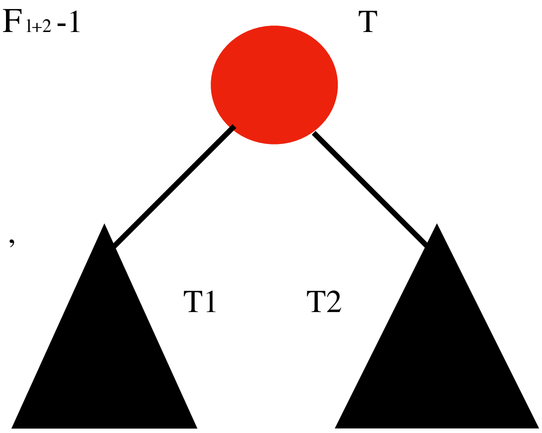
ДОК: при L=0 : $n_0 = 0,\ F_{0+2} = 1 \quad \Rightarrow \quad n_0 = F_{0+2} - 1$

при L=1 : $n_1 = 1,\ F_{1+2} = 2 \quad \Rightarrow \quad n_1 = F_{1+2} - 1$

Пред. Что при $1 < L$ утверждение верно, те $n_l = F_{l+2} - 1$

Док. для L, те $n_L = F_{L+2} - 1$.

(*)$n_L = n_{L-1} + n_{L-2} + 1 = F_{L+1} - 1 + F_L - 1 + 1 = F_{L+2} - 1$.



Если у одного поддерева высота L - 1 , то у другого поддерева максимальная допустимая высота — L - 2 ,

чтобы разница в высотах (L - 1 - (L - 2) = 1) оставалась в допустимых пределах и дерево оставалось сбалансированным, те если L(T1)= L-1 то L(T2) = L-2 => (*) верна.

СЛЕДСТВИЕ: Число узлов в AVL дереве с уровнем $L \geq F_{L+2} - 1 = \theta(\varphi^L)$.

ДОК: $F_{n+2} = F_{n+1} + F_n \Rightarrow \lambda^{n+2} = \lambda^{n+1} + \lambda^n \Rightarrow \lambda^2 - \lambda - 1 = 0 \Rightarrow \lambda = (1 \pm \sqrt{5}) / 2$

$F_n = c_1 \cdot ((1 + \sqrt{5}) / 2)^n + c_2 \cdot ((1 - \sqrt{5}) / 2)^n$

Для больших n, второе слагаемое стремится к нулю, так как $c_2 \cdot ((1 - \sqrt{5}) / 2)^n \approx -0.618$, и его степень будет быстро убывать. Поэтому для больших значений n основным вкладом в выражение для F_n является первое слагаемое: $F_n \approx c_1 \cdot ((1 + \sqrt{5}) / 2)^n \approx 1{,}618$

Таким образом, для больших n числа Фибоначчи растут экспоненциально с основанием

$\varphi = (1 + \sqrt{5}) / 2$

Поскольку минимальное количество узлов в AVL-дереве с высотой L равно $n_L = F_{L+2} - 1$, мы можем оценить количество узлов как: $n_L \approx F_{L+2} \approx c_1 \cdot \varphi^{L+2}$.Таким образом, количество узлов в AVL-дереве растет экспоненциально с высотой дерева с основанием φ, что даёт асимптотику: $n_L = \theta(\varphi^L)$

Здесь:

- θ — это нотация для описания асимптотического поведения функции. Это означает, что функция растет с той же скоростью, что и φ^L, когда L становится очень большим.
- φ — это число, равное $(1 + \sqrt{5}) / 2$, которое является золотым сечением.
- L — это уровень в дереве, и выражение φ^L показывает, как быстро растет число узлов в AVL-дереве в зависимости от его высоты.

Заключение:

Следовательно, мы доказали, что число узлов в AVL-дереве с высотой L растет как $\theta(\varphi^L)$, где $\varphi = (1 + \sqrt{5}) / 2$ — золотое сечение.

Removing and Finding

Removing

Удаление элемента из AVL-дерева происходит по тому же принципу, что и в обычном бинарном дереве поиска:

1. Находим элемент для удаления.
2. Если у удаляемого узла есть два потомка, заменяем его на наименьший элемент из правого поддерева или на наибольший элемент из левого поддерева.
3. После удаления элемента обновляем высоты всех предков удалённого узла.
4. Проверяем балансировку каждого узла, начиная от удалённого узла до корня дерева.
5. Если балансировка нарушена, выполняем соответствующие **повороты**.

Повороты при удалении:

- **Левый-левый случай (Left-Left, LL)**:
  - Дерево стало слишком левым.
  - Выполняем **правый поворот**.
- **Правый-правый случай (Right-Right, RR)**:
  - Дерево стало слишком правым.
  - Выполняем **левый поворот**.
- **Левый-правый случай (Left-Right, LR)**:
  - Дерево стало сбалансированным, но с перегрузкой на правой ветви левого поддерева.
  - Выполняем **левый поворот**, затем **правый поворот**.
- **Правый-левый случай (Right-Left, RL)**:

- ◦ Дерево стало сбалансированным, но с перегрузкой на левой ветви правого поддерева.
- ◦ Выполняем **правый поворот**, затем **левый поворот**.

Алгоритм балансировки после удаления элемента из AVL-дерева описывается следующими шагами:
Если баланс узла нарушен, выполняется один из четырех типов поворотов:

- **Левый-левый случай (Left-Left Case)**: Если баланс текущего узла больше 1, и баланс его левого поддерева также больше или равен 0, выполняем **правый поворот** для текущего узла.

- **Правый-правый случай (Right-Right Case)**: Если баланс текущего узла меньше -1, и баланс его правого поддерева меньше или равен 0, выполняем **левый поворот** для текущего узла.

- **Левый-правый случай (Left-Right Case)**: Если баланс текущего узла больше 1, и баланс его левого поддерева меньше 0, то сначала выполняем **левый поворот** для левого поддерева, а затем **правый поворот** для текущего узла.

- **Правый-левый случай (Right-Left Case)**: Если баланс текущего узла меньше -1, и баланс его правого поддерева больше 0, то сначала выполняем **правый поворот** для правого поддерева, а затем **левый поворот** для текущего узла.

Пример удаления:

1. Исходное дерево:

```
    10
   /  \
  5    15
 / \
3   7
```

2. Удаление элемента 15

```
    10
   /
  5
 / \
3   7
```

3. После удаления балансировка нарушена на узле 10 фактор балансировки равен 2).
   Выполняем **правый поворот** на узле 10

```
    5
   / \
  3   10
      /
     7
```

Finding

Поиск элемента в AVL-дереве работает так же, как и в обычном бинарном дереве поиска. Мы начинаем с корня дерева и, в зависимости от значения элемента, перемещаемся в левое или правое поддерево, пока не найдем элемент или не дойдем до пустого узла (что означает, что элемент не найден).

Алгоритм поиска:

1. Сравниваем искомый элемент с текущим узлом.
2. Если элемент меньше текущего узла, переходим в левое поддерево.
3. Если элемент больше текущего узла, переходим в правое поддерево.
4. Если нашли элемент, возвращаем его. Если дошли до пустого узла, значит, элемента нет в дереве.

Заключение

- **Вставка** и **удаление** элементов в AVL-дереве требуют проверки и поддержания балансировки дерева. После каждой операции вставки или удаления важно проверять балансировку узлов и при необходимости выполнять повороты.
- **Поиск** элемента в AVL-дереве работает по тому же принципу, что и в обычном бинарном дереве поиска, так как AVL-дерево также сохраняет свойства бинарного дерева поиска.

Время выполнения: Все операции в AVL-дереве (поиск, вставка, удаление) выполняются за время O(log n), где n — количество узлов в дереве, благодаря поддержанию баланса.

(Все операции в AVL-дереве (поиск, вставка, удаление) начинаются с корня и идут вниз по дереву. Так как высота дерева ограничена **log n**(ранее доказали это), количество шагов, которые нужно пройти, чтобы найти или вставить/удалить элемент, также ограничено **O(logn)**.)