Алгоритмы и структуры данных

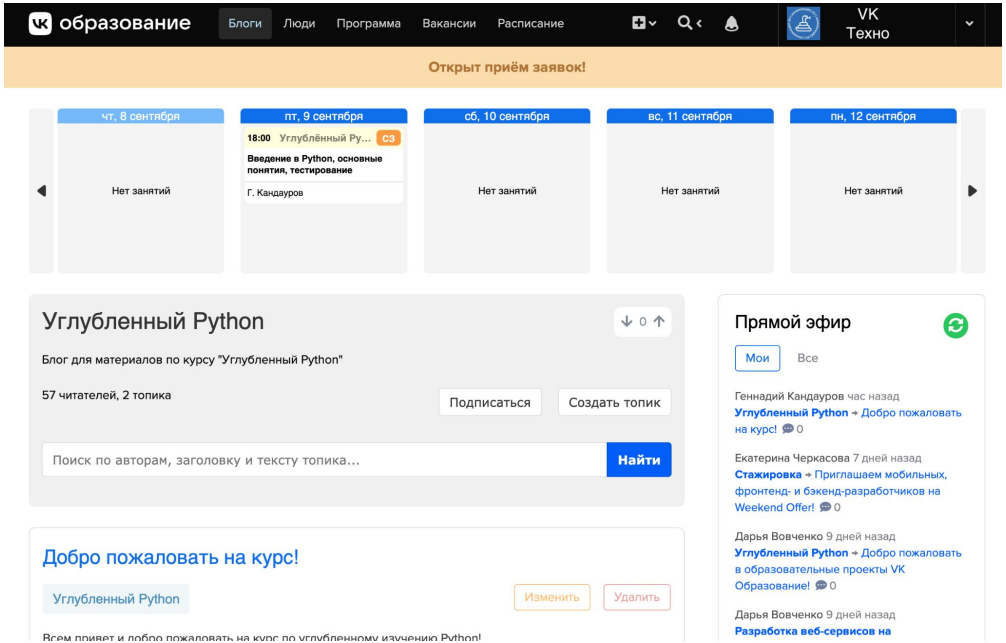
Лекция 5
Деревья, основы

Кандауров Геннадий

education

Напоминание отметиться на портале

+ оставить отзыв



Квиз про прошлой лекции

Содержание занятия

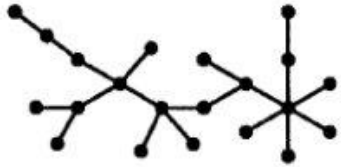
1. Деревья, обход, операции
2. Деревья поиска
3. AVL

Деревья

Дерево

Дерево (свободное) – непустая коллекция вершин и ребер, удовлетворяющих определяющему свойству дерева.

Дерево (свободное) – неориентированный связный граф без циклов.



Вершина (узел) – простой объект, который может содержать некоторую информацию.

Ребро – связь между двумя вершинами.

Путь в дереве – список отдельных вершин, в котором следующие друг за другом вершины соединяются ребрами дерева.

Определяющее свойство дерева – существование только одного пути, соединяющего любые два узла.

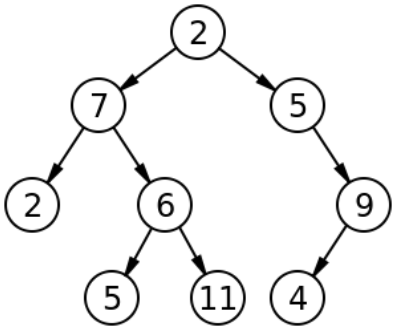
Дерево

Дерево с корнем —дерево, в котором один узел выделен и назначен «корнем» дерева. Существует только один путь между корнем и каждым из других узлов дерева.

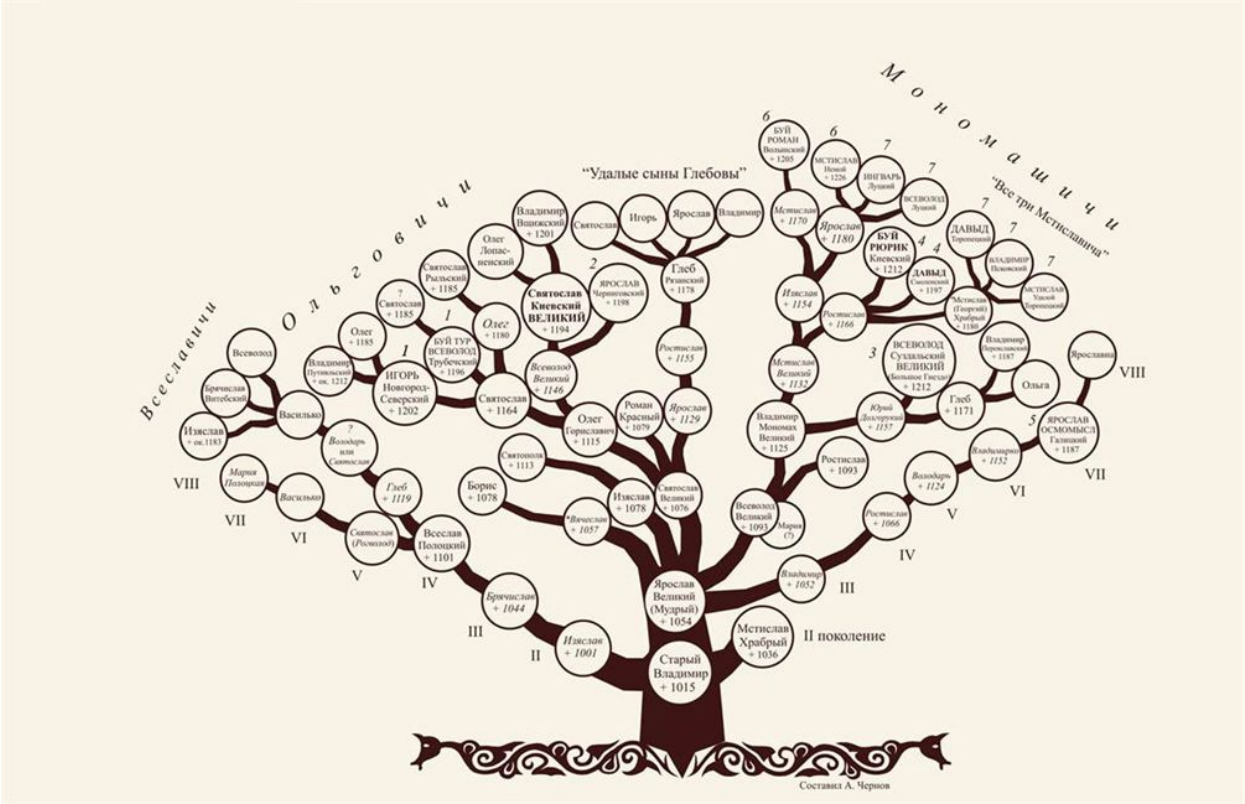
Высота (глубина) дерева с корнем — количество вершин в самом длинном пути от корня.

Обычно дерево с корнем рисуют с корнем, расположенным сверху. Узел y располагается под узлом x (а x располагается над y), если x располагается на пути от y к корню.

Каждый узел (за исключением корня) имеет только один узел, расположенный над ним. Такой узел называется **родительским.**
Узлы, расположенные непосредственно под данным узлом, называются его **дочерними** узлами.
Узлы, не имеющие дочерних узлов называются **листьями**.



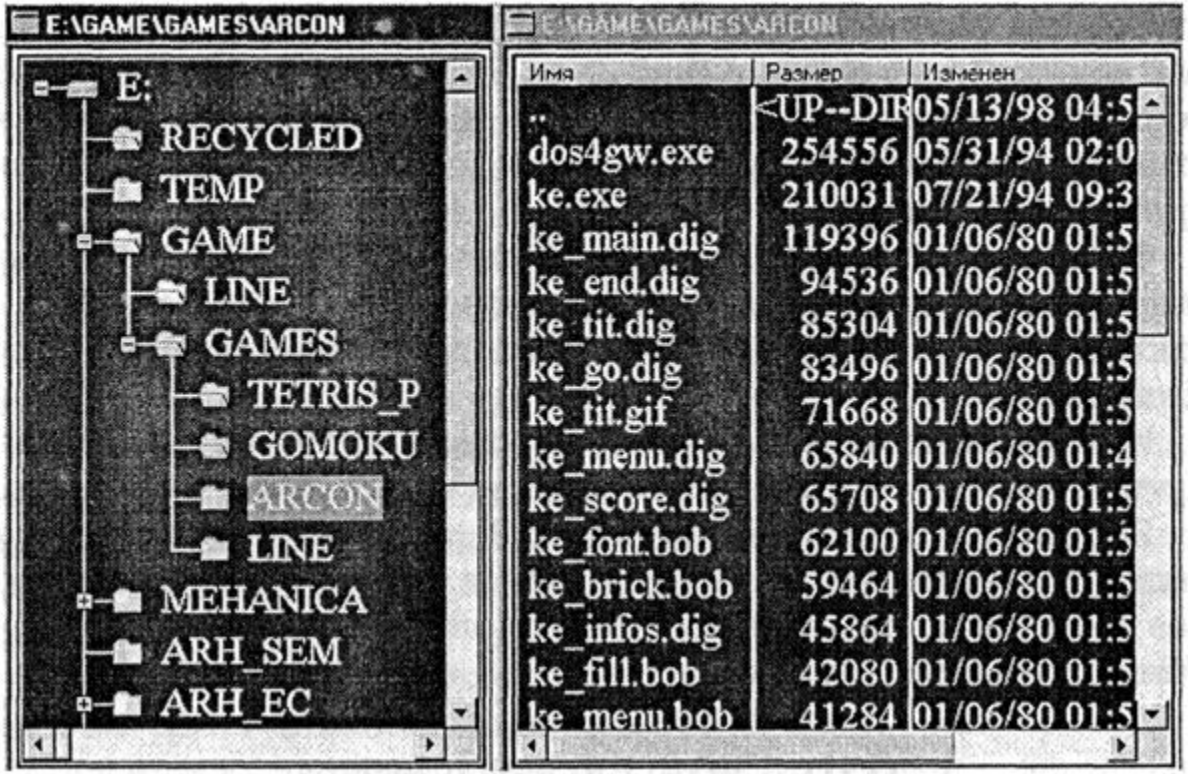
Примеры деревьев: генеалогическое



Примеры деревьев: XML

```
<?xml version="1.0"?>
- <job>
  - <production>
      <ApprovalType>WebCenter</ApprovalType>
      <Substrate>carton 150 gr</Substrate>
      <SheetSize>220-140</SheetSize>
      <press>SuperFlat2</press>
      <finishing>standard</finishing>
      <urgency>normal</urgency>
    </production>
  - <customer>
      <name>FruitCo</name>
      <number>2712</number>
      <currency>USD</currency>
    </customer>
</job>
```

Примеры деревьев: файловая система



Вершины и ребра

- Любое дерево (с корнем) содержит листовую вершину.

  Доказательство. Самая глубокая вершина является листовой.

- Дерево, состоящее из N вершин, содержит N – 1 ребро.

  Доказательство. По индукции.

  База индукции. N = 1. Одна вершина, ноль ребер.

  Шаг индукции. Пусть дерево состоит из N + 1 вершины. Найдем листовую вершину. Эта вершина содержит ровно 1 ребро. Дерево без этой вершины содержит N вершин, а по предположению индукции N – 1 ребро. Следовательно, исходное дерево содержит N ребер, ч.т.д.

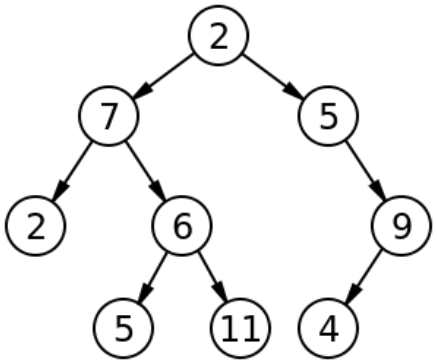
Виды деревьев

- **Двоичное (бинарное) дерево**

  это дерево, в котором степени вершин не превосходят 3.

- **Двоичное (бинарное) дерево с корнем**

  это дерево, в котором каждая вершина имеет не более двух дочерних вершин.



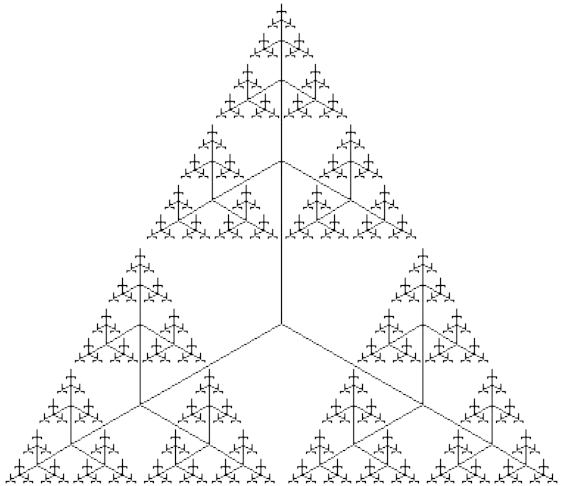
Виды деревьев

- **N-арное дерево**

  это дерево, в котором степени вершин не превосходят N + 1.

- **N-арное дерево с корнем**

  это дерево, в котором каждая вершина имеет не более N дочерних вершин.

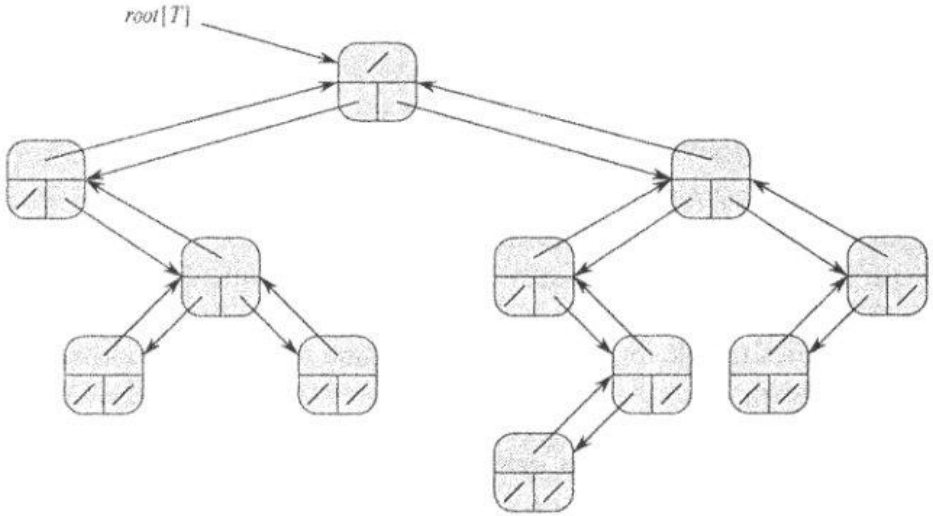


Двоичное дерево

СД Двоичное дерево — представление двоичного дерева с корнем.

Узел – структура, содержащая данные и указатели на левый и правый дочерний узел.

Также может содержать указатель на родительский узел.

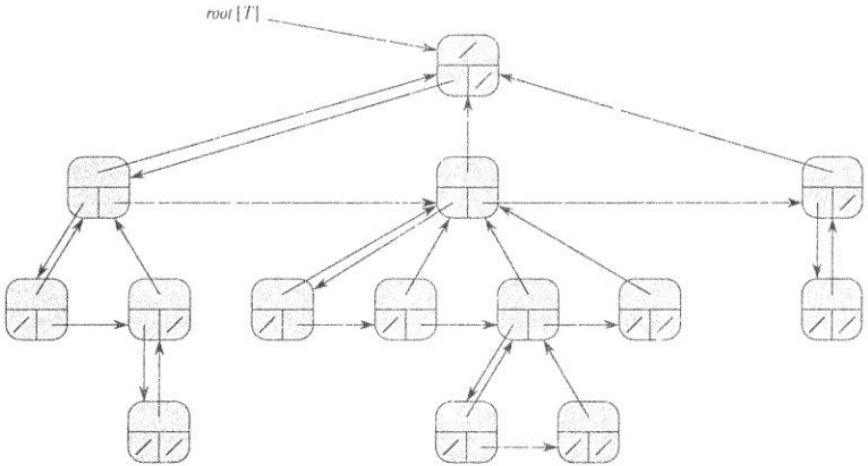


N-арное дерево

СД N-арное дерево — представление N-арного дерева с корнем.

Узел – структура, содержащая данные, указатель на следующий родственный узел и указатель на первый дочерний узел.

Также может содержать указатель на родительский узел.



Обход дерева в глубину

Пошаговый перебор элементов дерева по связям между узлами-предками и узлами-потомками называется **обходом дерева**.

Обходом двоичного дерева в глубину (DFS) называется процедура, выполняющая в некотором заданном порядке следующие действия с поддеревом:

- просмотр (обработка) узла-корня поддерева,
- рекурсивный обход левого поддерева,
- рекурсивный обход правого поддерева.

DFS – Depth First Search

Обход в глубину не начинает обработку других поддеревьев, пока полностью не обработает текущее поддерево

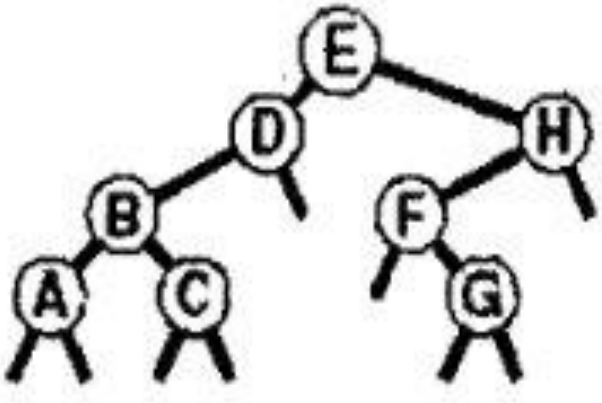
DFS

Прямой обход (сверху вниз, pre-order)

Вначале обрабатывается узел, затем посещается левое и правое поддеревья.

Порядок обработки узлов дерева на рисунке:

E, D, B, A, C, H, F, G



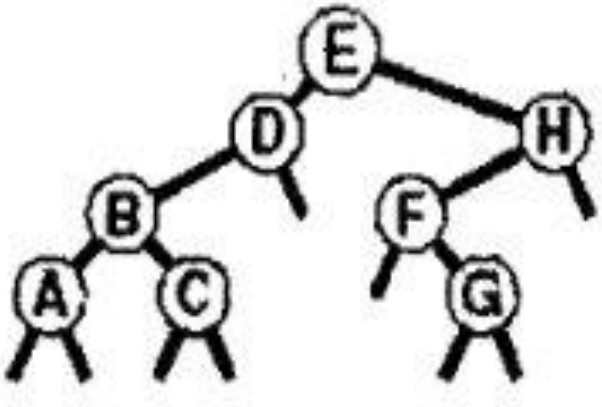
DFS

Обратный обход (снизу вверх, post-order)

Вначале посещаются левое и правое поддеревья, а затем обрабатывается узел.

Порядок обработки узлов дерева на рисунке:

A, C, B, D, G, F, H, E.



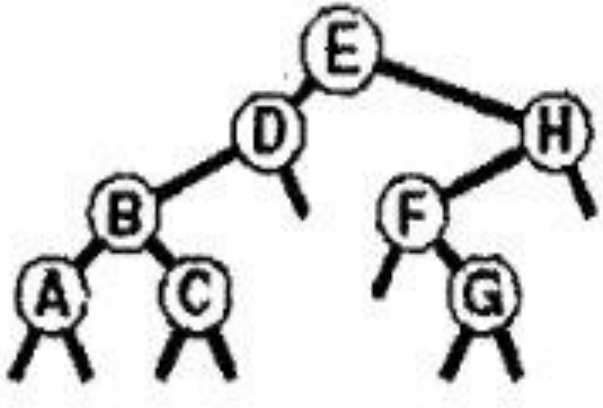
DFS

Поперечный обход (слева направо, in-order)

Вначале посещается левое поддерево, затем узел и правое поддерево.

Порядок обработки узлов дерева на рисунке:

A, B, C, D, E, F, G, H.



Обход дерева в ширину

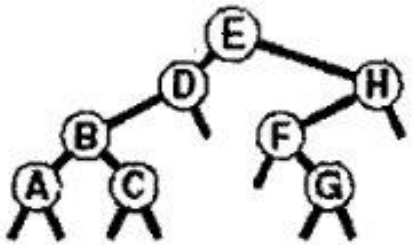
Обход двоичного дерева в ширину (BFS) — обход вершин дерева по уровням (слоям), начиная от корня.
BFS – Breadth First Search.

Используется очередь, в которой хранятся вершины, требующие просмотра.

За одну итерацию алгоритма:

- если очередь не пуста, извлекается вершина из очереди,
- посещается (обрабатывается) извлеченная вершина,
- в очередь помещаются все дочерние.

Порядок обработки узлов дерева:
E, D, H, B, F, A, C, G



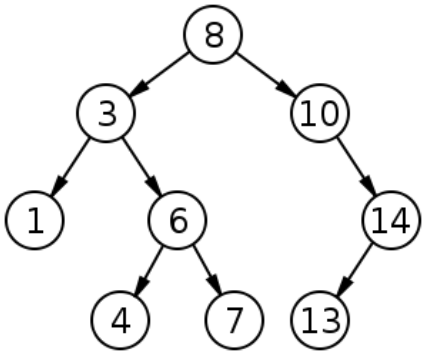
Разница между BFS и DFS

BFS (в ширину)	DFS (в глубину)
▪ Сложность по памяти – O(w), где w – максимальная ширина дерева (максимальная ширина на глубине h = 2^h). Экономичнее в несбалансированных деревьях. ▪ Эффективнее, если при обходе нас интересуют элементы, находящиеся близко к корню.	▪ Сложность по памяти – O(h), где h – максимальная глубина дерева (максимальная глубина в дереве из n элементов = n). Экономичнее в сбалансированных деревьях. ▪ Эффективнее, если при обходе нас интересуют элементы, находящиеся близко к листьям.

Двоичное дерево поиска

Двоичное дерево поиска (binary search tree, BST) – это двоичное дерево, с каждым узлом которого связан ключ, и выполняется следующее дополнительное условие:

- Ключ в любом узле X больше или равен ключам во всех узлах левого поддерева X и меньше или равен ключам во всех узлах правого поддерева X.



Двоичное дерево поиска

Операции с двоичным деревом поиска:

1. Поиск по ключу
2. Поиск минимального, максимального ключей
3. Вставка
4. Удаление
5. Обход дерева в порядке возрастания ключей

BST: поиск по ключу

Дано: корень дерева X и ключ K.

Проверить, есть ли узел с ключом K в дереве, и если да, то вернуть указатель на этот узел.

Алгоритм:

Если дерево пусто, сообщить, что узел не найден, и остановиться.

Иначе сравнить K со значением ключа корневого узла X.

- Если K == X, выдать ссылку на этот узел и остановиться.
- Если K > X, рекурсивно искать ключ K в правом поддереве X.
- Если K < X, рекурсивно искать ключ K в левом поддереве X.

Время работы: O(h), где h – глубина дерева.

BST: поиск минимального ключа

Дано: указатель на корень непустого дерева X.

Найти узел с минимальным значением ключа.

Алгоритм:

Переходить в левый дочерний узел, пока такой существует.

Время работы: O(h), где h – глубина дерева.

BST: добавление узла

Дано: указатель на корень дерева X и ключ K.

Вставить узел с ключом K в дерево (возможно появление дубликатов).

Алгоритм:

Если дерево пусто, заменить его на дерево с одним корневым узлом и остановиться.

Иначе сравнить K с ключом корневого узла X.

- Если K < X, рекурсивно добавить K в левое поддерево X.
- Иначе рекурсивно добавить K в правое поддерево X.

Время работы: O(h), где h – глубина дерева.

BST: удаление узла

Дано: указатель на корень дерева X и ключ K.

Удалить из дерева узел с ключом K (если такой есть).

Алгоритм:

Если дерево пусто, остановиться.

Иначе сравнить K с ключом корневого узла X.

- Если K < X, рекурсивно удалить K из левого поддерева T.
- Если K > X, рекурсивно удалить K из правого поддерева T.
- Если K == X, то необходимо рассмотреть три случая:
    1. Обоих дочерних нет. Удаляем узел X, обнуляем ссылку.
    2. Одного дочернего нет. Переносим дочерний узел в X, удаляем узел.
    3. Оба дочерних узла есть.

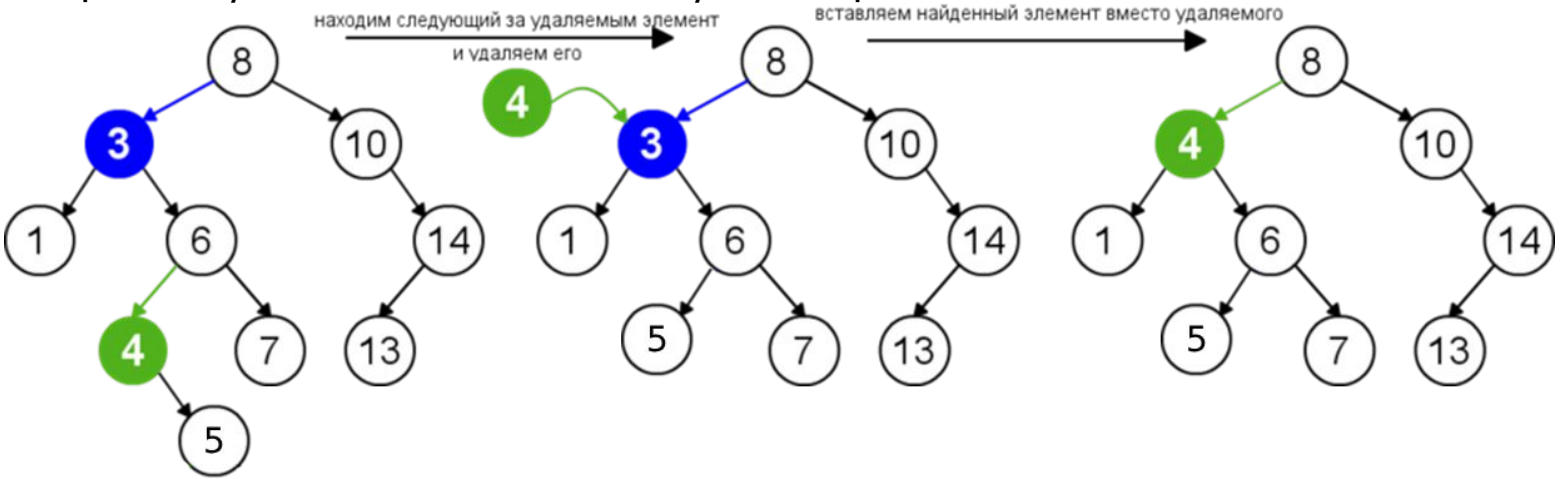
BST: удаление узла, есть оба дочерних узла

Заменяем ключ удаляемого узла на ключ минимального узла из правого поддерева, удаляя последний.

Пусть удаляемый узел – X, а Y – его правый дочерний.

- Если у узла Y отсутствует левое поддерево, то копируем из Y в X ключ и указатель на правый узел. Удаляем Y.
- Иначе найдем минимальный узел Z в поддереве Y. Копируем ключ из Z, удаляем Z. При удалении Z копируем указатель на правый дочерний узел Z в левый дочерний узел родителя Z.

Время работы удаления: O(h), где h – глубина дерева.

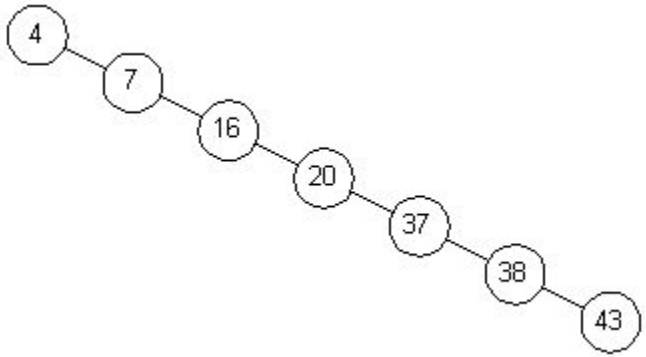


Несбалансированное дерево

Операции поиска в дереве - **O(h)**, **h** - глубина дерева.

Глубина может достигать **n**.

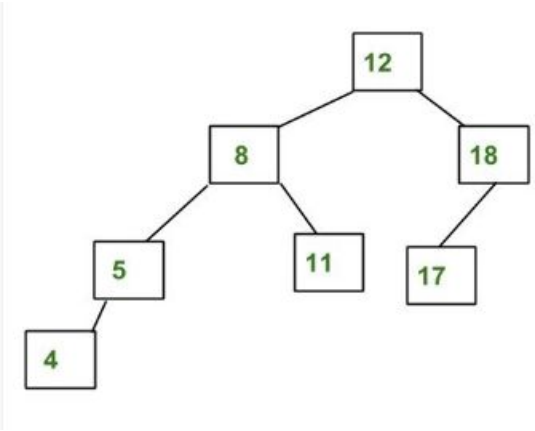
Необходима балансировка

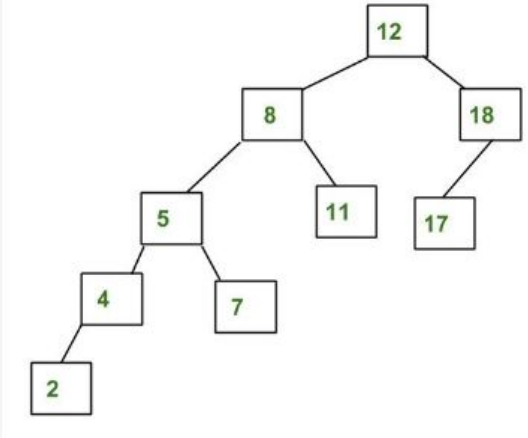


АВЛ-дерево

АВЛ-дерево — сбалансированное двоичное дерево поиска. Для каждой его вершины высоты её двух поддеревьев различаются не более чем на 1.

Изобретено Адельсон-Вельским Г.М. и Ландисом Е.М. в 1962г.



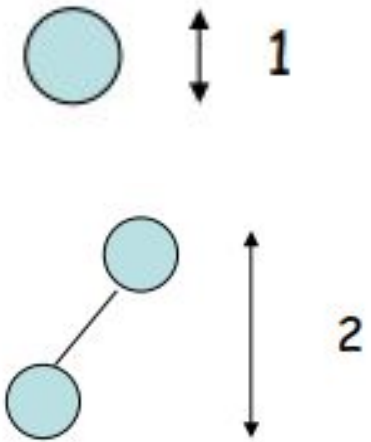


АВЛ-дерево

Теорема. Высота АВЛ-дерева h = O(logN).

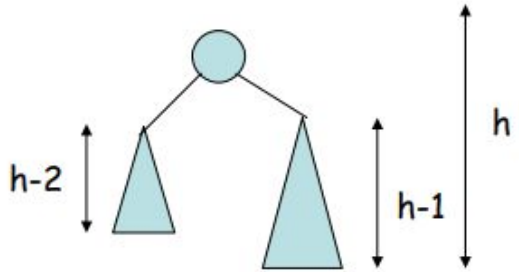
Рассмотрим n(h): минимальное число узлов в АВЛ-дереве высоты h.

n(1) = 1
n(2) = 2



АВЛ-дерево

Для n ≥ 3, АВЛ-дерево с минимальным числом узлов высоты h состоит из корня, АВЛ-поддерева высоты h-1 и АВЛ-поддерева высоты h-2.



Таким образом, n(h) = 1 + n(h-1) + n(h-2)

АВЛ-дерево

$n(h) = 1 + n(h-1) + n(h-2)$

Так как $n(h-1) > n(h-2)$, то

$n(h) > 1 + n(h-2) + n(h-2)$

Таким образом, $n(h) > 2n(h-2)$

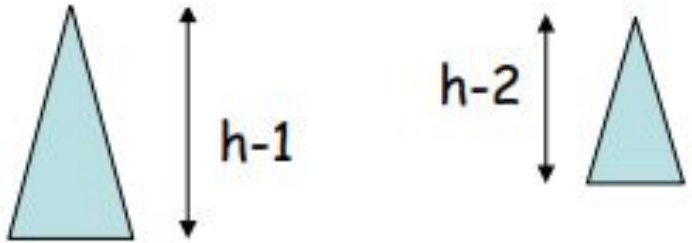
Заметим, что

$n(h-2) > 2n(h-4)$

$n(h-4) > 2n(h-6)$

...

То есть $n(h) > 2^i n(h-2i)$



АВЛ-дерево

$n(h) > 2^i n(h-2i)$

При $i = h/2 - 1$ получается $n(h) > 2^{h/2 - 1} * n(2)$, то есть $n(h) > 2^{h/2}$

Прологарифмируем обе части неравенства:

$$\log(n(h)) > h/2$$

$$h < 2\log(n(h))$$

То есть, высота АВЛ-дерева $h = O(\log N)$

АВЛ-дерево

Специальные балансирующие операции, восстанавливающие основное свойство «высоты двух поддеревьев различаются не более чем на 1» – вращения.

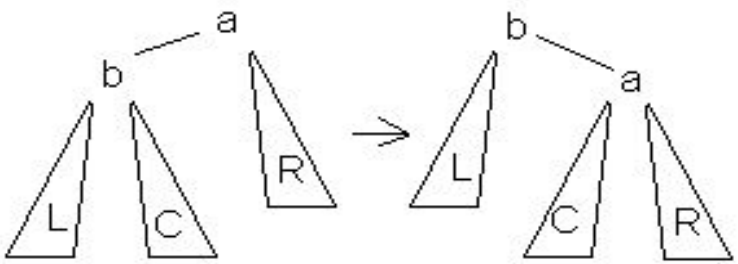
- Малое правое вращение
- Малое левое вращение
- Большое правое вращение
- Большое левое вращение

АВЛ-дерево

- **Малое правое вращение**

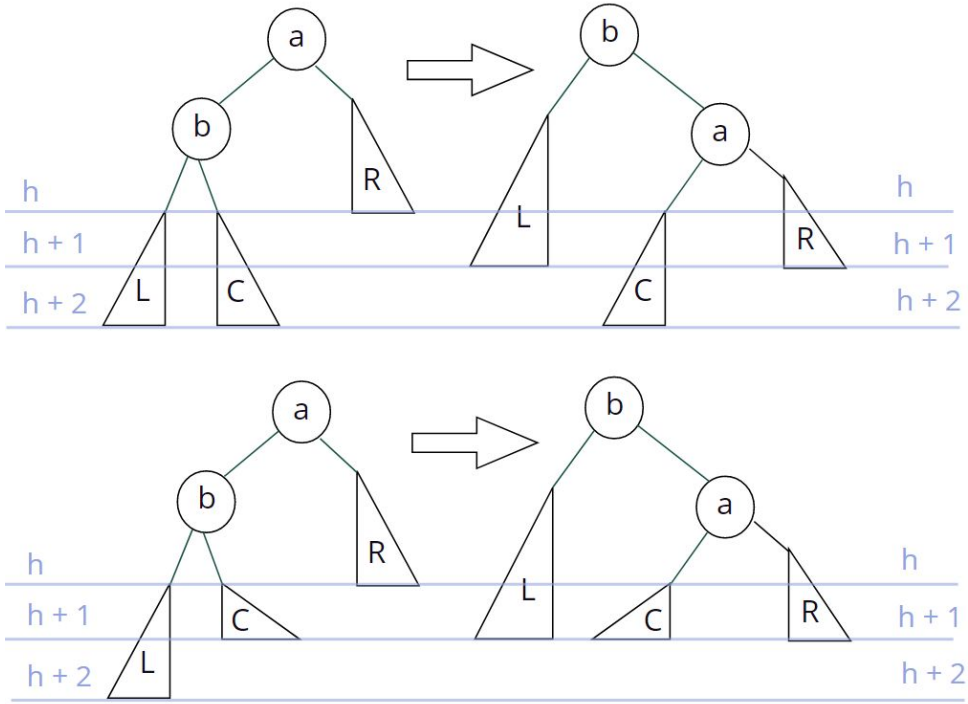
Используется, когда:
высота(L) = высота(R) + 2 и высота(C) ≤ высота(L).

После операции:
высота дерева останется прежней, если высота(C) = высота(L),
высота дерева уменьшится на 1, если высота(C) < высота(L).



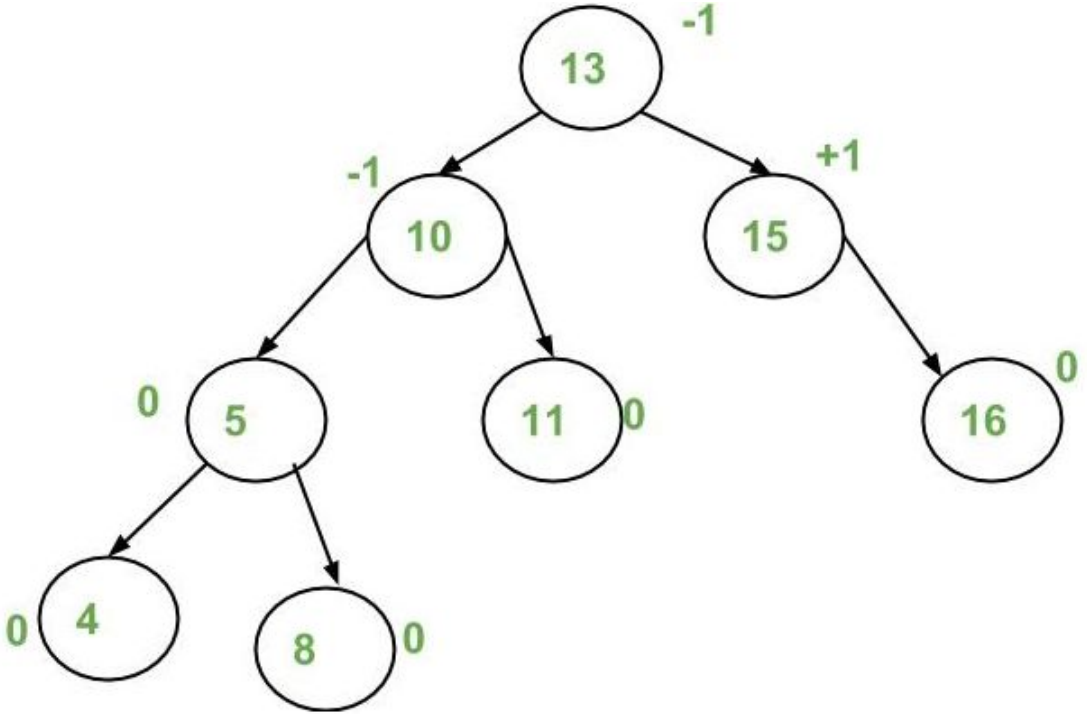
АВЛ-дерево

Как малое правое вращение устраняет дисбаланс высот



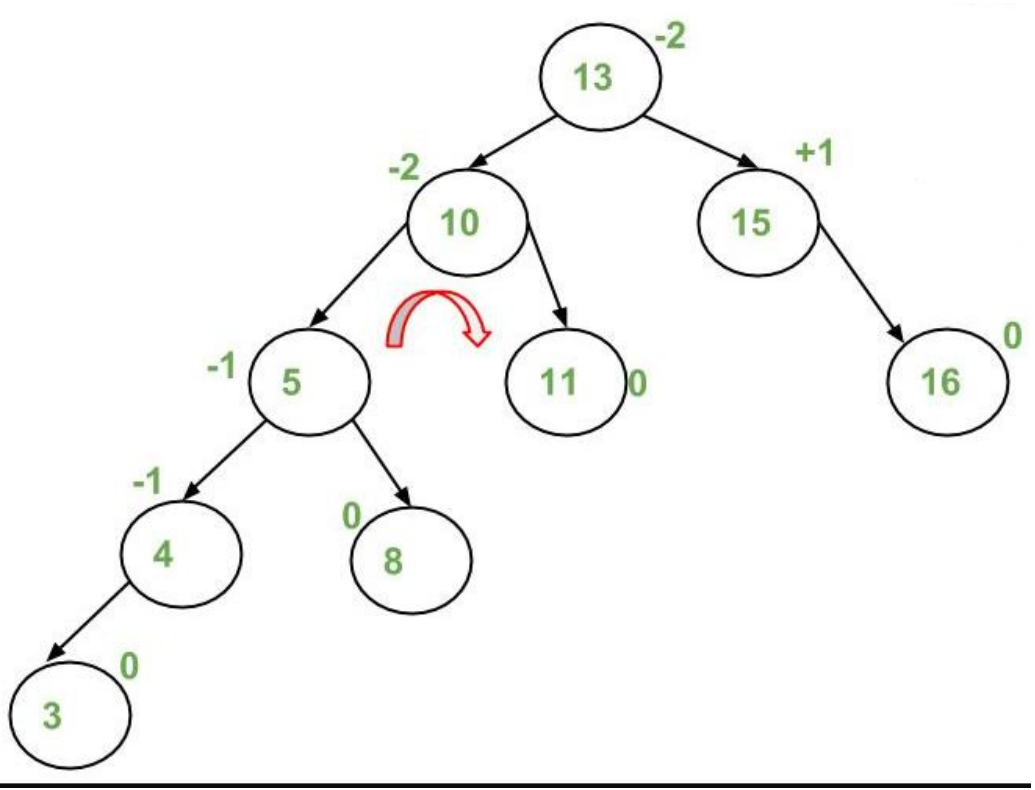
АВЛ-дерево

Добавим элемент 3



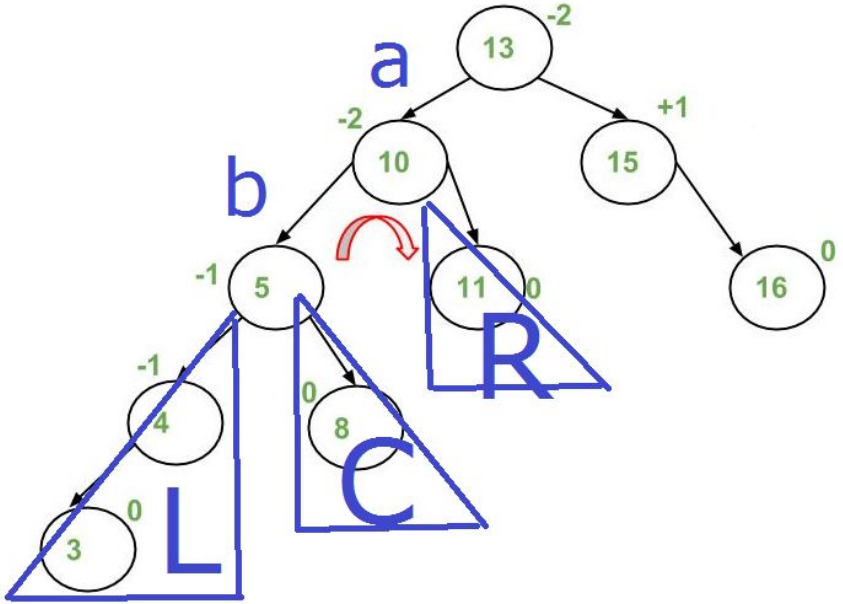
АВЛ-дерево

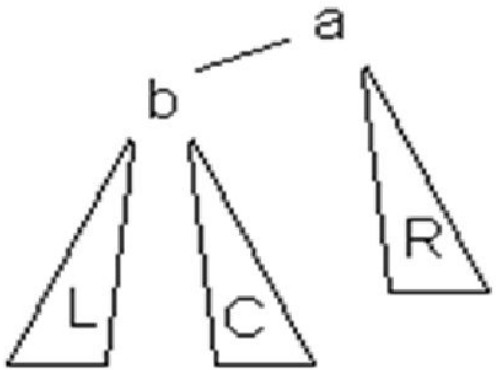
Больше не AVL дерево. Нужно балансировать.



АВЛ-дерево

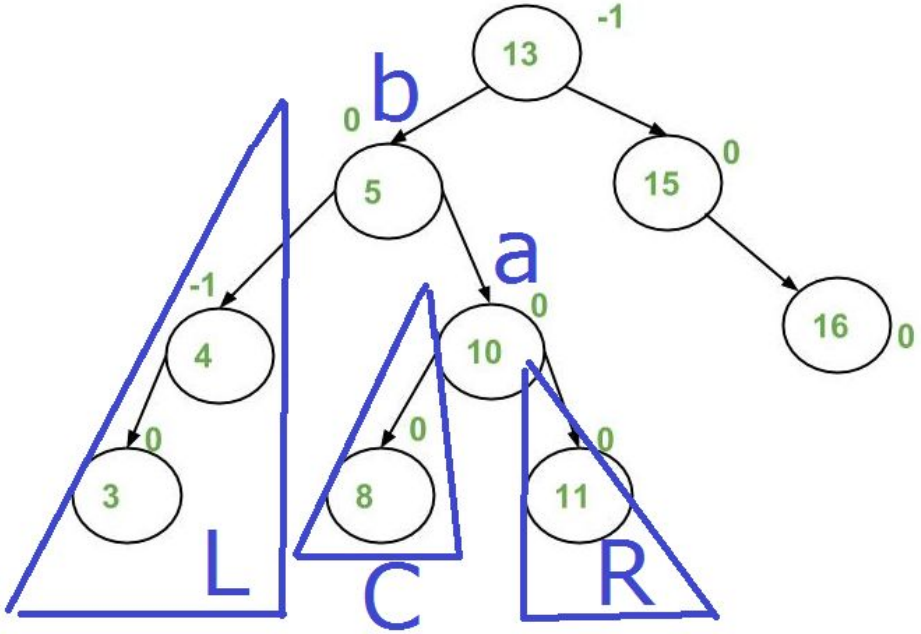
Делаем малый правый поворот

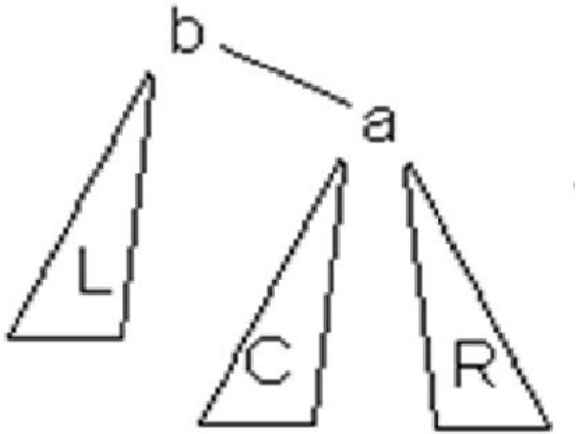




АВЛ-дерево

Снова AVL



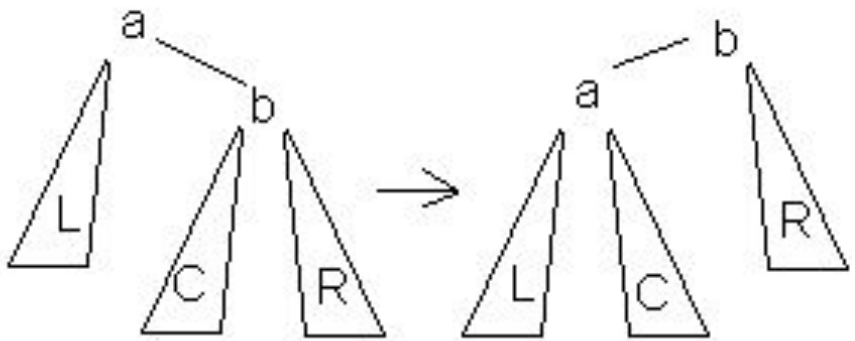


АВЛ-дерево

- **Малое левое вращение**

Используется, когда:
высота(R) = высота(L) + 2 и высота(C) ≤ высота(R).

После операции:
высота дерева останется прежней, если высота(C) = высота(R),
высота дерева уменьшится на 1, если высота(C) < высота(R).

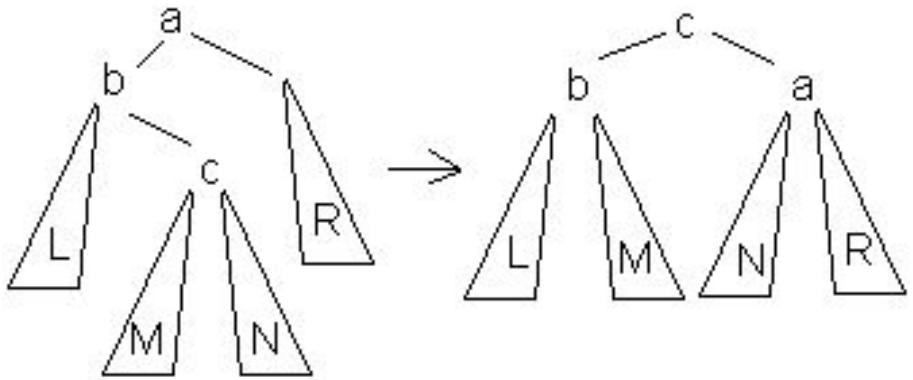


АВЛ-дерево

- **Большое правое вращение**

Используется, когда:
высота(L) = высота(R) + 1 и высота(C) = высота(R) + 2.

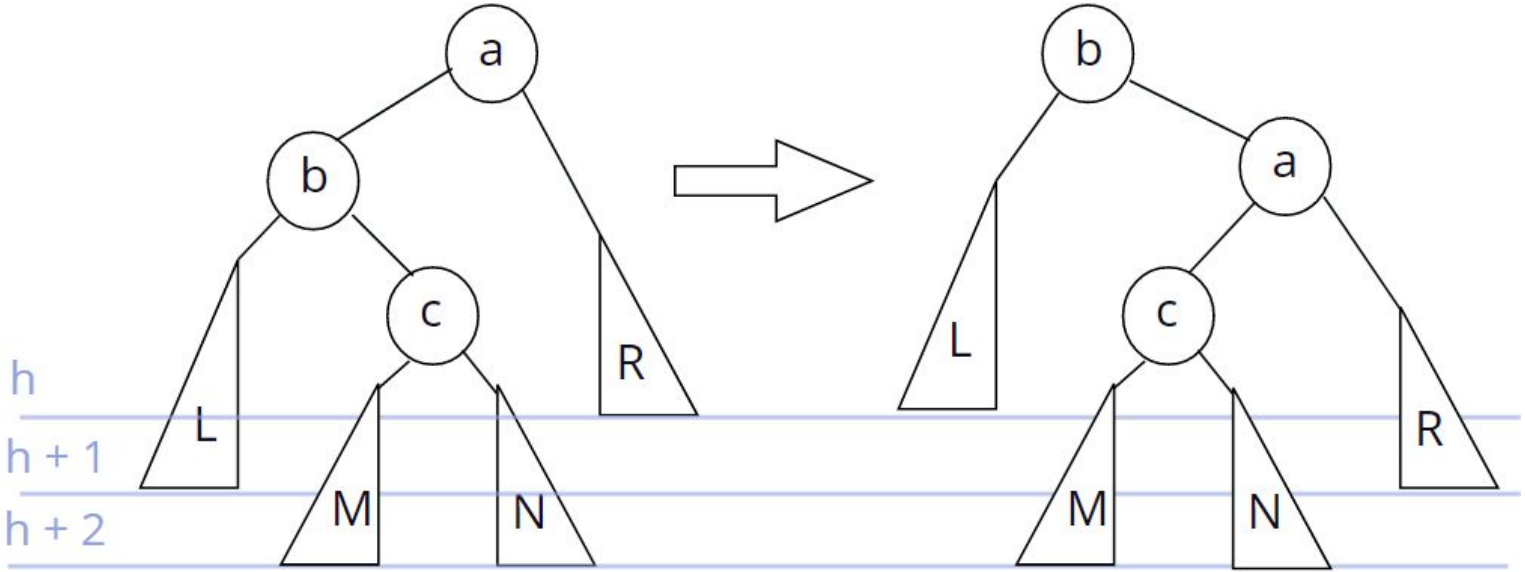
После операции:
высота дерева уменьшается на 1.



АВЛ-дерево

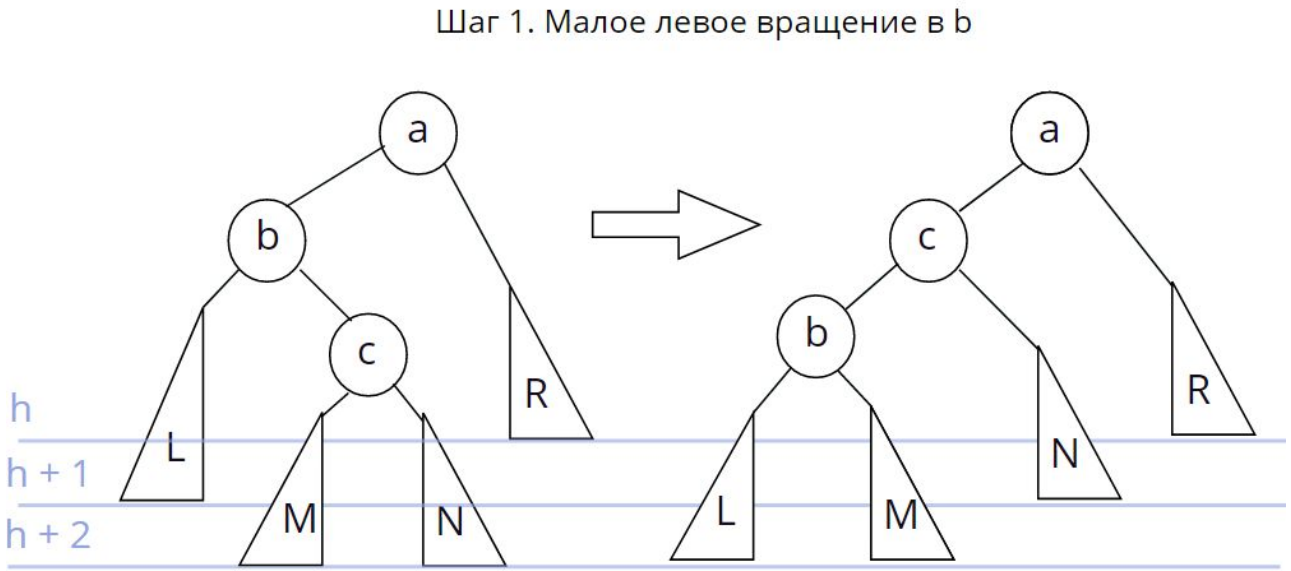
Большое правое вращение. Почему малое правое вращение не поможет.

Разницу высот устранить не смогли! Высота(L) – высота(c) = 2



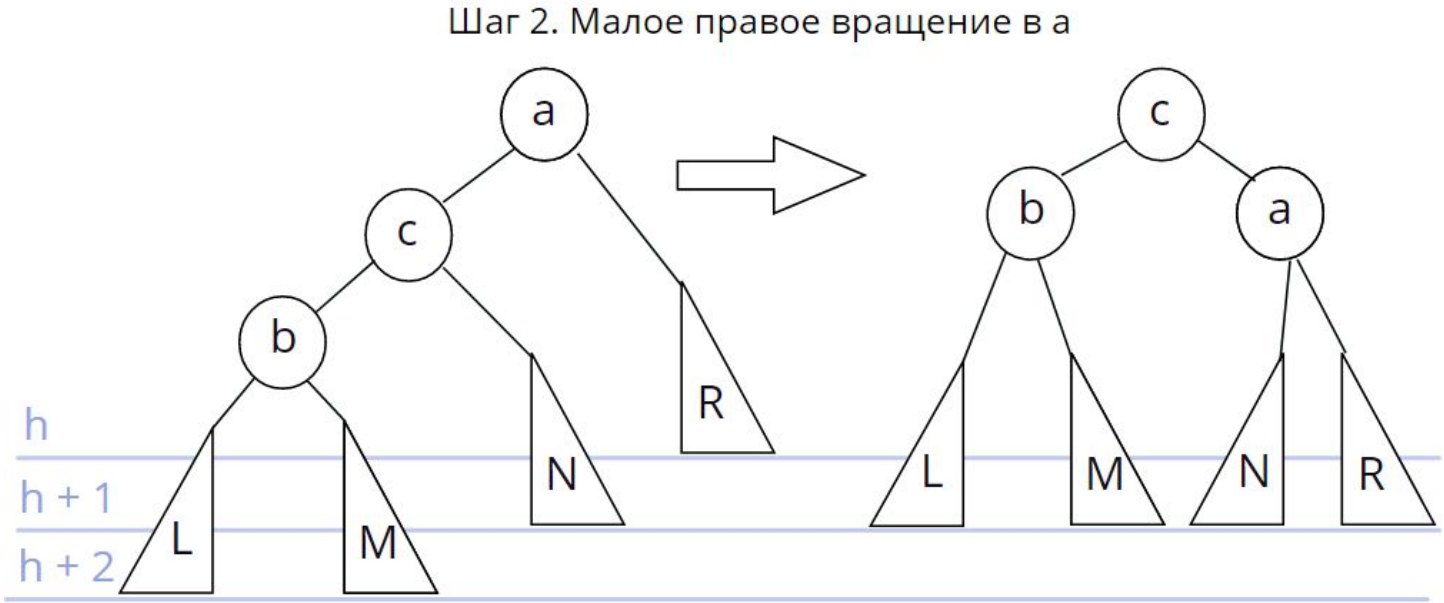
АВЛ-дерево

Большое правое вращение
Также возможны ситуации:
Высота(M) = h + 1, Высота(N) = h + 2
Высота(M) = h + 2, Высота(N) = h + 1



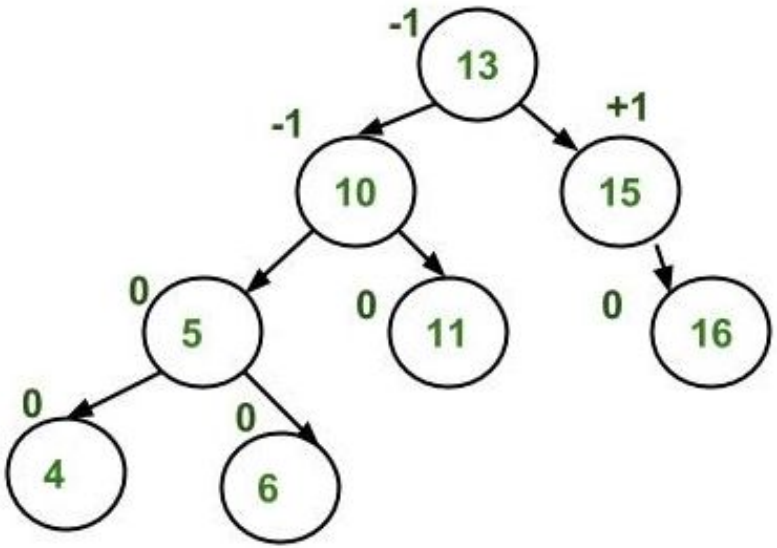
АВЛ-дерево

Большое правое вращение
Если в изначальном дереве высота(M) была h + 1, то теперь она h
Если в изначальном дереве высота(N) была h + 1, то теперь она h
В любом случае, мы исправили дисбаланс и уменьшили высоту на 1



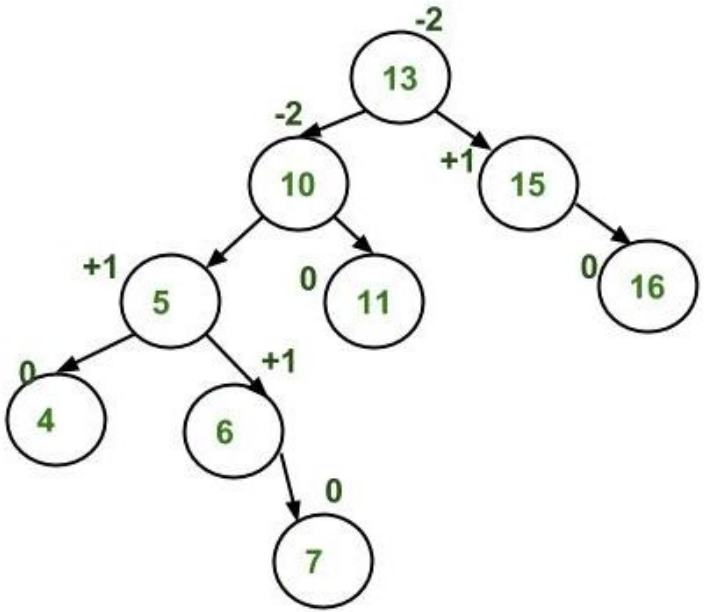
АВЛ-дерево

Добавим элемент **7**



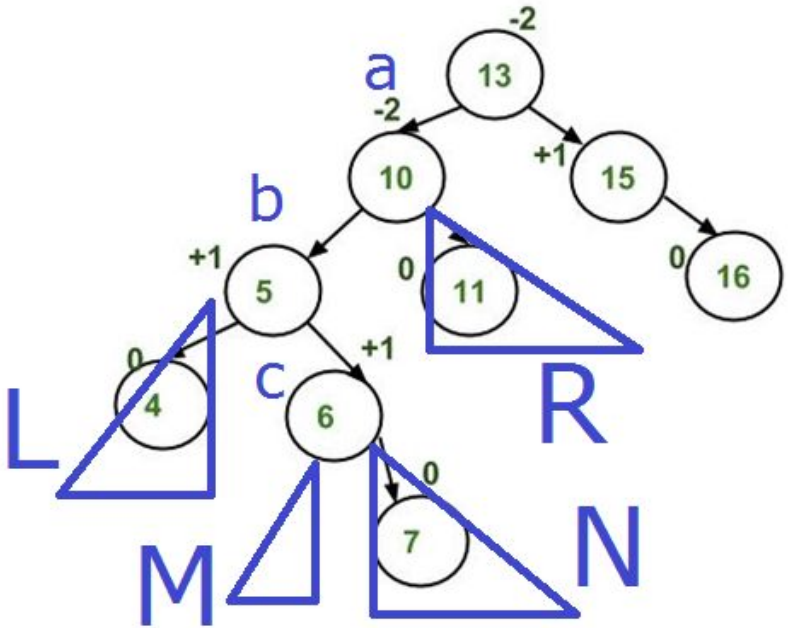
АВЛ-дерево

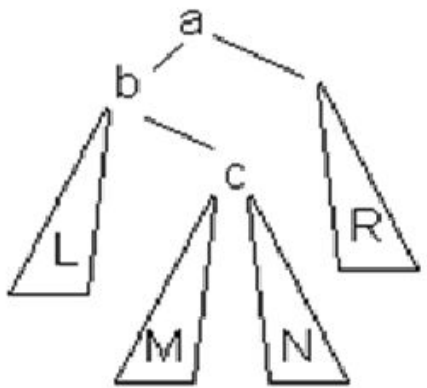
Нарушили баланс - больше не AVL дерево



АВЛ-дерево

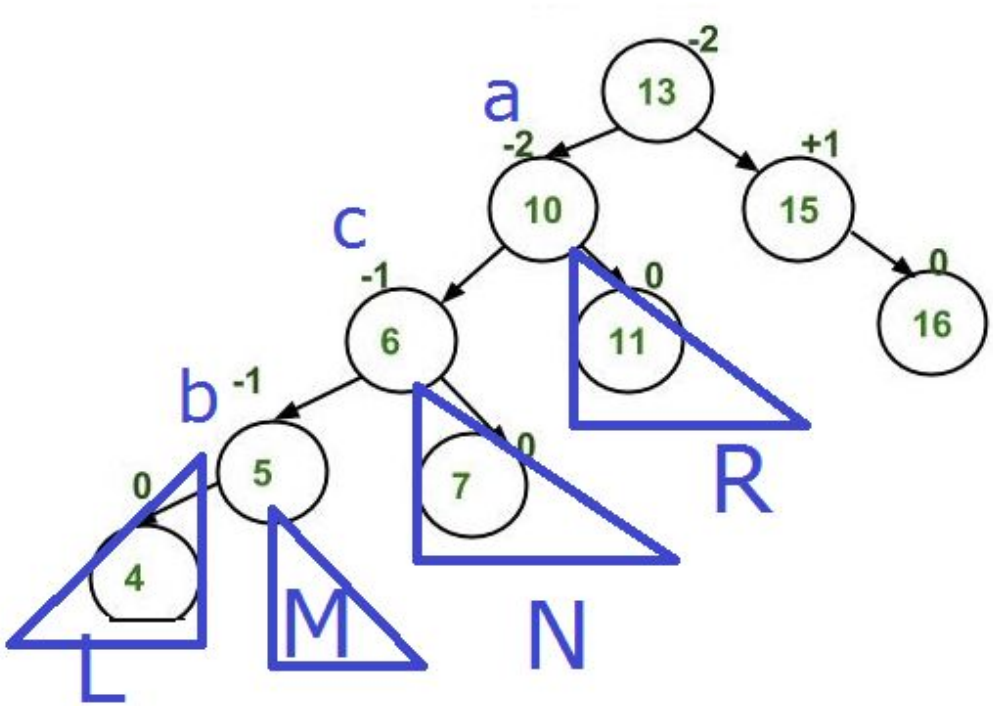
Необходимо **большое правое вращение**:
1. Малое левое вращение в b.
2. Малое правое вращение в a.





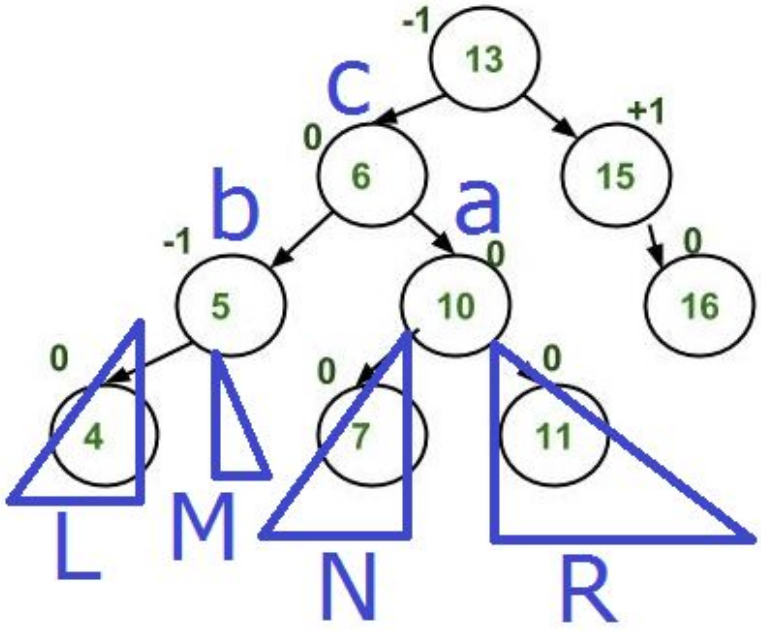
АВЛ-дерево

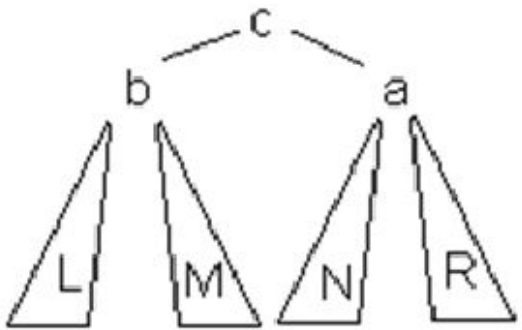
Сделали **малое левое вращение**



АВЛ-дерево

Сделали **малое правое вращение** - снова AVL дерево



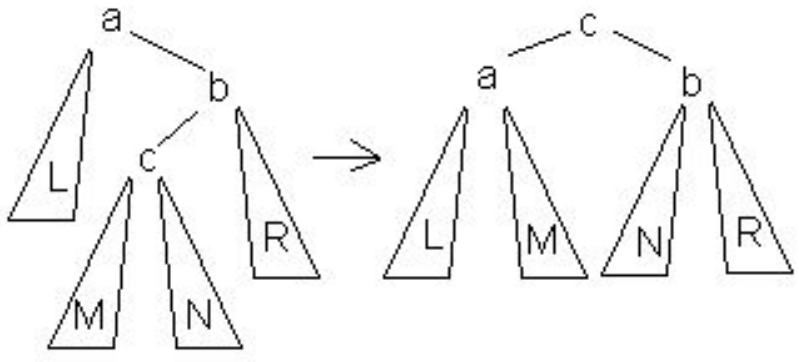


АВЛ-дерево

- **Большое левое вращение**

Используется, когда:
высота(R) = высота(L) + 1 и высота(C) = высота(L) + 2.

После операции:
высота дерева уменьшается на 1.



АВЛ-дерево

Вставка элемента

1. Проходим по пути поиска, пока не убедимся, что ключа в дереве нет;
2. Включаем новую вершину так, как в стандартной операции вставки в дерево поиска;
3. “Отступаем” назад от добавленной вершины к корню, проверяем в каждой вершине сбалансированность, если разность высот поддеревьев равна 2 - выполняем нужное вращение.

Время работы: $O(\log n)$

АВЛ-дерево

Удаление элемента

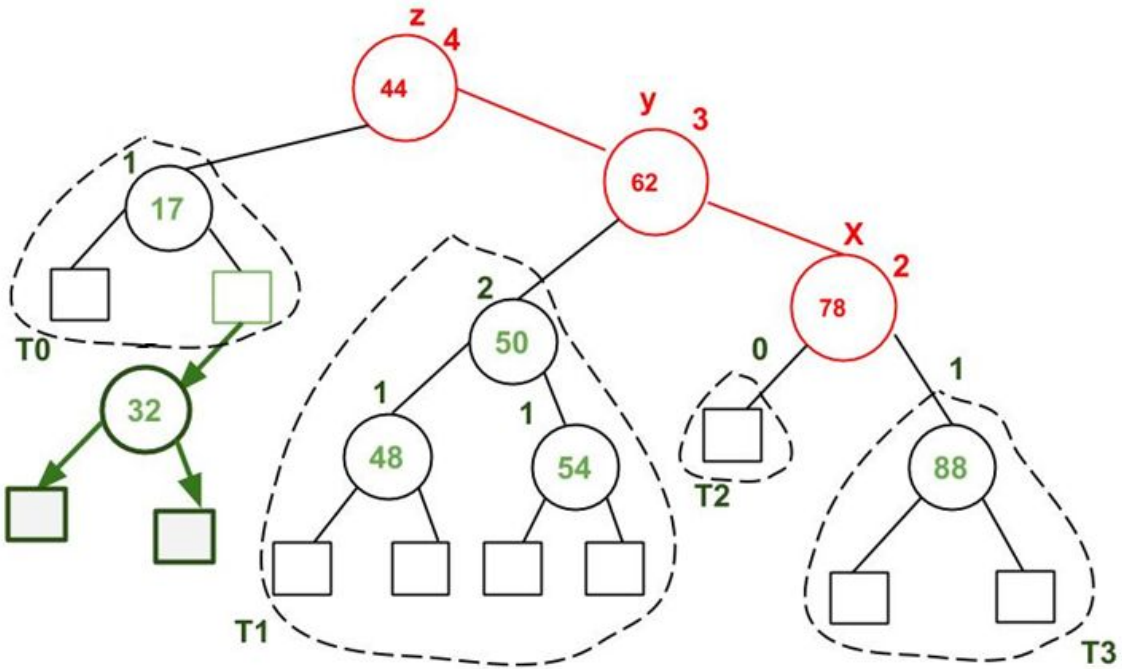
1. Ищем вершину *D*, которую требуется удалить;
2. Проверяем сколько поддеревьев в *D*:
   - если D - лист или у него одно поддерево, то удаляем *D*;
   - если у *D* два поддерева, то ищем вершину *M*, следующую по значению после *D*, как в стандартном алгоритме удаления из дерева поиска, переносим значение из *M* в *D*, удаляем *M*;
3. “Отступаем” назад от удаленной вершины к корню, проверяем в каждой вершине сбалансированность, если разность высот поддеревьев равна 2 - выполняем нужное вращение.

Время работы: $O(\log n)$

АВЛ-дерево

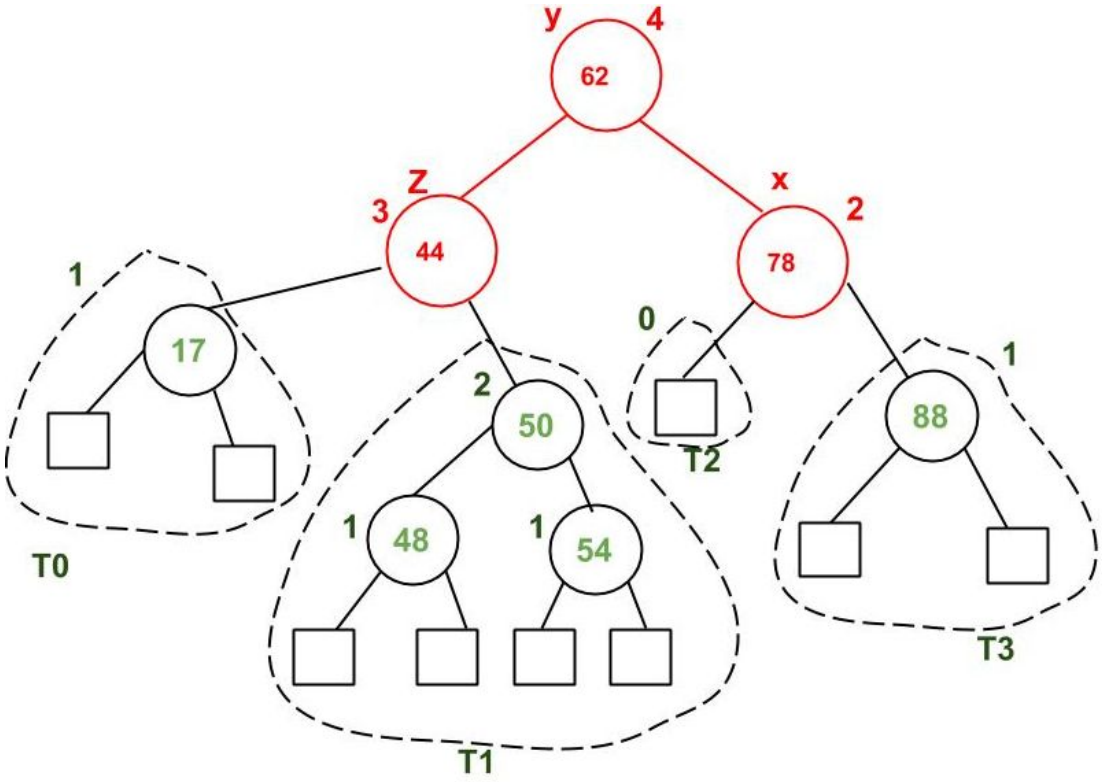
Удаляем элемент 32.

Высота T0 была 2, теперь 1. Нужно малое левое вращение в z.



АВЛ-дерево

Восстановили сбалансированность



АВЛ-дерево

Расход памяти и время работы

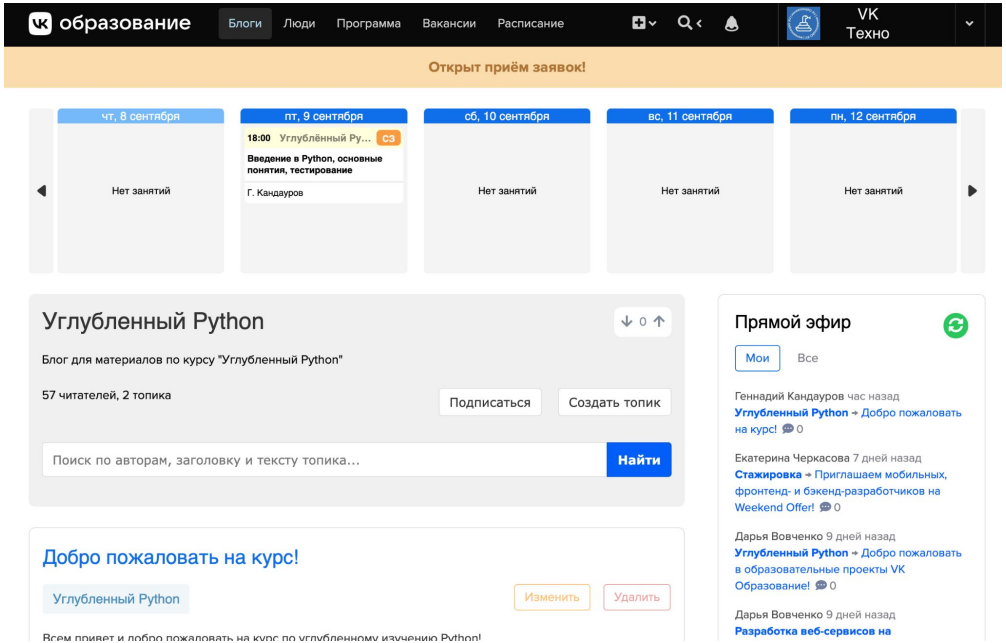
	В среднем случае	В худшем случае
Расход памяти	O(n)	O(n)
Поиск	O(log(n))	O(log(n))
Вставка	O(log(n))	O(log(n))
Удаление	O(log(n))	O(log(n))

Домашнее задание #03

- Реализация AVL-дерева

Напоминание отметиться на портале Vol 2

+ оставить отзыв после лекции



Спасибо за внимание

education