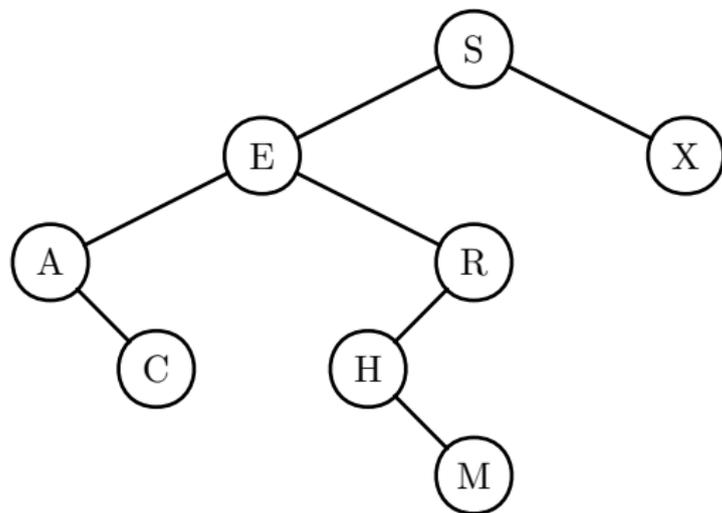


ИВАНОВ А.Б.

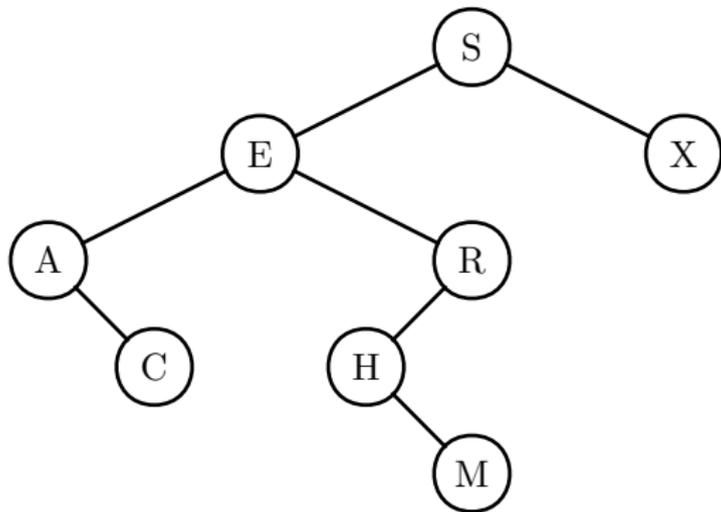
АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

ДВОИЧНЫЕ ДЕРЕВЬЯ ПОИСКА

УДАЛЕНИЕ В BST



УДАЛЕНИЕ В BST



- Tombstone
- Hibbard deletion

УДАЛЕНИЕ ХИББАРДА

УДАЛЕНИЕ ХИББАРДА

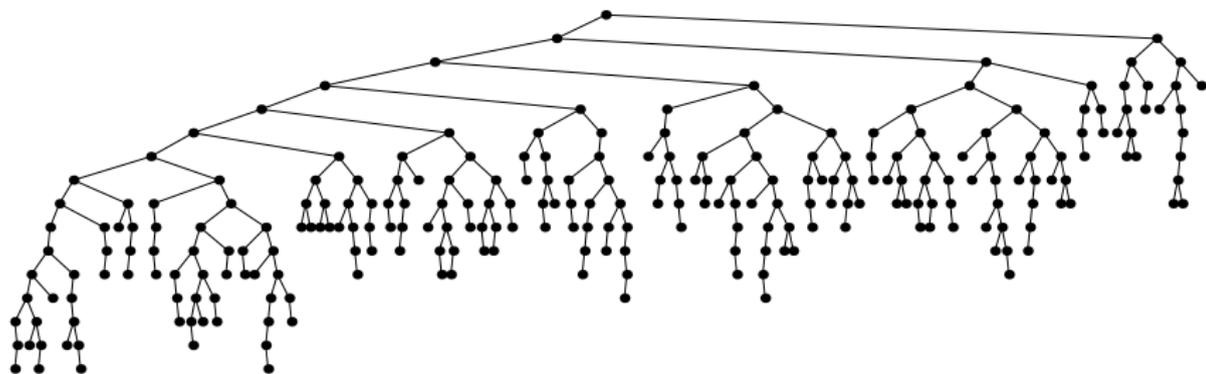
Утв. (Т.Н.Хиббард, 1962) После удаления вершины из случайного двоичного дерева поиска, дерево остается случайным.

(Г.Кнотт, 1972) Первая же вставка после удаления вершины из случайного двоичного дерева поиска может нарушать свойство случайности.

СИМУЛЯЦИЯ С ИСПОЛЬЗОВАНИЕМ УДАЛЕНИЙ

- Добавляем N случайных ключей.
- Цикл N^2 раз:
 - удаляем случайный ключ;
 - добавляем случайный ключ.

$N = 255$ $\max = 16$ $\text{avg} = 9.14$



СИМУЛЯЦИЯ С ИСПОЛЬЗОВАНИЕМ УДАЛЕНИЙ

$$N = 8191 \quad \max = 78 \quad \text{avg} = 34.8$$

$$N = 16383 \quad \max = 102 \quad \text{avg} = 42.2$$

$$N = 32767 \quad \max = 147 \quad \text{avg} = 57.9$$

Факт: высота дерева при такой симуляции $\sim \sqrt{N}$.

СБАЛАНСИРОВАННЫЕ ДЕРЕВЬЯ

Идеально сбалансированное дерево = все пути от корня до конечных вершин имеют одинаковую длину.

Хочется, чтобы \min/\max длины отличались не сильно.

- AVL (Г.М. Адельсон-Вельский, Е.М. Ландис, 1962)

Для любого узла разница между высотами левого и правого поддерева не превосходит 1

- Красно-черные (Rudolf Bayer, 1972)

Термин: Leonidas J. Guibas, Robert Sedgwick, 1978

Количество черных ребер в любом пути одинаково,
количество красных ребер в пути не превосходит количество черных + 1

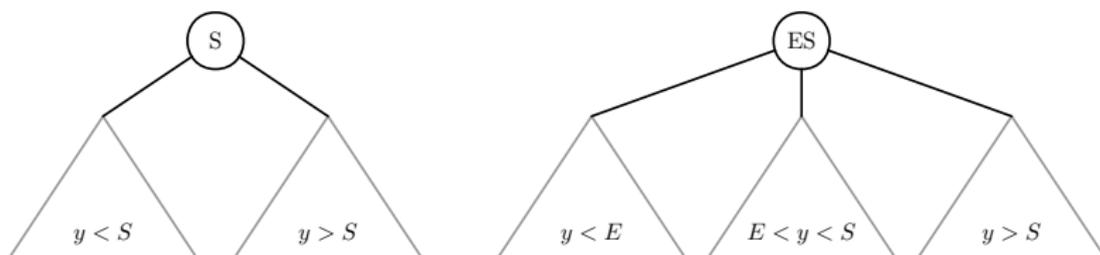
⇒ гарантированная сложность $O(\log N)$.

2-3 ДЕРЕВЬЯ

- Может содержать 2-узлы и 3-узлы
- Идеально сбалансированное

2-узел: содержит 1 ключ, имеет до 2 детей

3-узел: содержит 2 ключа, имеет до 3 детей



2-3 ДЕРЕВЬЯ: ВСТАВКА

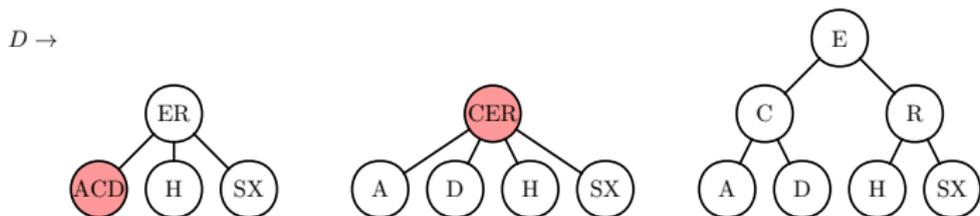
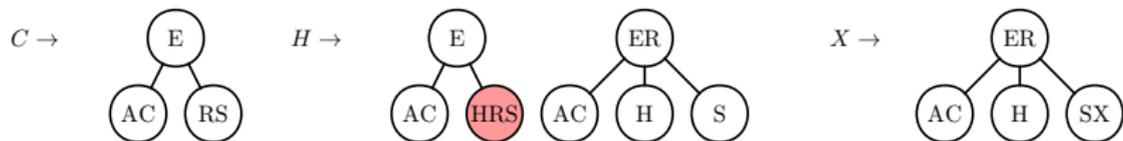
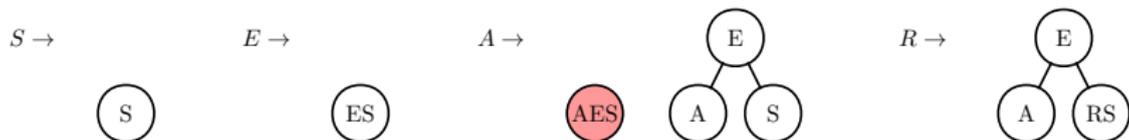
1. Выполняем поиск, находим лист, в который надо вставить ключ.
2. Если лист является 2-узлом, то превращаем его в 3-узел.
3. Если лист является 3-узлом, то превращаем в 4-узел.
4. 4-узел хранит 3 ключа – средний ключ перемещаем в родителя, левый и правый ключи превращаем в 2-узлы.
5. Если родитель был 3-узлом, то он превратился в 4-узел – выполняем для него шаг 4.

2-3 ДЕРЕВО: ПРИМЕР

S, E, A, R, C, H, X, D

2-3 ДЕРЕВО: ПРИМЕР

S, E, A, R, C, H, X, D



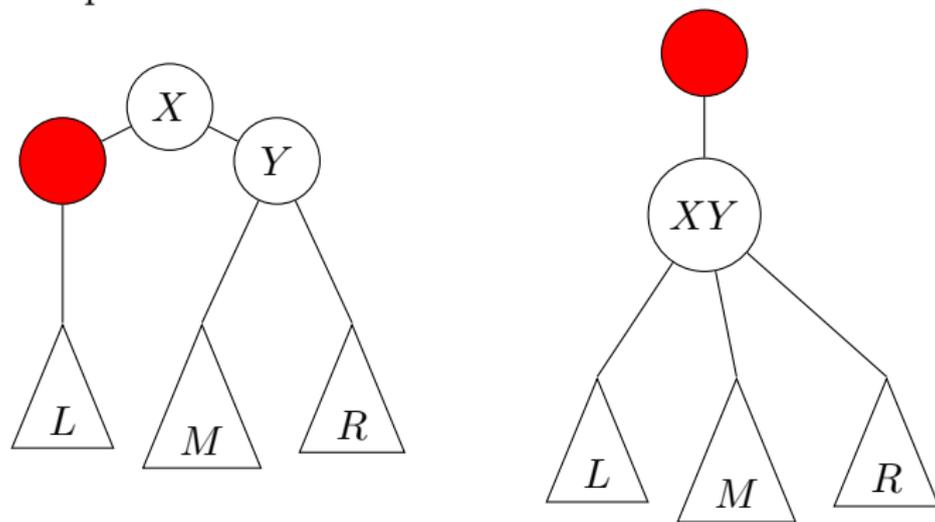
2-3 ДЕРЕВЬЯ: УДАЛЕНИЕ

- удаление ключа из 3 листа: просто удаляем ключ
- удаление ключа из 2 листа: удаляем ключ и остается пустое место
- удаление ключа из внутренней вершины: удаление Хиббарда, но, возможно, остается пустое место вместо листа

В последних двух случаях надо исправить структуру 2-3 дерева.

2-3 ДЕРЕВЬЯ: УДАЛЕНИЕ

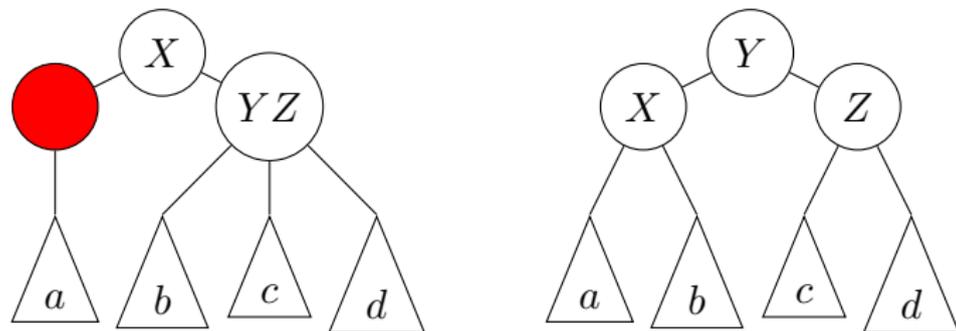
Случай 1: у пустого места родитель и ребенок родителя - 2-вершина.



L, M, R имеют одинаковую высоту.

2-3 ДЕРЕВЬЯ: УДАЛЕНИЕ

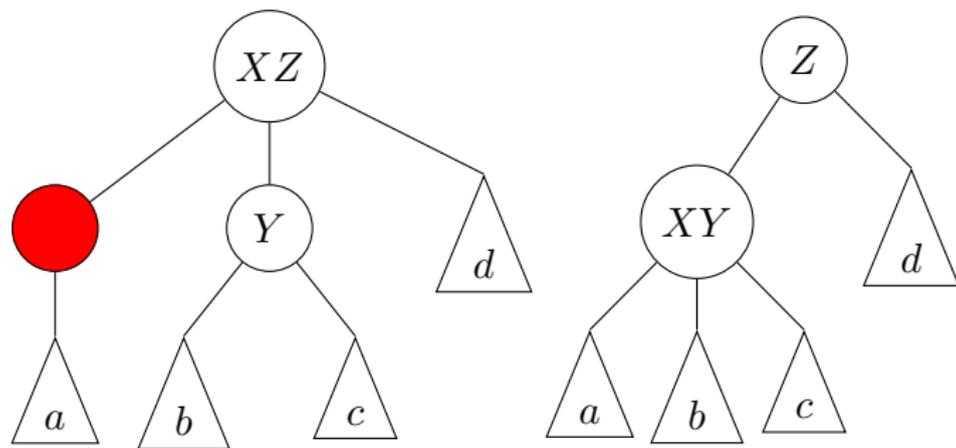
Случай 2: у пустого места 2-родитель и 3-ребенок родителя.



a, b, c, d имеют одинаковую высоту.

2-3 ДЕРЕВЬЯ: УДАЛЕНИЕ

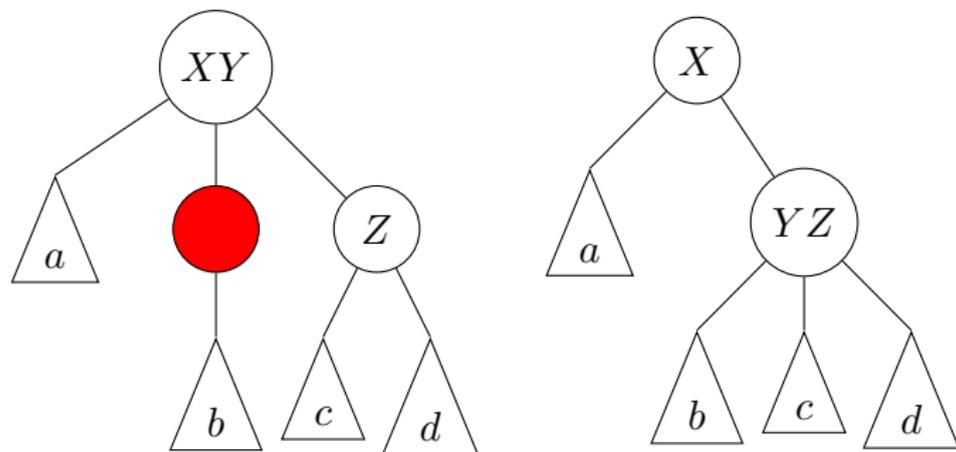
Случай 3а: у пустого места 3-родитель и 2-ребенок родителя.



a, b, c имеют высоту на 1 меньше d .

2-3 ДЕРЕВЬЯ: УДАЛЕНИЕ

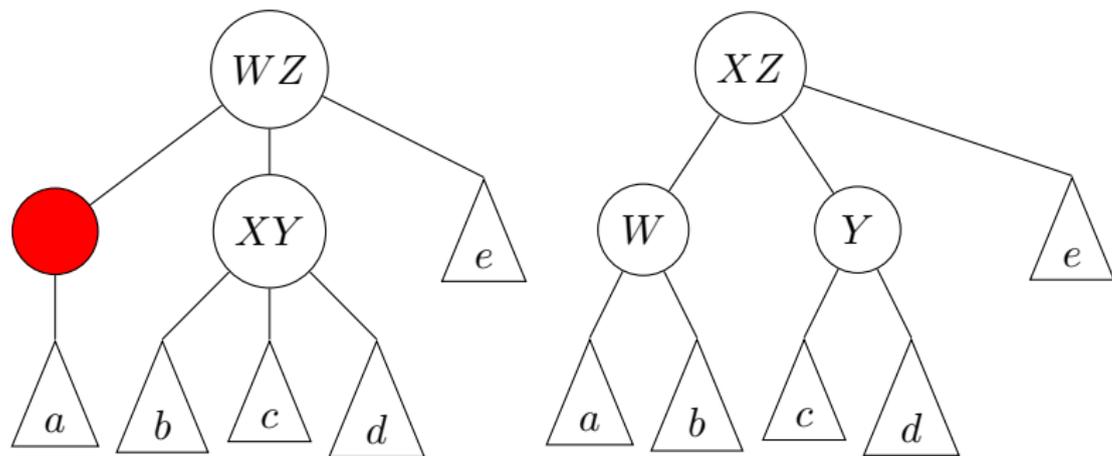
Случай 3б: у пустого места 3-родитель и 2-ребенок родителя.



b, c, d имеют высоту на 1 меньше a .

2-3 ДЕРЕВЬЯ: УДАЛЕНИЕ

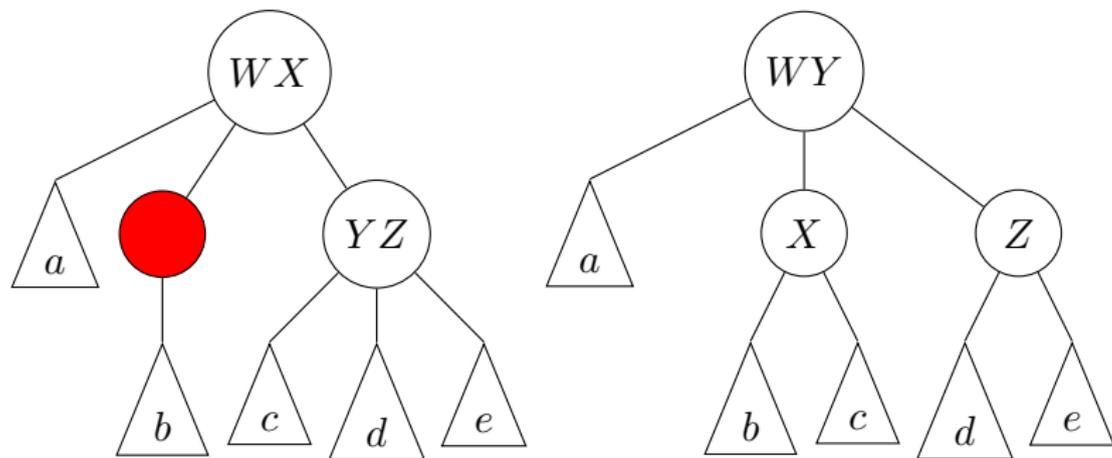
Случай 4а: у пустого места 3-родитель и 3-ребенок родителя.



a, b, c, d имеют высоту на 1 меньше e .

2-3 ДЕРЕВЬЯ: УДАЛЕНИЕ

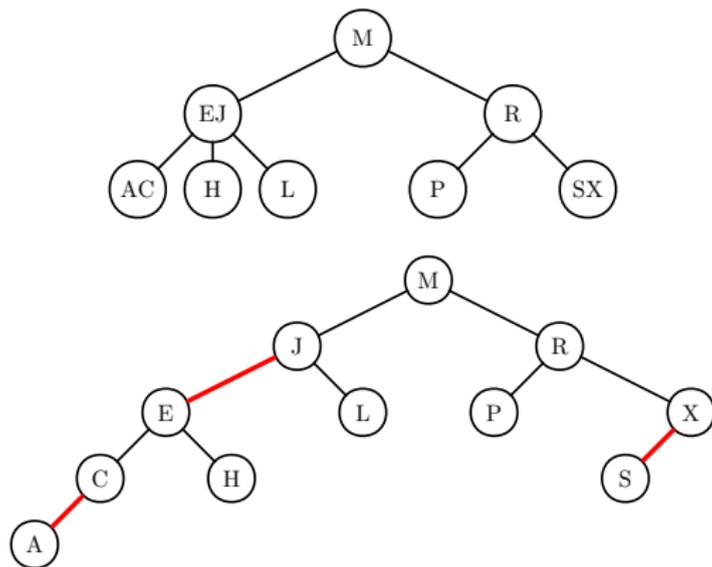
Случай 4б: у пустого места 3-родитель и 3-ребенок родителя.



b, c, d, e имеют высоту на 1 меньше a .

КРАСНО-ЧЕРНЫЕ ДЕРЕВЬЯ

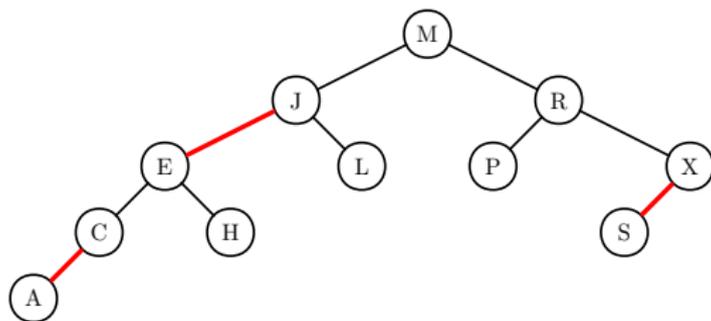
Вставим красное ребро в каждый 3-узел 2-3 дерева.



LLRB = Left-leaning red black trees

LLRB-ДЕРЕВЬЯ

- Количество черных ребер в любом пути одинаково
- Все красные ребра идут налево
- Нет двух последовательных красных ребер
- Взаимнооднозначно соответствуют 2-3 деревьям
- Реализация `get` – как в обычном BST
- Реализация `put` – ...



LLRB-ДЕРЕВЬЯ: РЕАЛИЗАЦИЯ

Утв. LLRB-дерево высоты h содержит по меньшей мере $2^{\frac{h}{2}} - 1$ вершину. ($\Rightarrow h \leq 2 \log(n + 1)$, где n - количество вершин)

В каждом пути от корня до листа есть как минимум $\frac{h}{2}$ черных ребер.

\Rightarrow Существует полное двоичное поддерево высоты $\frac{h}{2}$

\Rightarrow Количество вершин в этом дереве $2^{\frac{h}{2}} - 1$

LLRB-ДЕРЕВЬЯ: РЕАЛИЗАЦИЯ

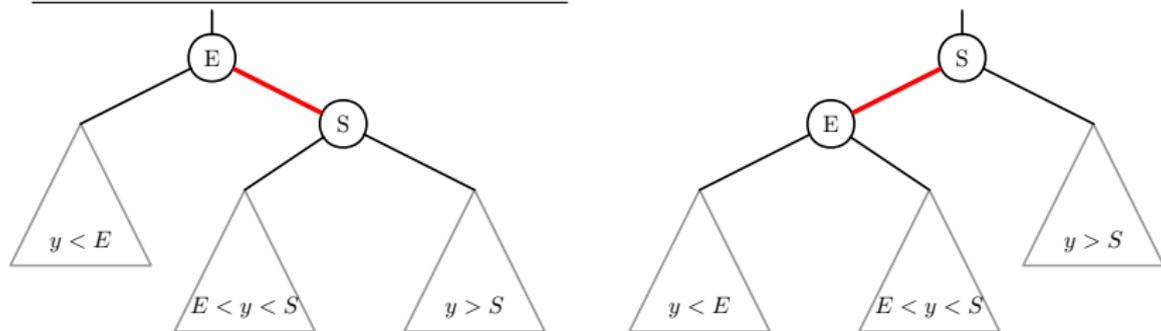
```
private static final boolean RED = true;
private static final boolean BLACK = false;

private class Node
{
    Key key;
    Value val;
    Node left, right;
    boolean color; // color of parent link
}

private boolean isRed(Node x)
{
    if (x == null) return false;
    return x.color == RED;
}
```

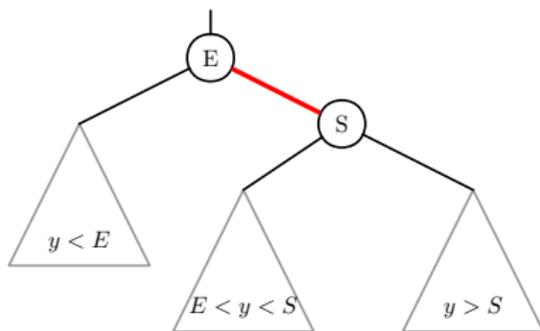
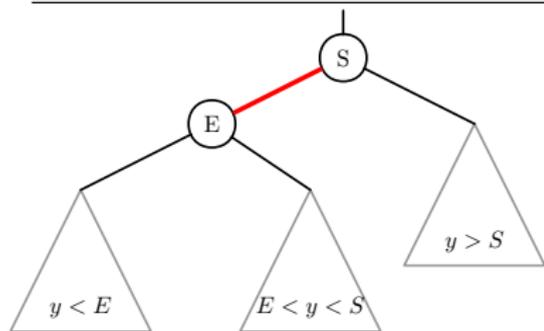
ЭЛЕМЕНТАРНАЯ ОПЕРАЦИЯ-1: rotateLeft

```
private Node rotateLeft(Node h)
{
    // assert isRed(h.right);
    Node x = h.right;
    h.right = x.left;
    x.left = h;
    x.color = h.color;
    h.color = RED;
    return x;
}
```



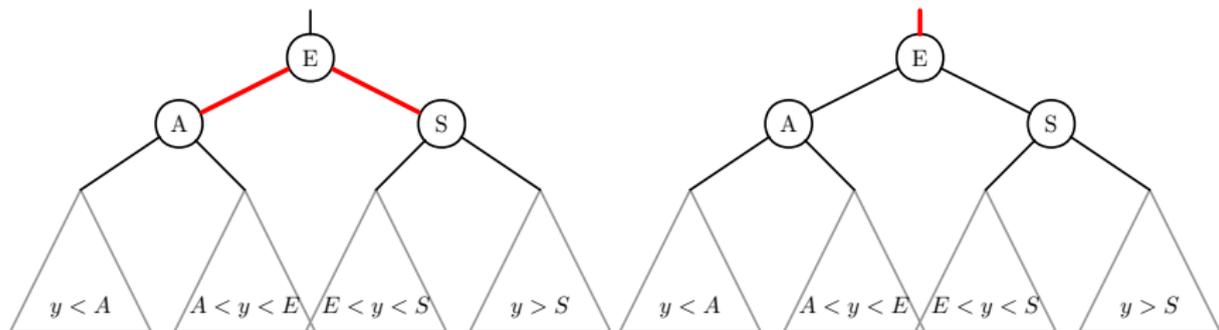
ЭЛЕМЕНТАРНАЯ ОПЕРАЦИЯ-2: rotateRight

```
private Node rotateRight(Node h)
{
    // assert isRed(h.left);
    Node x = h.left;
    h.left = x.right;
    x.right = h;
    x.color = h.color;
    h.color = RED;
    return x;
}
```



ЭЛЕМЕНТАРНАЯ ОПЕРАЦИЯ-3: flipColors

```
private void flipColors(Node h)
{
    // assert !isRed(h);
    // assert isRed(h.left);
    // assert isRed(h.right);
    h.color = RED;
    h.left.color = BLACK;
    h.right.color = BLACK;
}
```



ДОБАВЛЕНИЕ В LLRB С ДВУМЯ УЗЛАМИ

LLRB-ДЕРЕВЬЯ: put

```
private Node put(Node h, Key key, Value val)
{
    if (h == null) return new Node(key, val, RED);

    int cmp = key.compareTo(h.key);
    if (cmp < 0) h.left = put(h.left, key, val);
    else if (cmp > 0) h.right = put(h.right, key, val);
    else if (cmp == 0) h.val = val;

    if (isRed(h.right) && !isRed(h.left)) h = rotateLeft(h);
    if (isRed(h.left) && isRed(h.left.left)) h = rotateRight(h);
    if (isRed(h.left) && isRed(h.right)) flipColors(h);

    return h;
}
```

ДОБАВЛЕНИЕ В LLRB С ДВУМЯ УЗЛАМИ

AVL ДЕРЕВЬЯ

1962 г. - Георгий Максимович Адельсон-Вельский, Евгений Михайлович Ландис

Инвариант: Для любого узла разница между высотами левого и правого поддерева не превосходит 1.

Утв. Высота AVL дерева - $O(\log n)$

Доказательство Пусть N_h - минимальное количество вершин в AVL дереве высоты h . Чему равно N_h ?

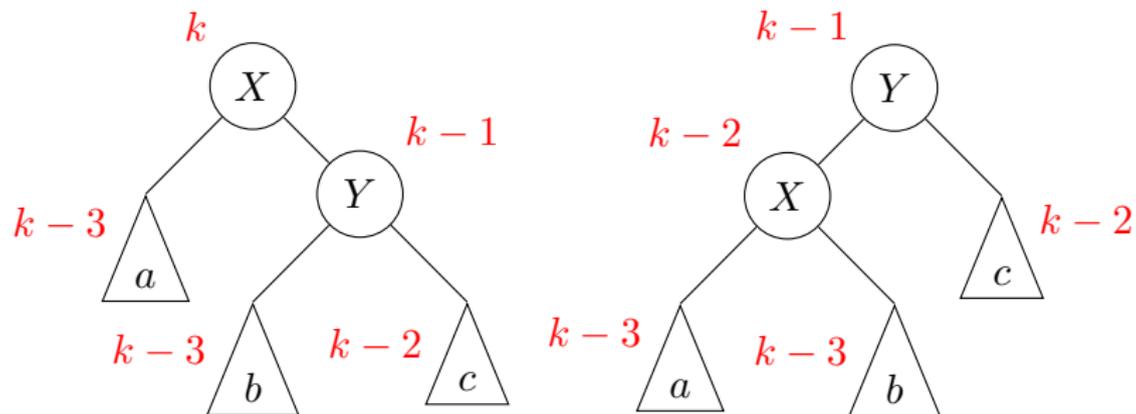
AVL ДЕРЕВЬЯ, ВСТАВКА

Для восстановления свойства AVL дерева после вставки используются уже известные нам операции - поворот влево и поворот вправо.

Пусть X самый "нижний" узел, в котором нарушено свойство пусть правое поддерево x выше левого. Вершина правого поддерева - Y .

AVL-ДЕРЕВО ВСТАВКА

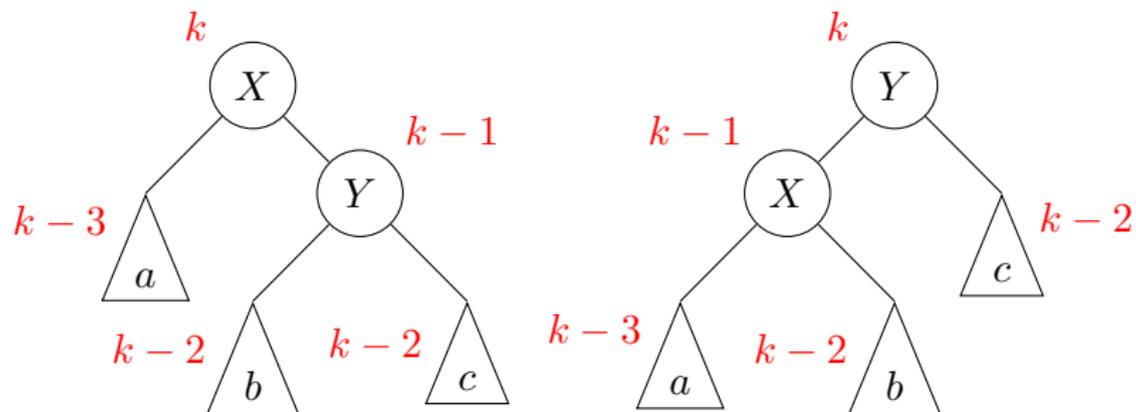
Случай 1: правое поддереву Y выше левого.



Выполняем левый поворот.

AVL-ДЕРЕВО ВСТАВКА

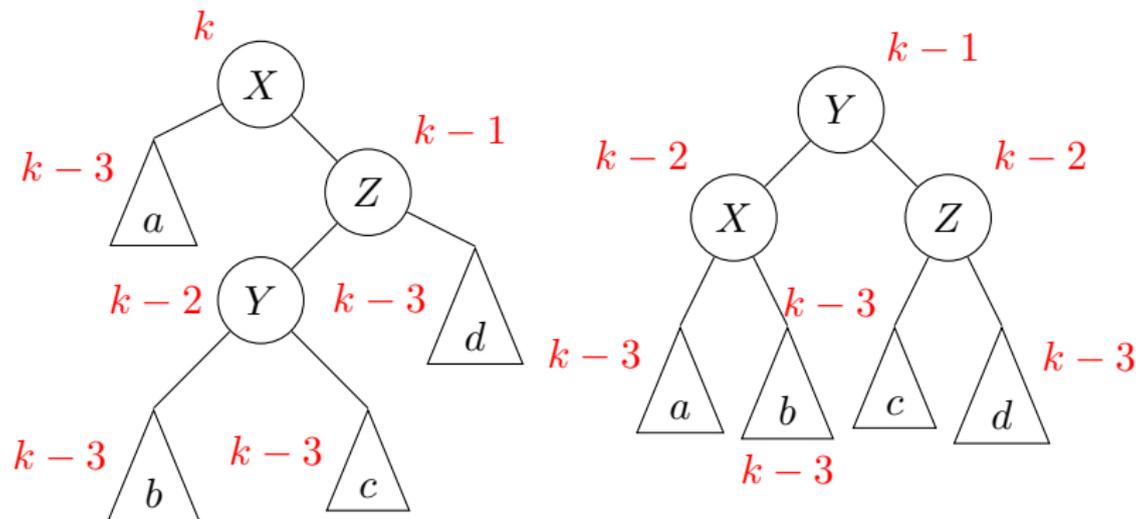
Случай 2: поддеревья Y одинаковой высоты.



Выполняем левый поворот.

AVL-ДЕРЕВО ВСТАВКА

Случай 3: левое поддерево Y выше правого.

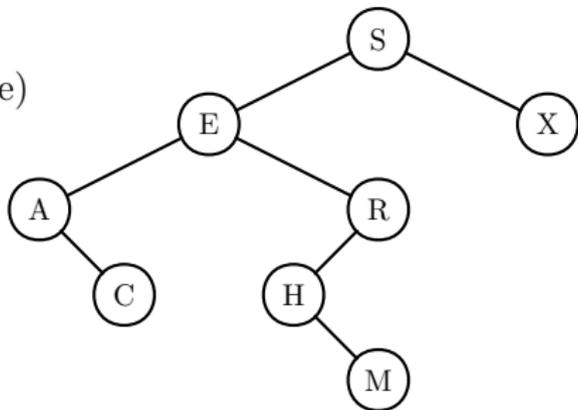


Выполняем правый поворот. Затем выполняем левый поворот. (Есть еще несколько случаев в зависимости от балансировки дерева с корнем Y .)

ОПЕРАЦИИ В BST, ИСПОЛЬЗУЮЩИЕ ПОРЯДОК

В силу использования сравнений – у нас есть дополнительный набор операций, использующих порядок (не входящих в интерфейс ассоциативного массива)

- min / max
- floor / ceiling
- rank
- in-order traversal
- поиск по диапазону (range)



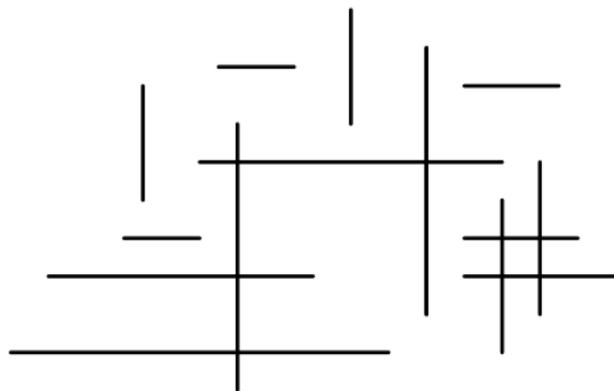
ГЕОМЕТРИЧЕСКИЕ ПРИЛОЖЕНИЯ

- Пересечения перпендикулярных отрезков
- Поиск в 2-мерном диапазоне
- Поиск ближайшего соседа
- Интервальное дерево поиска
- Пересечения перпендикулярных прямоугольников

ПЕРЕСЕЧЕНИЯ ПЕРПЕНДИКУЛЯРНЫХ ОТРЕЗКОВ

Дано: N горизонтальных и вертикальных отрезков.

Найти: все точки пересечения.



ПОИСК В 2-МЕРНОМ ДИАПАЗОНЕ

Дано: N точек на плоскости.

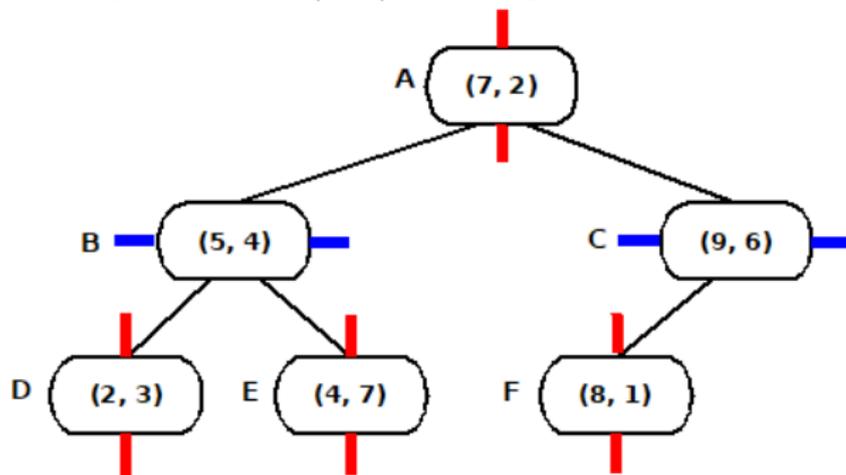
Запрос: Для прямоугольника $[x_1, x_2] \times [y_1, y_2]$ найти все точки, лежащие в нем.

2D-ДЕРЕВО

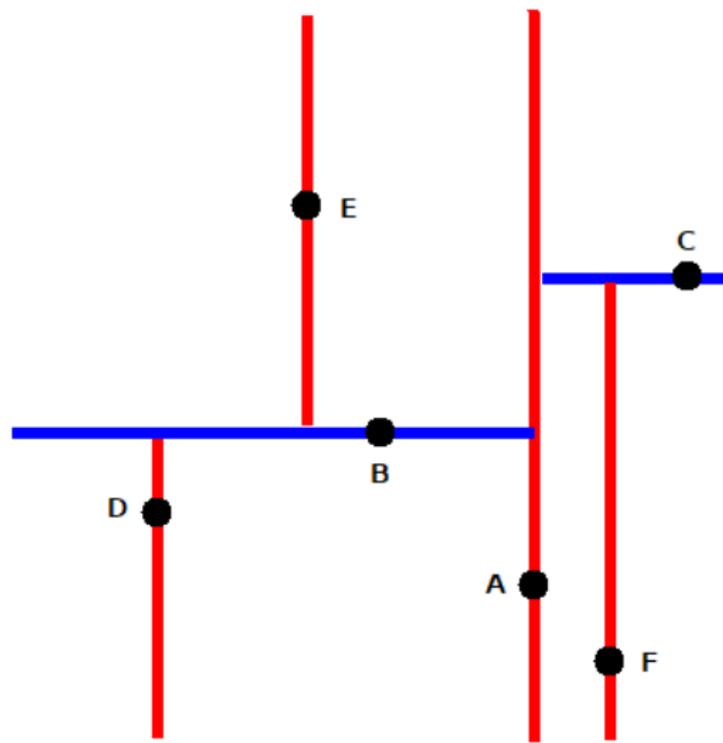
В вершинах храним пары ключей (a, b) .

В вершинах на уровнях 0, 2, 4, ... делим по координате x : в левом поддереве точки (x, y) с x -координатой $x < a$, в правом – $x \geq a$.

В вершинах на уровнях 1, 3, 5, ... делим по координате y : в левом поддереве точки (x, y) с y -координатой $y < b$, в правом – $y \geq b$.



2D-ДЕРЕВО



ПОИСК БЛИЖАЙШЕГО СОСЕДА

Дано: N точек на плоскости.

Запрос: Для точки (x, y) найти ближайшую точку из этих N .

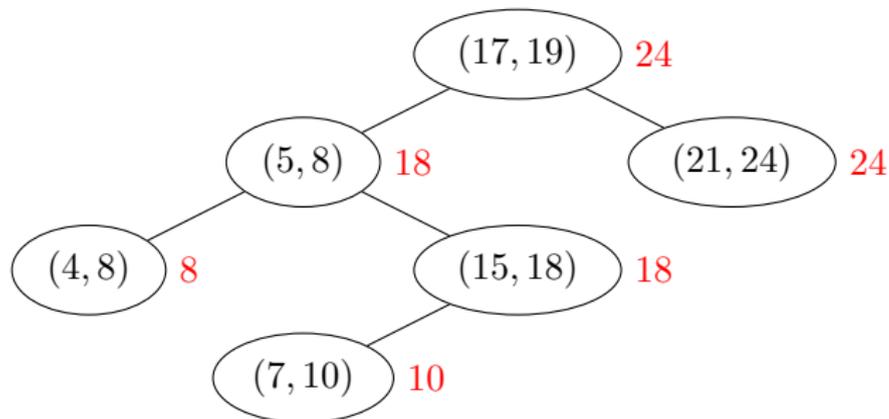
ИНТЕРВАЛЬНОЕ ДЕРЕВО ПОИСКА

Структура данных для хранения (перекрывающихся) интервалов.

- Добавить интервал (a, b)
- Найти интервал (a, b)
- Удалить интервал (a, b)
- Для заданного интервала (a, b) найти все пересекающиеся с ним интервалы, хранящиеся в нашей структуре.

ИНТЕРВАЛЬНОЕ ДЕРЕВО ПОИСКА

- Левый конец интервала используем как ключ
- Дополнительно храним максимальный правый конец в поддереве



ИНТЕРВАЛЬНОЕ ДЕРЕВО ПОИСКА

Алгоритм поиска интервала в интервальном дереве, пересекающегося с (a, b) :

- Если текущий узел пересекается с (a, b) , возвращаем его
- Иначе, если левое поддерево пустое, идем направо
- Иначе, если максимальный правый конец в левом поддереве меньше a , идем направо
- Иначе, идем налево

ПЕРЕСЕЧЕНИЯ ПРЯМОУГОЛЬНИКОВ

Дано: N прямоугольников $[x_1, x_2] \times [y_1, y_2]$.

Найти: все пары пересекающихся прямоугольников.