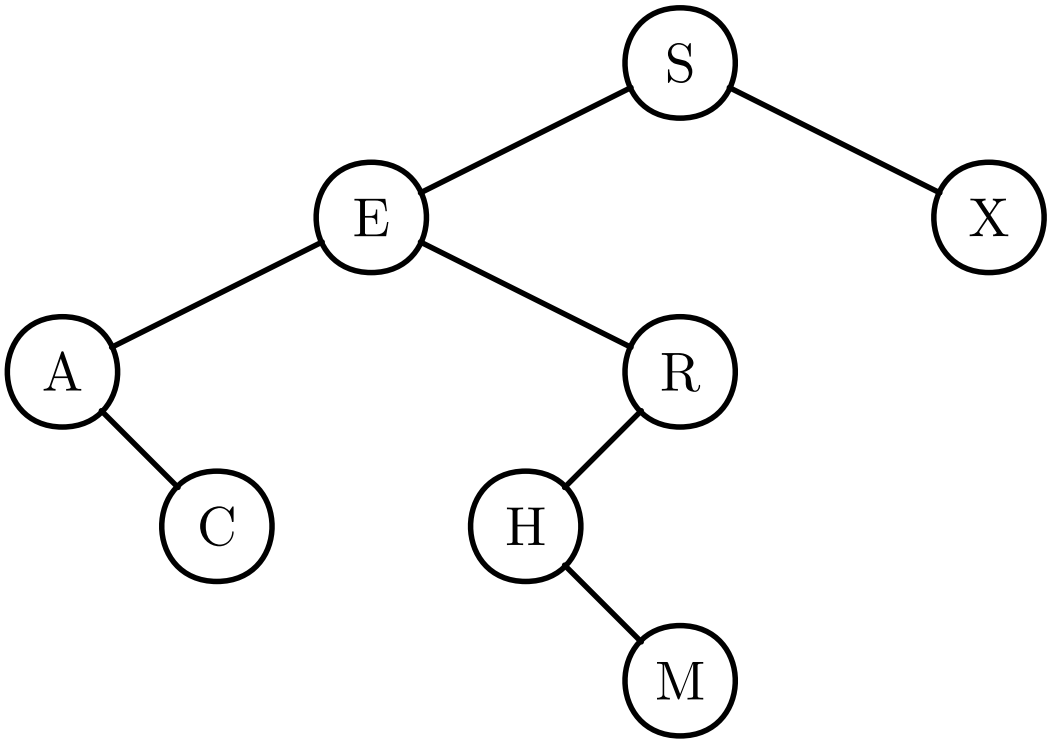
Адрианов Н.М.
Иванов А.Б.

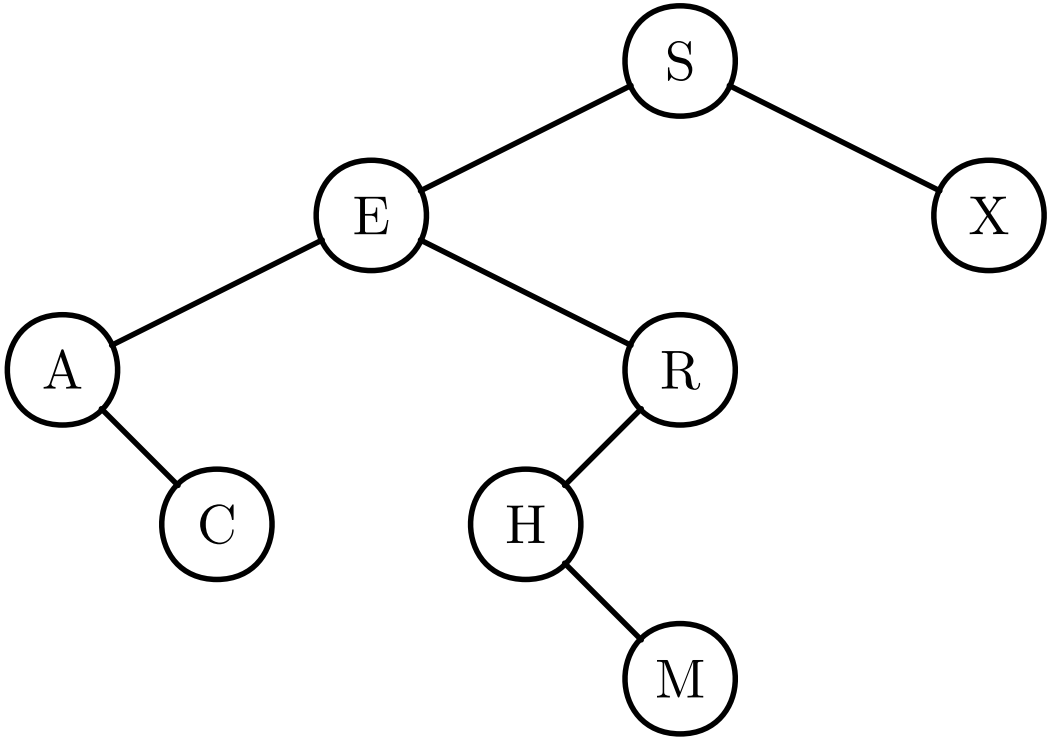
Алгоритмы и структуры данных

Абстрактные типы данных
Структуры данных
Ассоциативные массивы

Удаление в BST



Удаление в BST

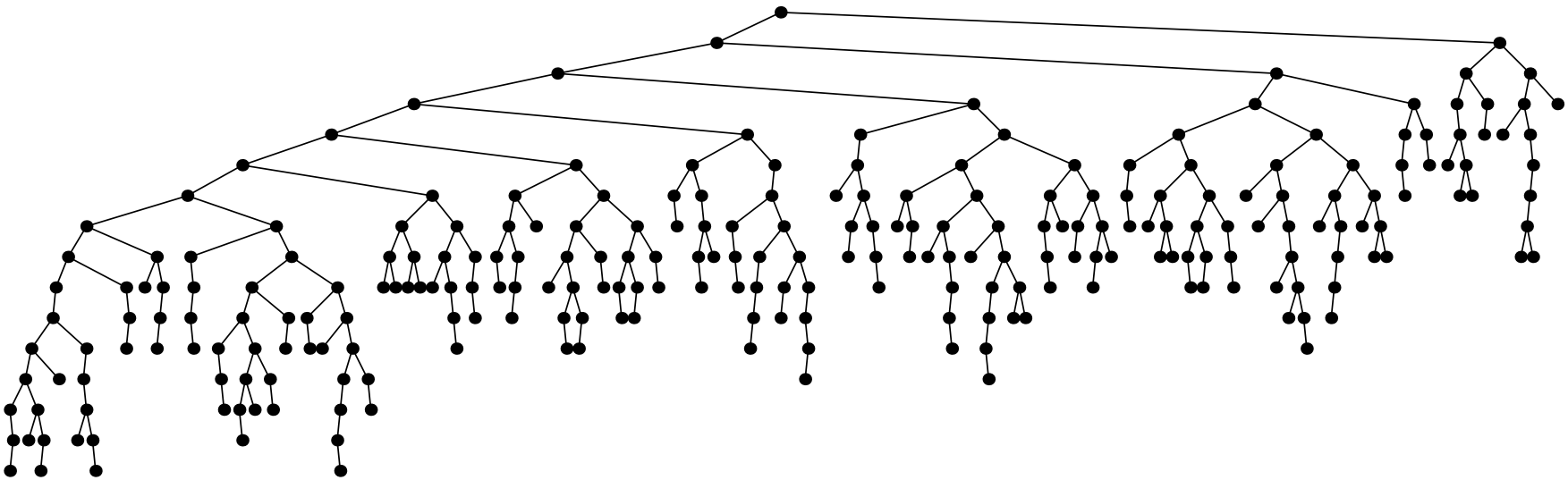


- Tombstone
- Hibbard deletion

Симуляция с использованием удалений

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

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



Симуляция с использованием удалений

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

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

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

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 Sedgewick, 1978

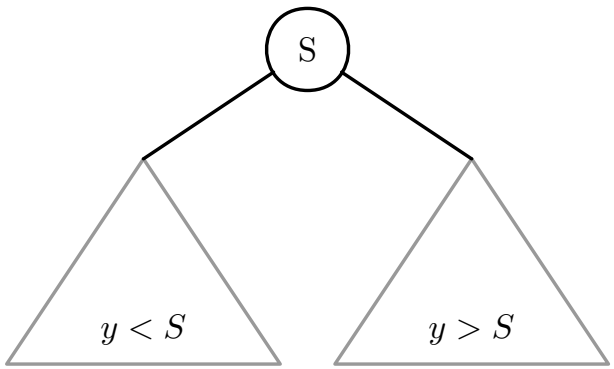
  Количество черных ребер в любом пути одинаково,
  количество красных ребер в пути не превосходит количество черных + 1

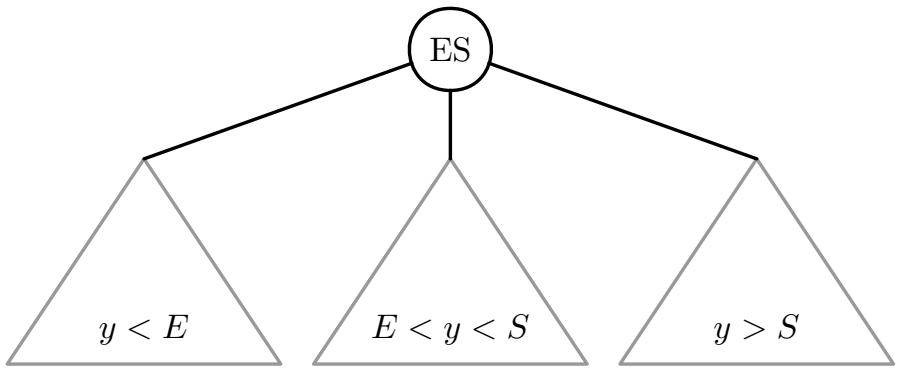
$\Rightarrow$ гарантированная сложность $O(\log N)$.

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

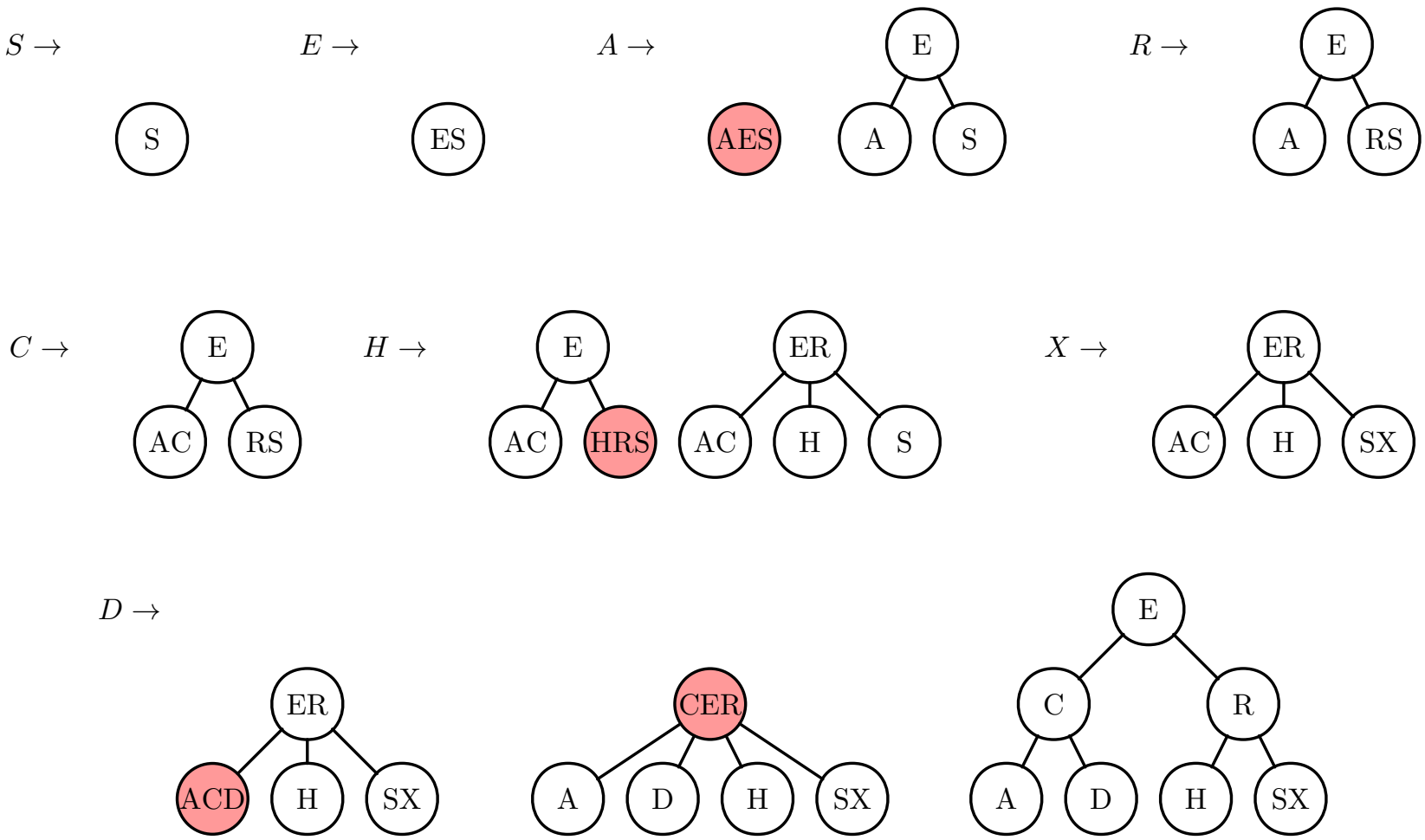
1. Выполняем поиск, находим лист, в который надо вставить ключ.
2. Если лист является 2-узлом, то превращаем его в 3-узел.
3. Если лист является 3-узлом, то превращаем в 4-узел.
4. 4-узел хранит 3 ключа – средний ключ перемещаем в родителя, левый и правый ключи превращаем в 2-узлы.
5. Если родитель был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



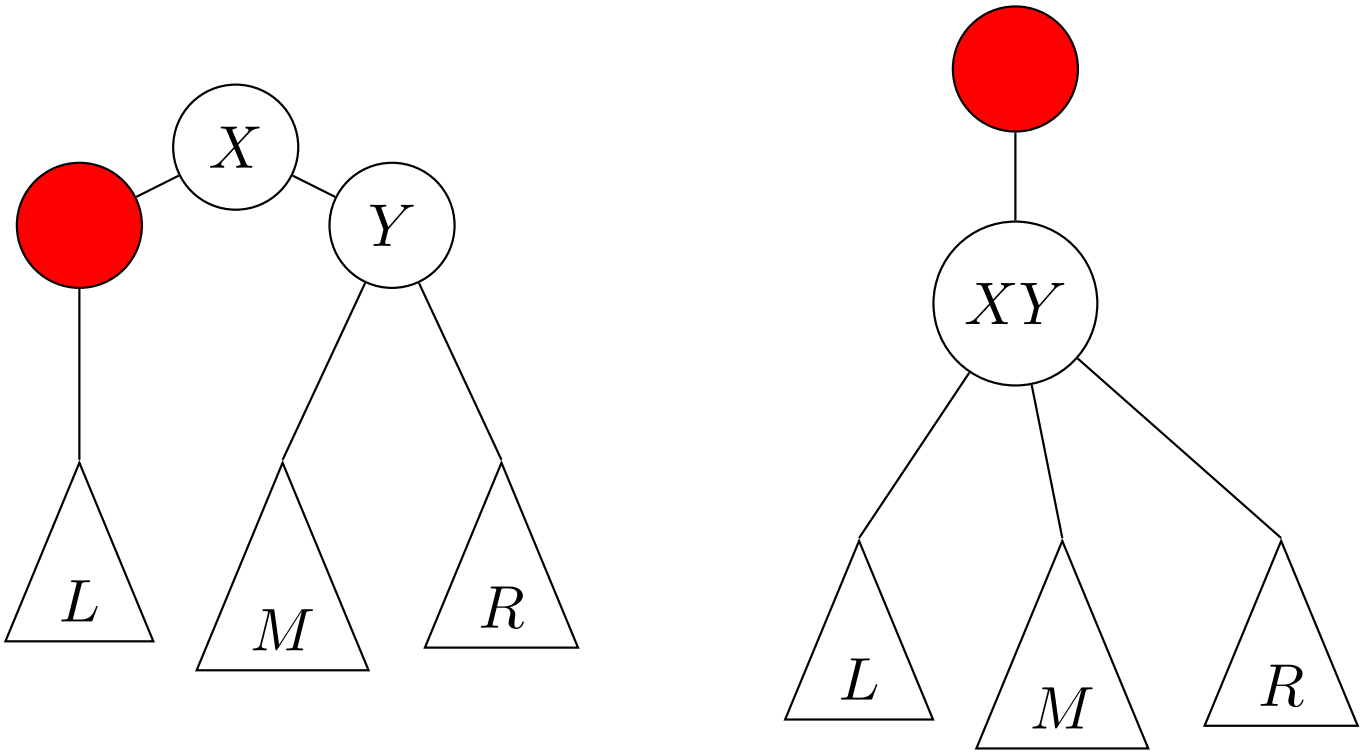
2-3 деревья: удаление

- 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

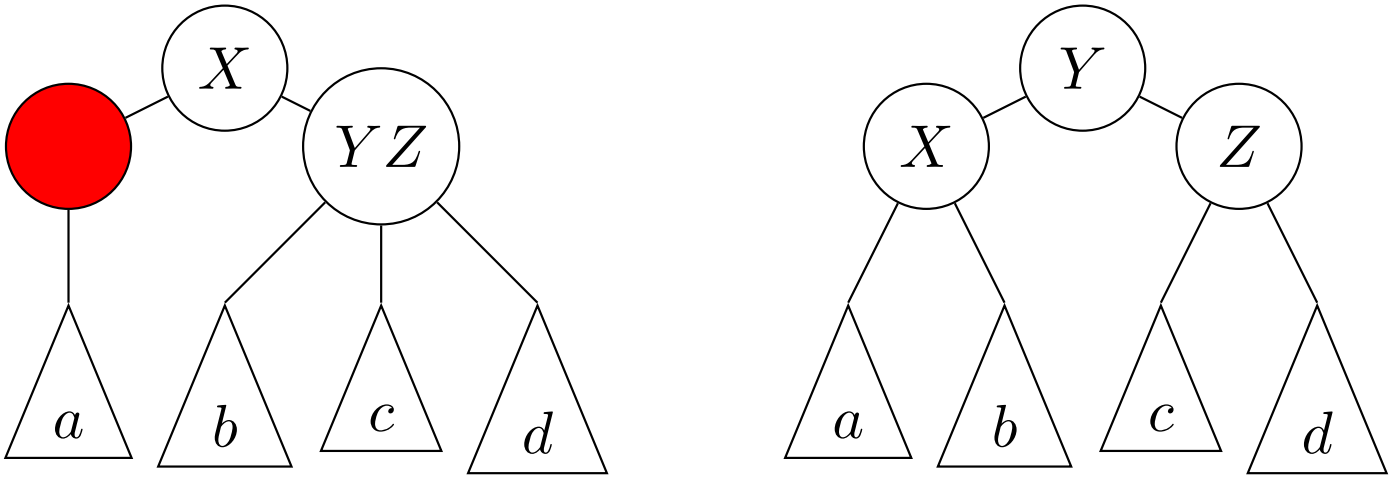
Случай 1: у пустого места родитель и ребенок родителя - 2-вершина.



L, M, R имеют одинаковую высоту.

2-3 деревья: удаление

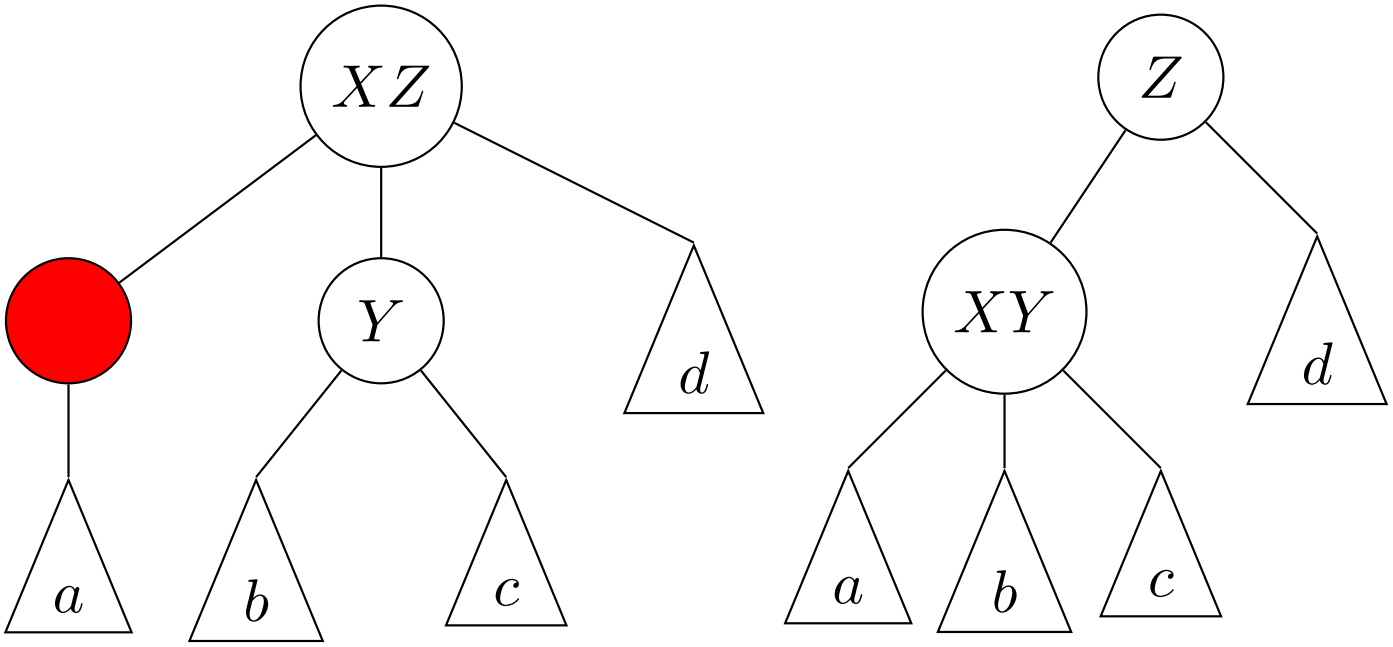
Случай 2: у пустого места 2-родитель и 3-ребенок родителя.



a, b, c, d имеют одинаковую высоту.

2-3 деревья: удаление

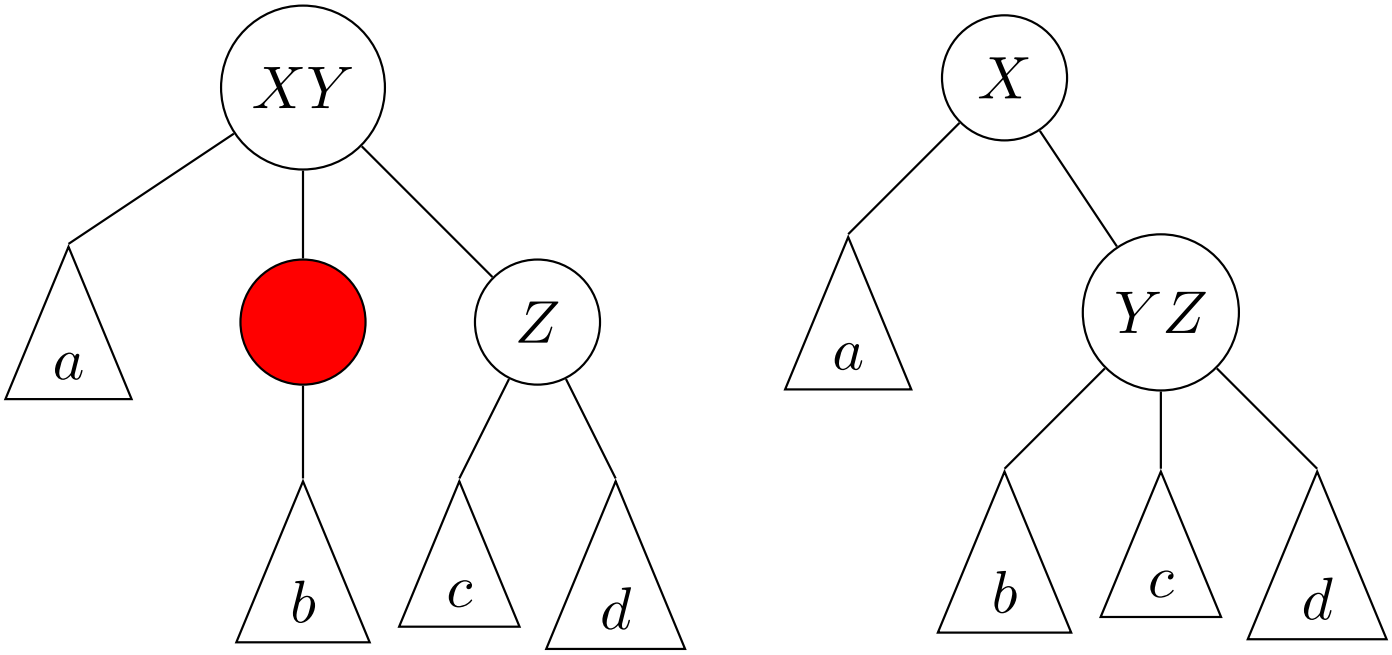
Случай 3a: у пустого места 3-родитель и 2-ребенок родителя.



a, b, c имеют высоту на 1 меньше d.

2-3 деревья: удаление

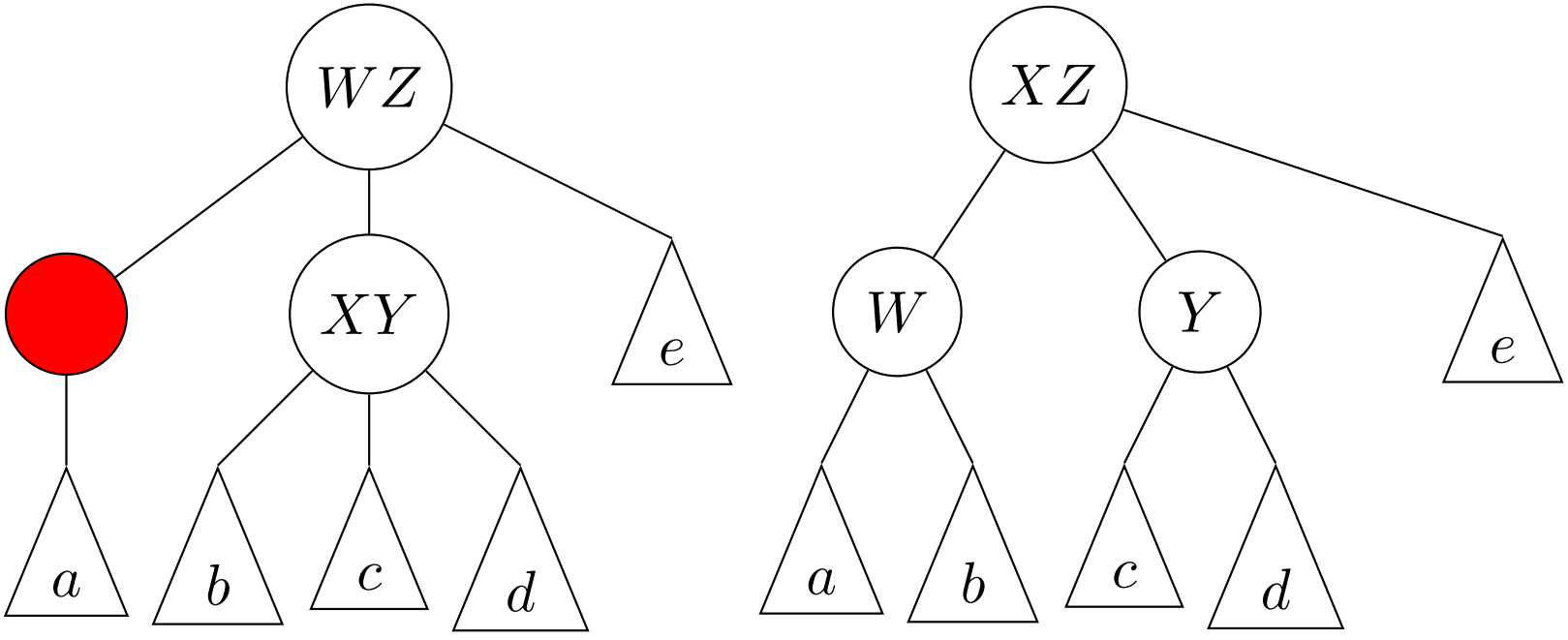
Случай 3б: у пустого места 3-родитель и 2-ребенок родителя.



b, c, d имеют высоту на 1 меньше a.

2-3 деревья: удаление

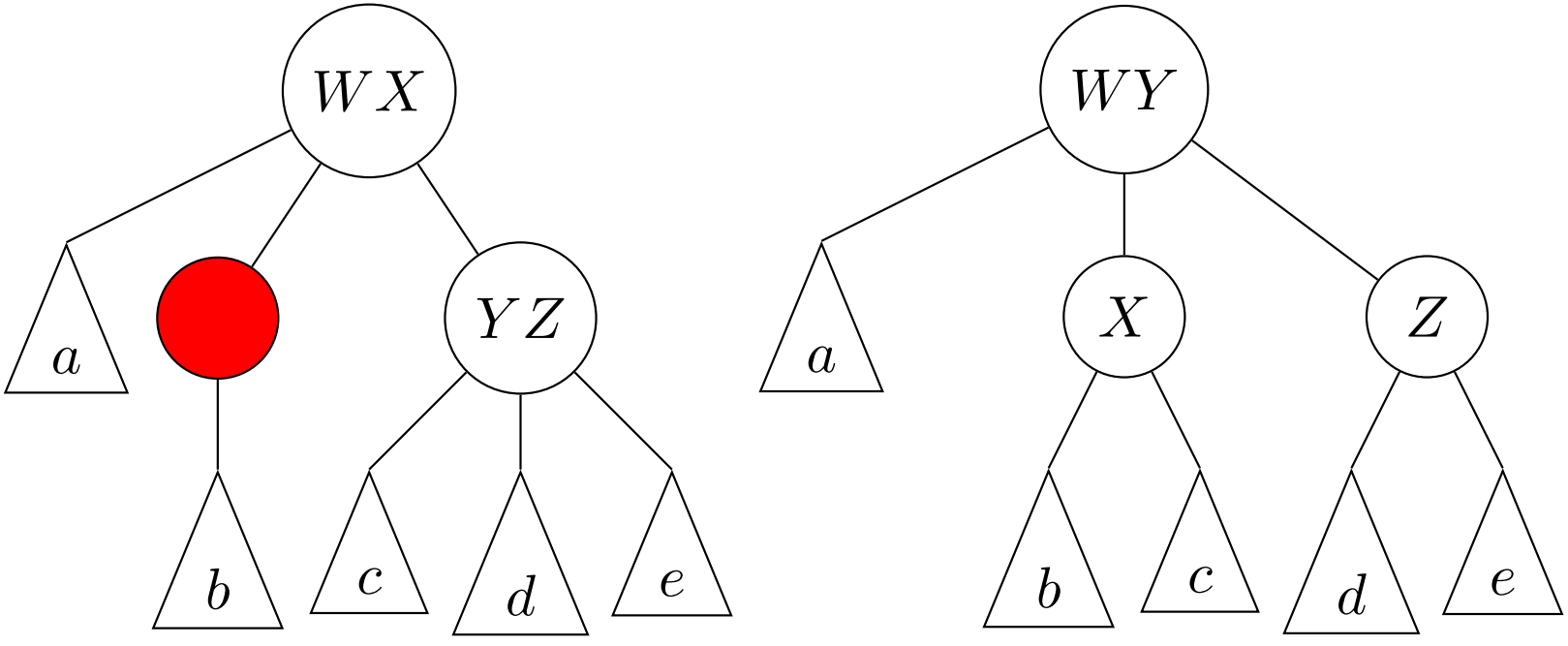
Случай 4a: у пустого места 3-родитель и 3-ребенок родителя.



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

2-3 деревья: удаление

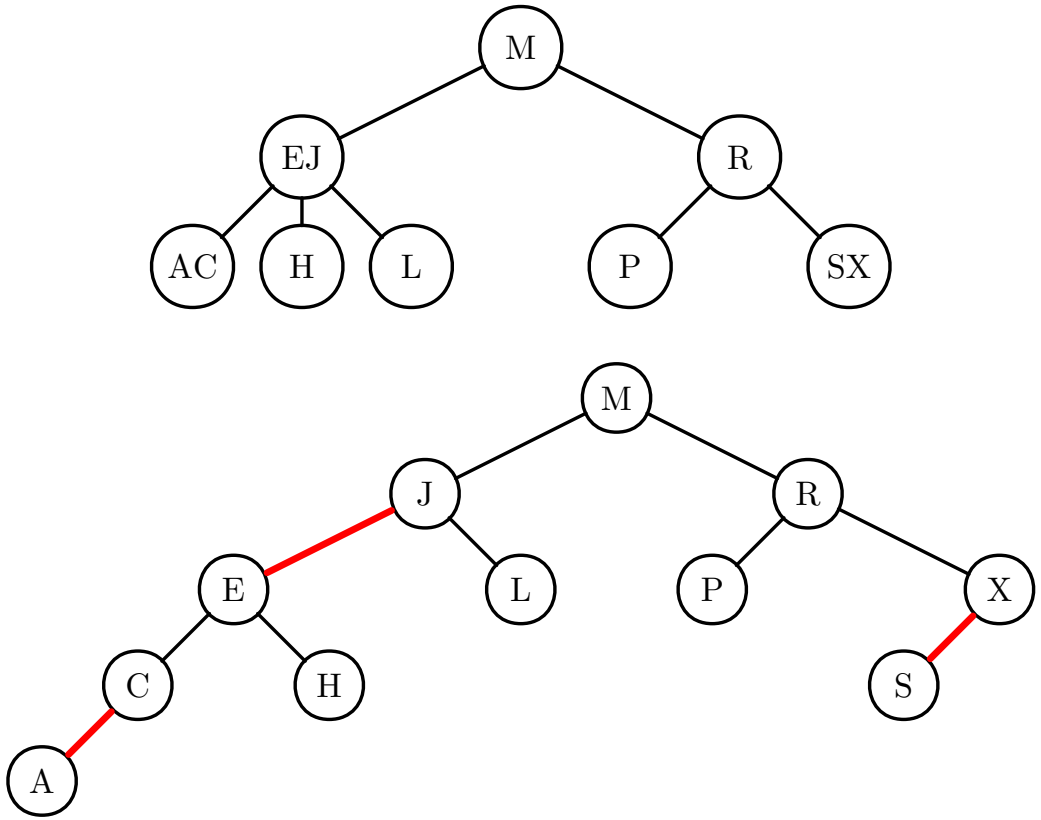
Случай 4б: у пустого места 3-родитель и 3-ребенок родителя.



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

Красно-черные деревья

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

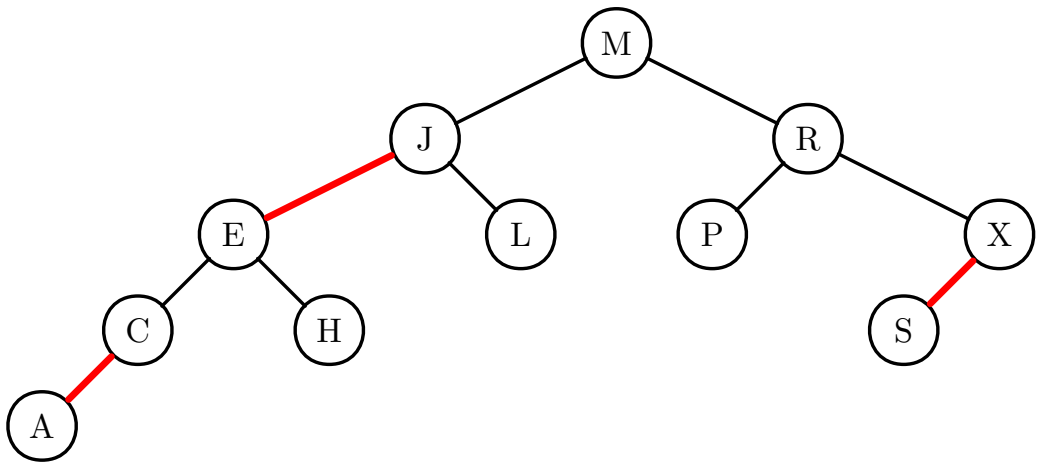


LLRB = Left-leaning red black trees

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` – ...

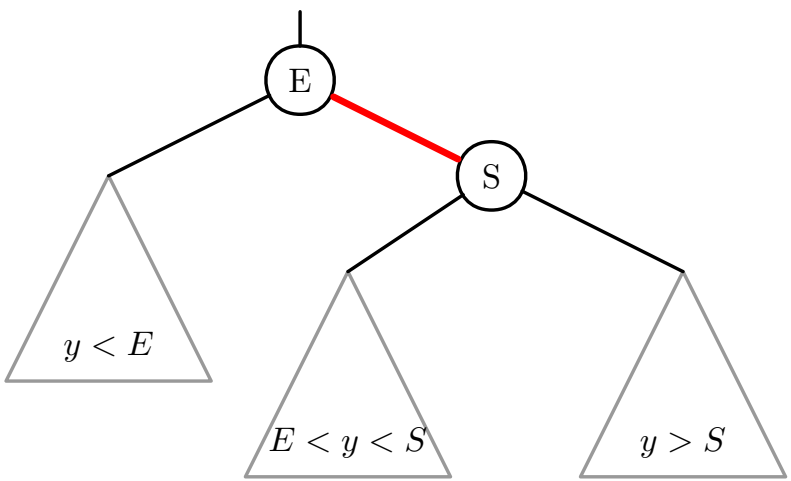


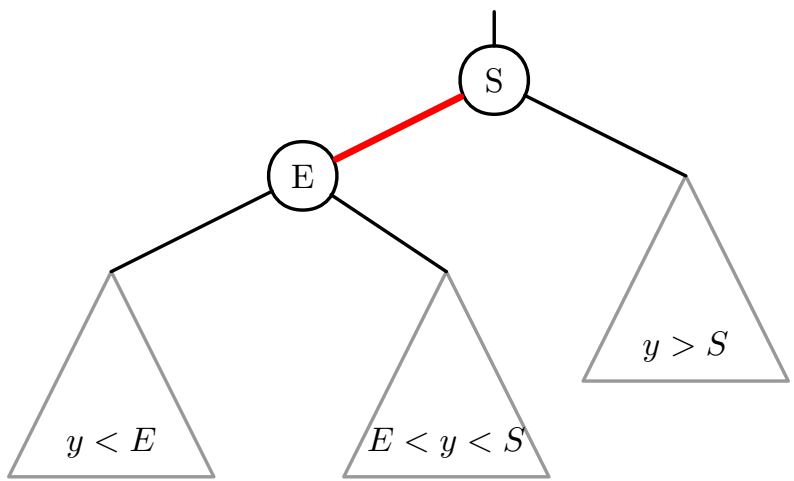
LLRB-деревья: реализация

```
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

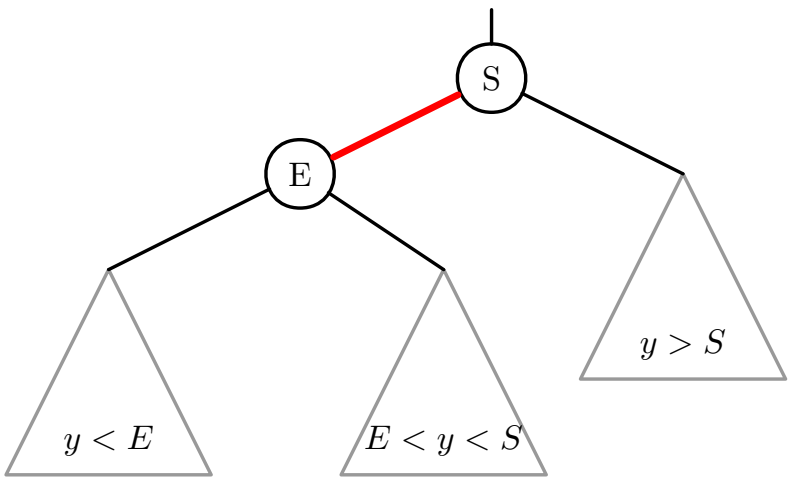
```
private Node rotateLeft(Node h)
{
    // assert isRed(h.right);
    Node x = h.right;
    h.right = x.left;
    x.left = h;
    x.color = h.color;
    h.color = RED;
    return x;
}
```

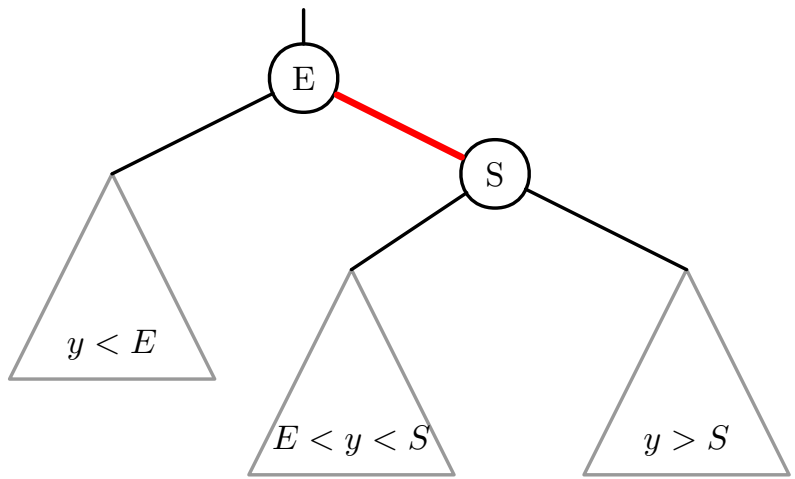




Элементарная операция-2: rotateRight

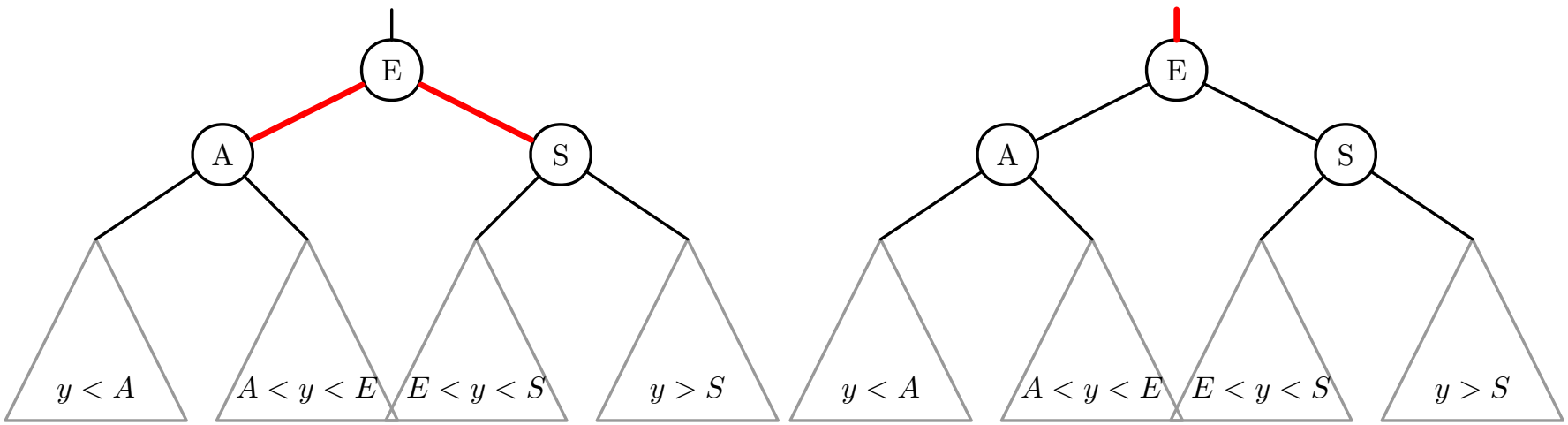
```
private Node rotateRight(Node h)
{
    // assert isRed(h.left);
    Node x = h.left;
    h.left = x.right;
    x.right = h;
    x.color = h.color;
    h.color = RED;
    return x;
}
```





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 колисчество вершин в AVL дереве высоты h. Чему равно N_h?

AVL деревья, вставка

Вставка в AVL дерево использует уже известные нам операции - поворот влево и вправо.

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)