Адрианов Н.М.
Иванов А.Б.

Алгоритмы и структуры данных

Черно-красные деревья
Геометрические приложения

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

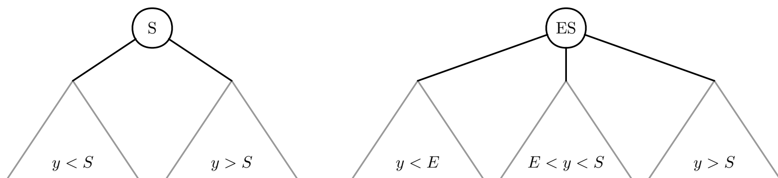
  Количество черных ребер в любом пути одинаково, количество красных ребер в пути не превосходит количество черных + 1

$\Rightarrow$ 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

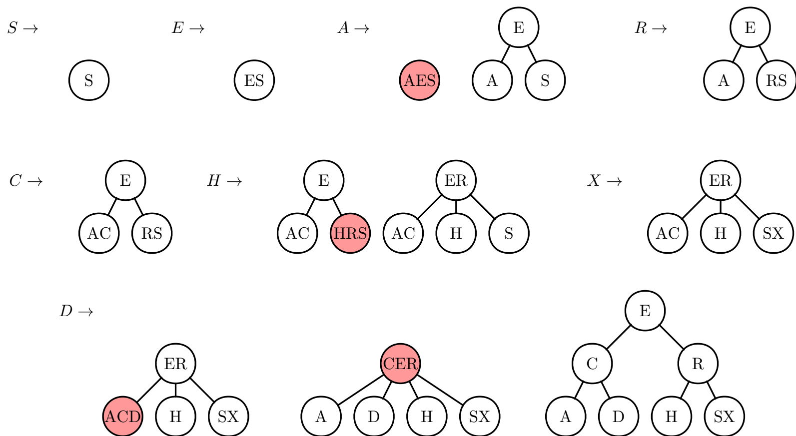
1. Выполняем поиск, находим лист, в который надо вставить ключ.
2. Если лист является 2-узлом, то превращаем его в 3-узел.
3. Если лист является 3-узлом, то превращаем в 4-узел.
4. 4-узел хранит 3 ключа – средний ключ перемещаем в родителя, левый и правый ключи превращаем в 2-узлы.
5. Если родитель был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

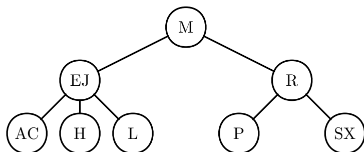
2-3 ДЕРЕВО: ПРИМЕР

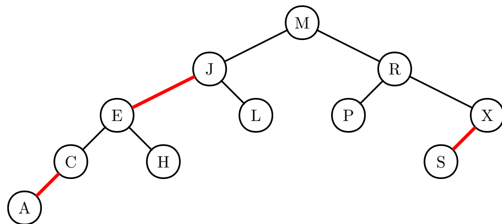
S, E, A, R, C, H, X, D



Красно-черные деревья

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



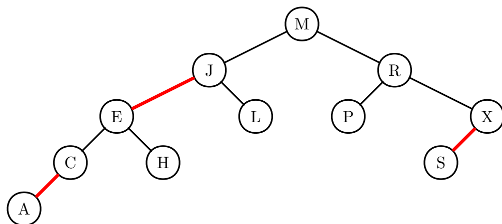


LLRB = Left-leaning red black trees

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

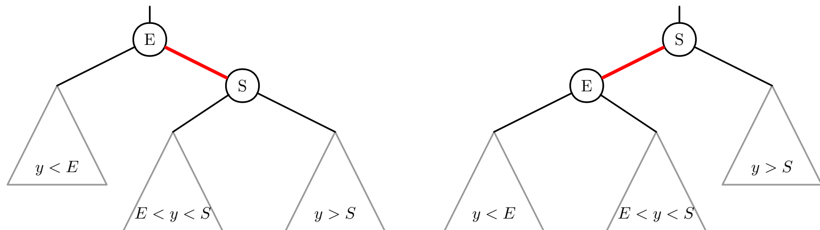
```
private static final boolean RED = true;
private static final boolean BLACK = false;

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;
}
```

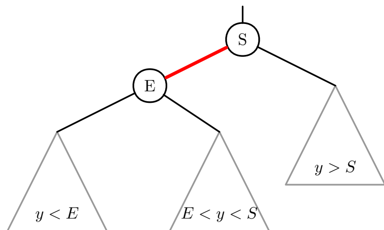
Элементарная операция-1: rotateLeft

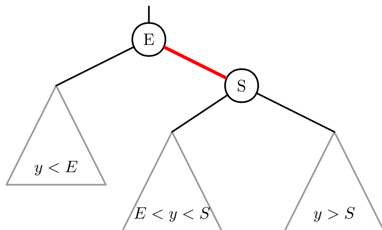
```
private Node rotateLeft(Node h)
{
    // assert isRed(h.right);
    Node x = h.right;
    h.right = x.left;
    x.left = h;
    x.color = h.color;
    h.color = RED;
    return x;
}
```



Элементарная операция-2: rotateRight

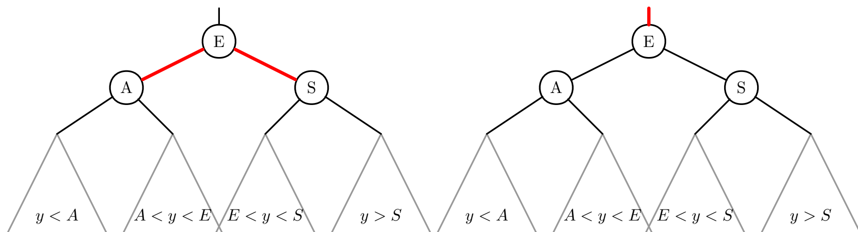
```
private Node rotateRight(Node h)
{
    // assert isRed(h.left);
    Node x = h.left;
    h.left = x.right;
    x.right = h;
    x.color = h.color;
    h.color = RED;
    return x;
}
```





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;
}
```

Стандартные библиотеки Java и .NET

Java:	Map	HashMap	TreeMap
		HashSet	TreeSet
C#:	IDictionary	Dictionary	SortedDictionary
		HashSet	SortedSet

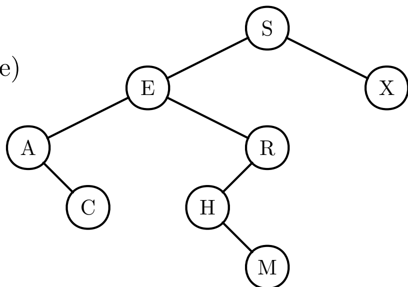
http://referencesource.microsoft.com/#System/compmod/system/collections/generic/sorteddictionary.cs

```
// A binary search tree is a red-black tree if it satisfies the
    following red-black properties:
// 1. Every node is either red or black
// 2. Every leaf (nil node) is black
// 3. If a node is red, then both its children are black
// 4. Every simple path from a node to a descendant leaf contains the
    same number of black nodes
//
// The basic idea of red-black tree is to represent 2-3-4 trees as
    standard BSTs but to add one extra bit of information
// per node to encode 3-nodes and 4-nodes.
// 4-nodes will be represented as:     B
//                                                        R           R
// 3 -node will be represented as:      B            or        B
//                                                        R          B
                 B      R
//
// For a detailed description of the algorithm, take a look at
    "Algorithms" by Robert Sedgewick.
```

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)



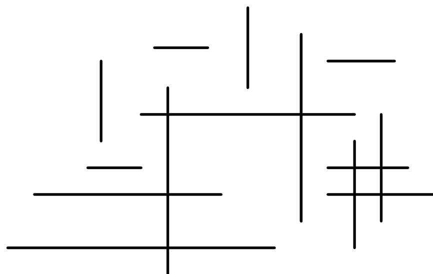
Геометрические приложения

- 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.

Поиск в 2-мерном диапазоне: сетка

Пусть все точки лежат в квадрате $[0, 1] \times [0, 1]$.
Разобьем квадрат на M^2 квадратов размера $1/M$.

Как выбрать M? Какова сложность алгоритма в среднем при равномерном распределении точек?

2D-дерево

В вершинах храним пары ключей $(a,\ b)$.

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

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

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

Пересечения прямоугольников

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

Найти: все пары пересекающихся прямоугольников.

B-деревья

(Bayer-McCreight, 1972)

- Данные читаются поблочно (постранично)
- Время чтения блока относительно велико

Применение:

- Файловые системы:
  NTFS (Windows), HFS, HFS+ (Mac),
  ReiserFS, XFS, Ext3FS, JFS (Linux)
- Базы данных:
  Oracle, MS SQL, DB2, PostgreSQL

B-деревья

B-деревья обобщают 2-3 деревья: максимально разрешается хранить до $M-1$ ключа в вершине

- Во всех вершинах кроме корневой – не менее $M/2$ ключей
- Внешние узлы хранят ключи (+данные)
- Внутренние узлы хранят ключи для обеспечения поиска

B-деревья

Для поиска или вставки в дерево с N ключами требуется количество чтений между $\log_{M-1} N$ и $\log_{M/2} N$.

На практике: не более 4 ($M = 1024 \Rightarrow N = 64 \cdot 10^9$)

Хинт: всегда держать корневую страницу загруженной в память.