Java Basic

lecture #14. BST – Binary Search Tree, tree balancing

Mentor:<….>

lecture #14. BST – Binary Search Tree

- Binary Search Tree
- AVL Tree
- Searching a key
  - implementation
- Insertion of a key
  - Implementation
- Delete a node from BST
  - Implementation
- Practice

BST – Binary Search Tree, tree balancing

- Двоичное дерево со сбалансированной высотой определяется как бинарное дерево, в котором высота левого и правого поддеревьев любого узла отличается не более чем на 1.

сбалансированными деревьями считаются AVL, красно-черные деревья

Условия для сбалансированного по высоте бинарного дерева:

1. Разница между высотами левого и правого поддеревьев для любого узла не более единицы.
2. Левое поддерево сбалансировано.
3. Правое поддерево сбалансировано.

Примечание. Пустое дерево также сбалансировано по высоте.
Баланс высоты узла = высота правого поддерева – высота левого поддерева

BST = AVL, tree balancing

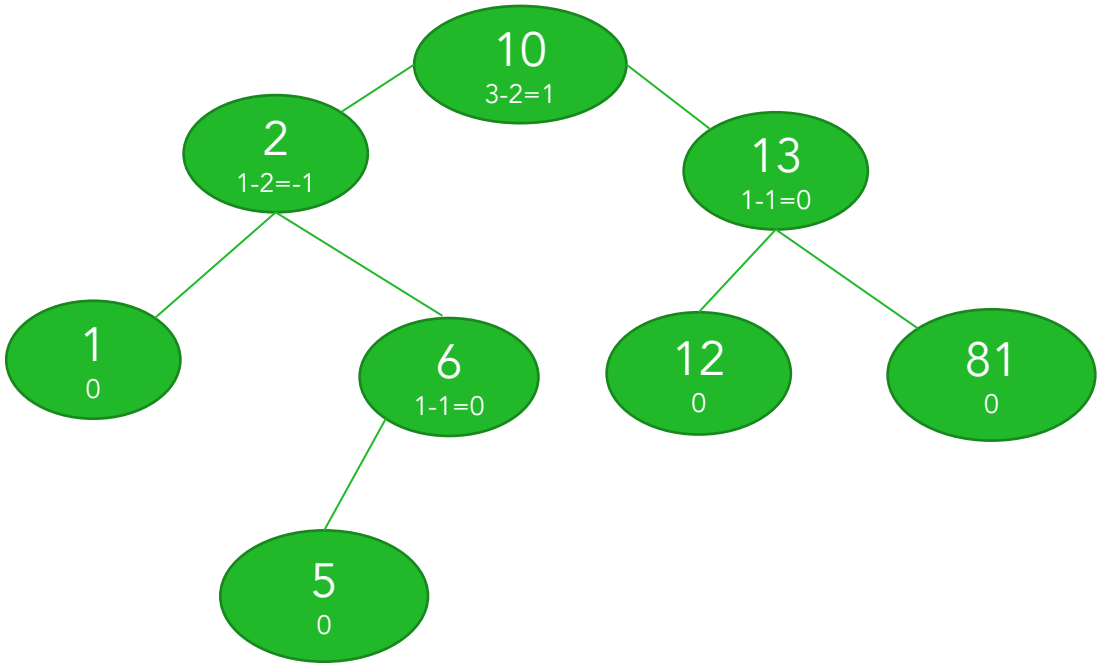
Самобалансирующееся дерево-это двоичное дерево поиска, которое балансирует высоту после вставки и удаления в соответствии с правилами балансировки:

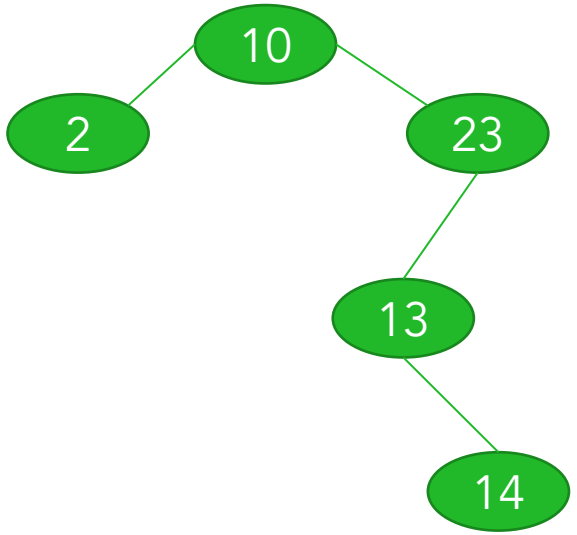
1. В дереве AVL коэффициент баланса узла может быть только одним из значений 1, 0 или -1.
2. Сохранить дерево AVL сбалансированным после любого изменения в его узлах
3. Ключ уникален в AVL дереве — нет двух узлов, имеющих один и тот же ключ

Коэффициент баланса (k) = высота (слева (k)) - высота (справа (k))

- Если коэффициент баланса любого узла равен 1, это означает, что левое поддерево на один уровень выше правого поддерева.
- Если коэффициент баланса любого узла равен 0, это означает, что левое поддерево и правое поддерево имеют одинаковую высоту.
- Если коэффициент баланса любого узла равен -1, это означает, что левое поддерево на один уровень ниже правого поддерева.

BST – Binary Search Tree, tree balancing





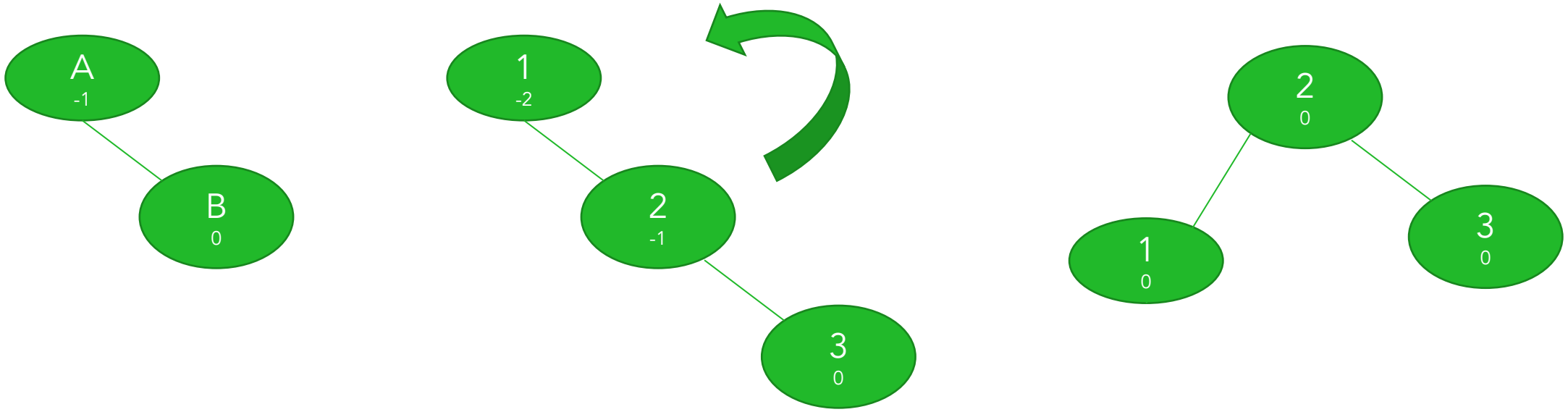
AVL – rotation

четыре типа вращений:

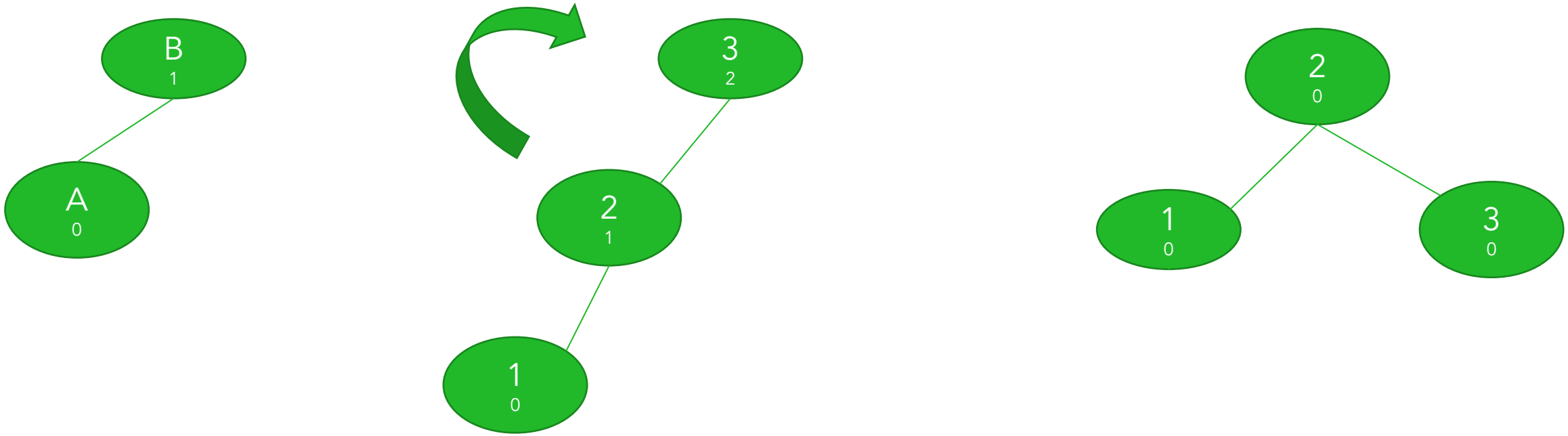
1. Вращение LL: вставленный узел находится в левом поддереве левого поддерева A
2. Вращение RR: вставленный узел находится в правом поддереве правого поддерева A
3. Вращение LR: вставленный узел находится в правом поддереве левого поддерева A
4. Вращение RL: вставленный узел находится в левом поддереве правого поддерева A

*A — **это узел, коэффициент баланса которого отличен от** -1, 0, 1*

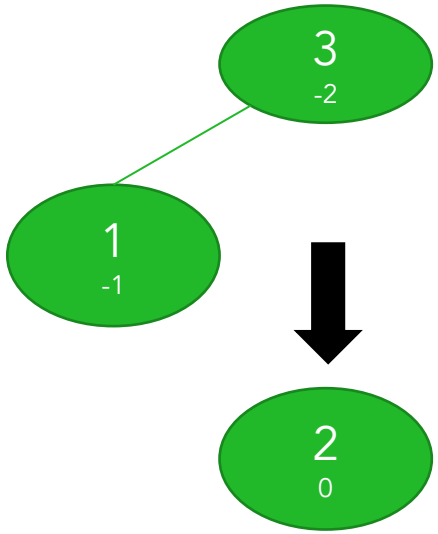
AVL – rotation -> Вращение RigthRight (RR)



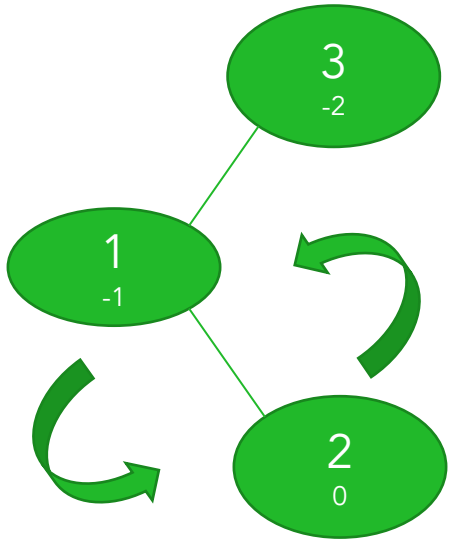
AVL – rotation -> Вращение LeftLeft (LL)



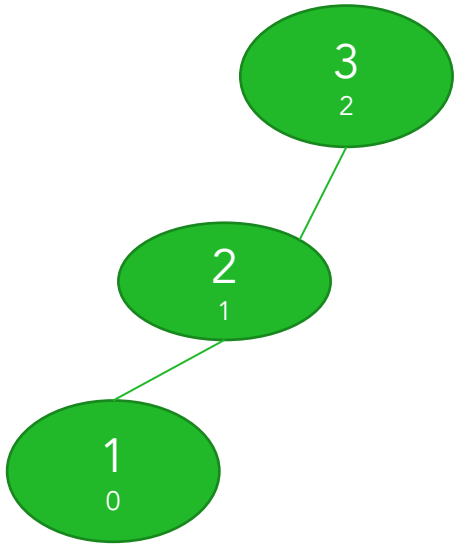
AVL – rotation -> Вращение LeftRight (LR)



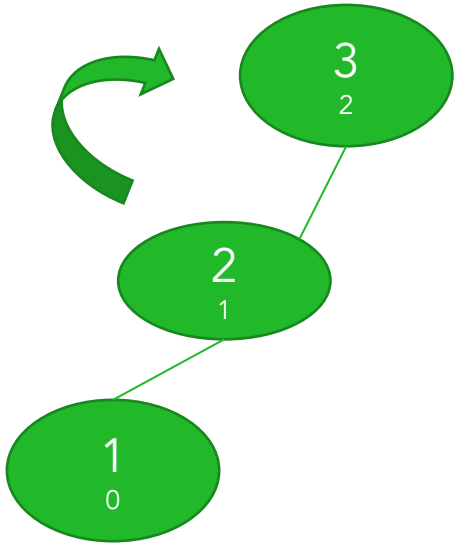
Шаг 1
Узел 2 был вставлен в правое поддерево 1 и левое поддерево 3.



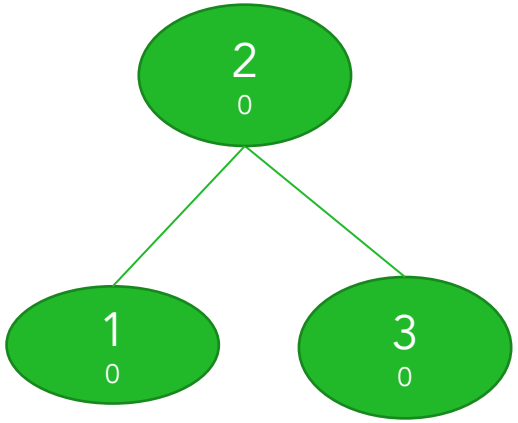
Шаг 2 RR
Выполнив вращение RR, узел A **<u>станет</u>** левым поддеревом B.



Шаг 3
Узел C все еще не сбалансирован.

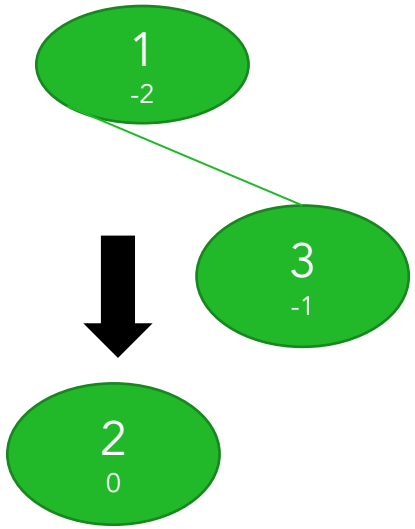


Шаг 4 LL
Узел C **<u>станет</u>** правым поддеревом узла B, A — левым поддеревом узла B.

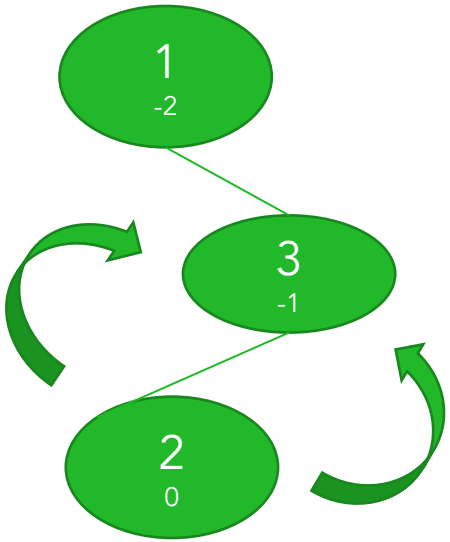


Шаг 5
BST сбалансирован

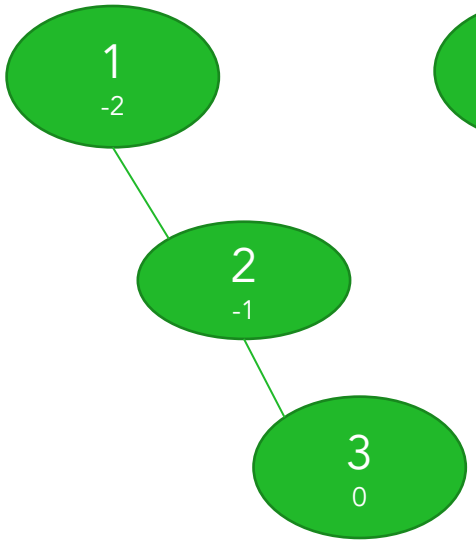
AVL – rotation -> Вращение RightLeft (RL)



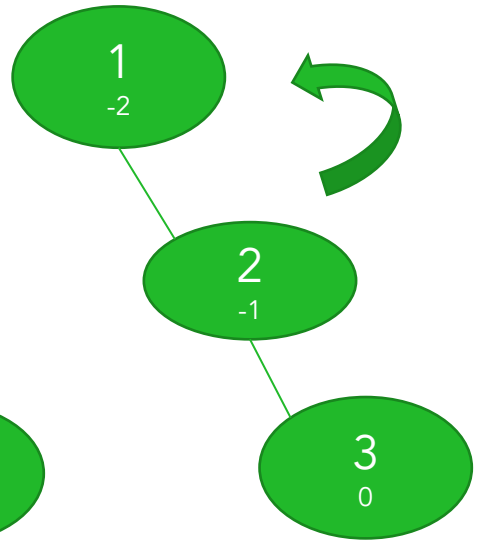
Шаг 1
Узел B был вставлен в левое поддерево C и правое поддерево A



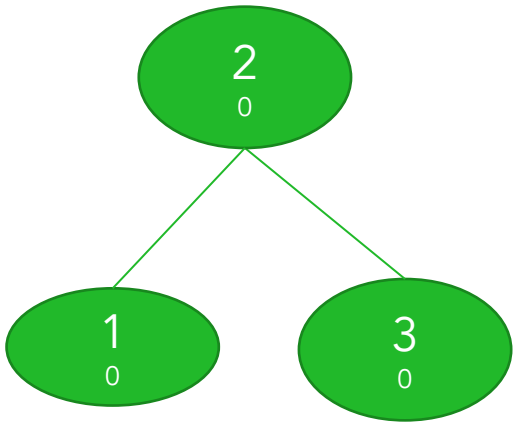
Шаг 2 LL
Выполнив вращение RR, узел A **<u>станет</u>** левым поддеревом B.



Шаг 3
Узел A все еще не сбалансирован.



Шаг 4 RR
Теперь узел C **<u>станет</u>** правым поддеревом узла B, а узел A станет левым поддеревом узла B.

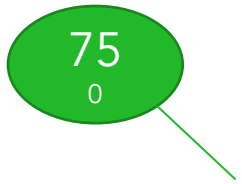


Шаг 5
BST сбалансирован

Постройте дерево AVL, вставив следующие элементы.

75, 9, 18, 29, 17, 100, 88, 81

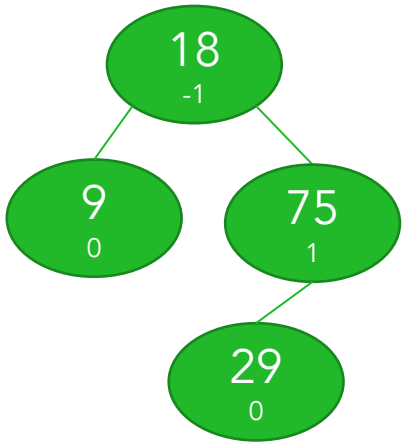
Шаг 1 Шаг 2 Шаг 3 Шаг 4



Постройте дерево AVL, вставив следующие элементы.

~~75~~, ~~9~~, ~~18~~, 29, 17, 100, 88, 81

Шаг 5



Шаг 6

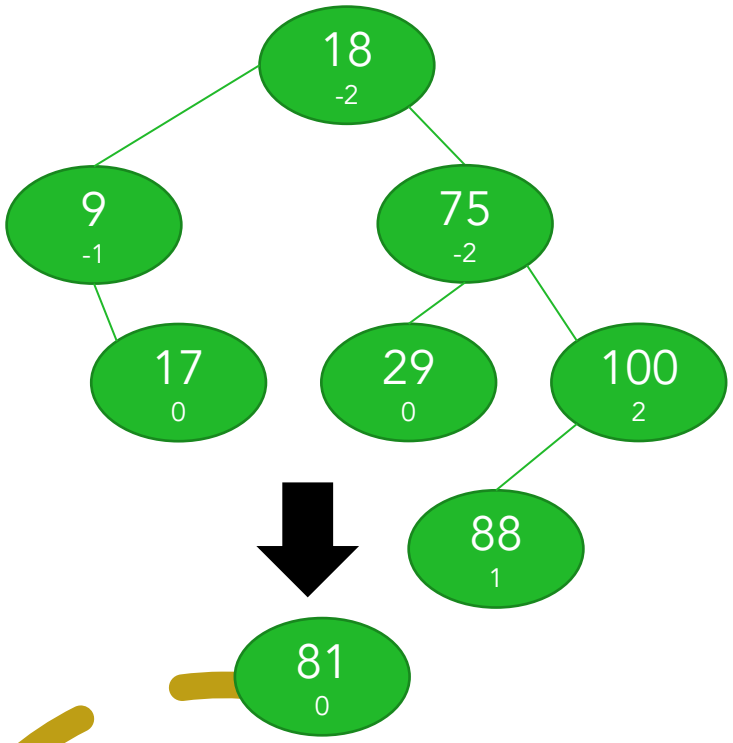
Шаг 7

Шаг 8

Постройте дерево AVL, вставив следующие элементы.

~~75~~, ~~9~~, ~~18~~, ~~29~~, ~~17~~, ~~100~~, ~~88~~, 81

Шаг 9

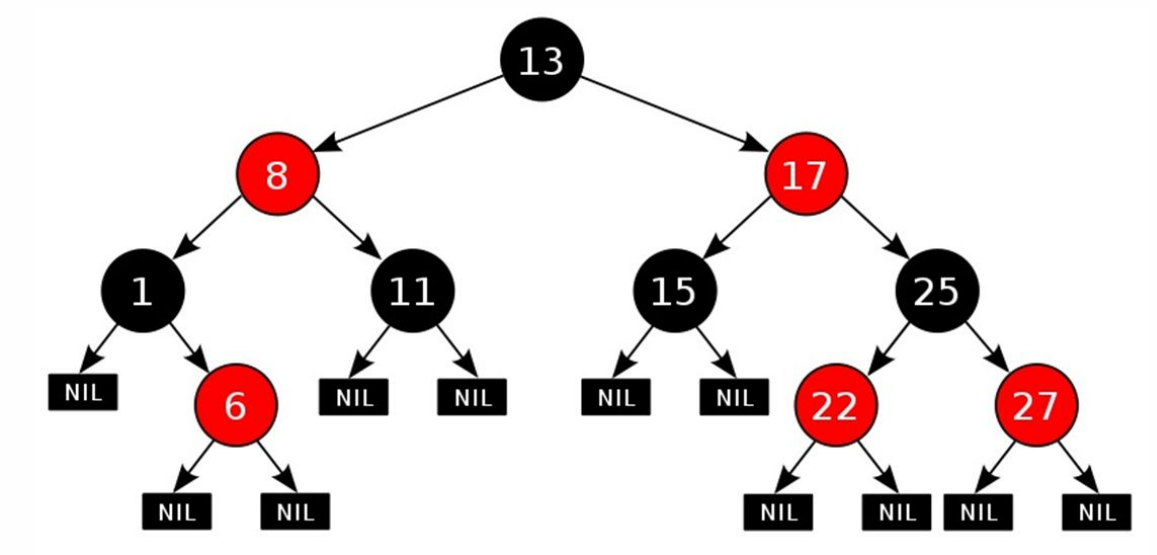


Шаг 10

This is Red-Black tree

Правила, которым следует каждое красно-черное дерево:

- Каждый узел имеет красный или черный цвет.
- Корень дерева всегда черный.
- Нет двух соседних красных узлов (красный узел не может иметь красного родителя или красного дочернего элемента).
- Каждый путь от узла (включая корень) к любому из его потомков NULL узлов имеет одинаковое количество черных узлов.
- Все листовые узлы являются черными узлами.



resource

AVL симулятор
https://www.cs.usfca.edu/~galles/visualization/AVLtree.html

SOLID

S

Принцип единственной ответственности (single responsibility principle)

Для каждого класса должно быть определено единственное назначение. Все ресурсы, необходимые для его осуществления, должны быть инкапсулированы в этот класс и подчинены только этой задаче.

O

Принцип открытости/закрытости (open-closed principle)

«программные сущности … должны быть открыты для расширения, но закрыты для модификации».

L

Принцип подстановки Лисков (Liskov substitution principle)

«функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа не зная об этом». См. также контрактное программирование.

I

Принцип разделения интерфейса (interface segregation principle)

«много интерфейсов, специально предназначенных для клиентов, лучше, чем один интерфейс общего назначения»

D

Принцип инверсии зависимостей (dependency inversion principle)

«Зависимость на Абстракциях. Нет зависимости на что-то конкретное»