



БЕЛОРУССКИЙ  
ГОСУДАРСТВЕННЫЙ  
УНИВЕРСИТЕТ

# ОРГАНИЗАЦИЯ ПОИСКА

СБАЛАНСИРОВАННЫЕ ПОИСКОВЫЕ ДЕРЕВЬЯ

**АВЛ-дерево**

## Словарные операции

- поиск элемента с заданным ключом  $x$
- добавление нового элемента с заданным ключом  $x$
- удаление элемента с заданным ключом  $x$

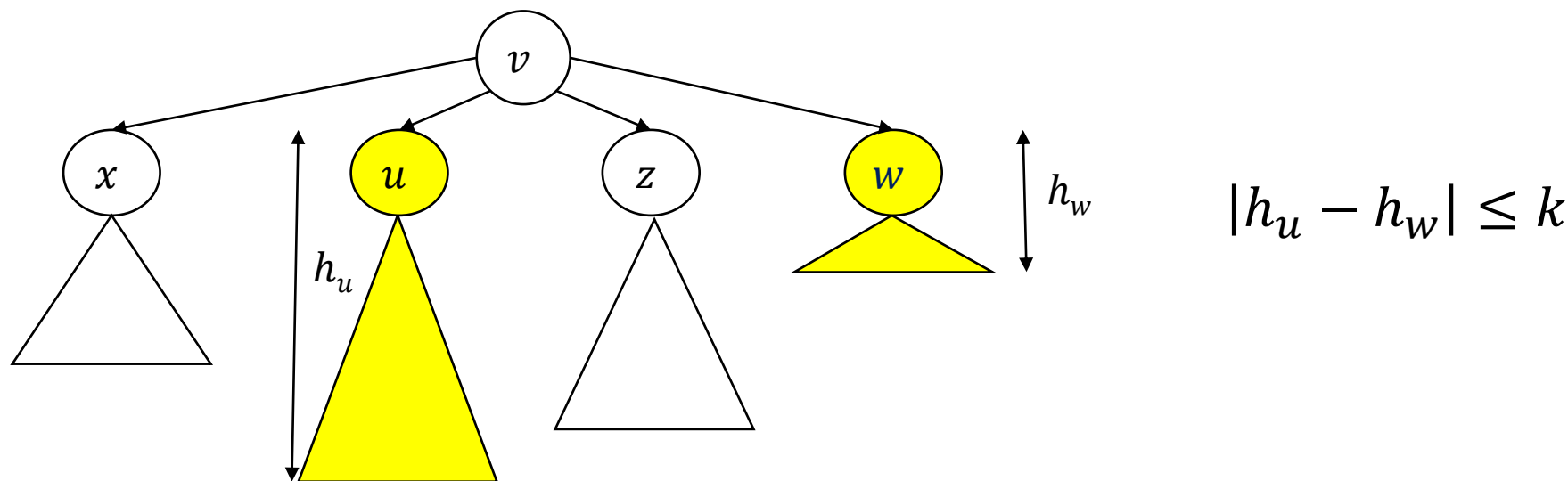
# Сбалансированные деревья

# Определение

Корневое дерево называется ***k*-сбалансированным по высоте**, если для каждой вершины  $v$  выполняется следующее свойство:

высоты её максимального (по высоте) и минимального (по высоте) поддеревьев отличаются не более, чем на  $k$ .

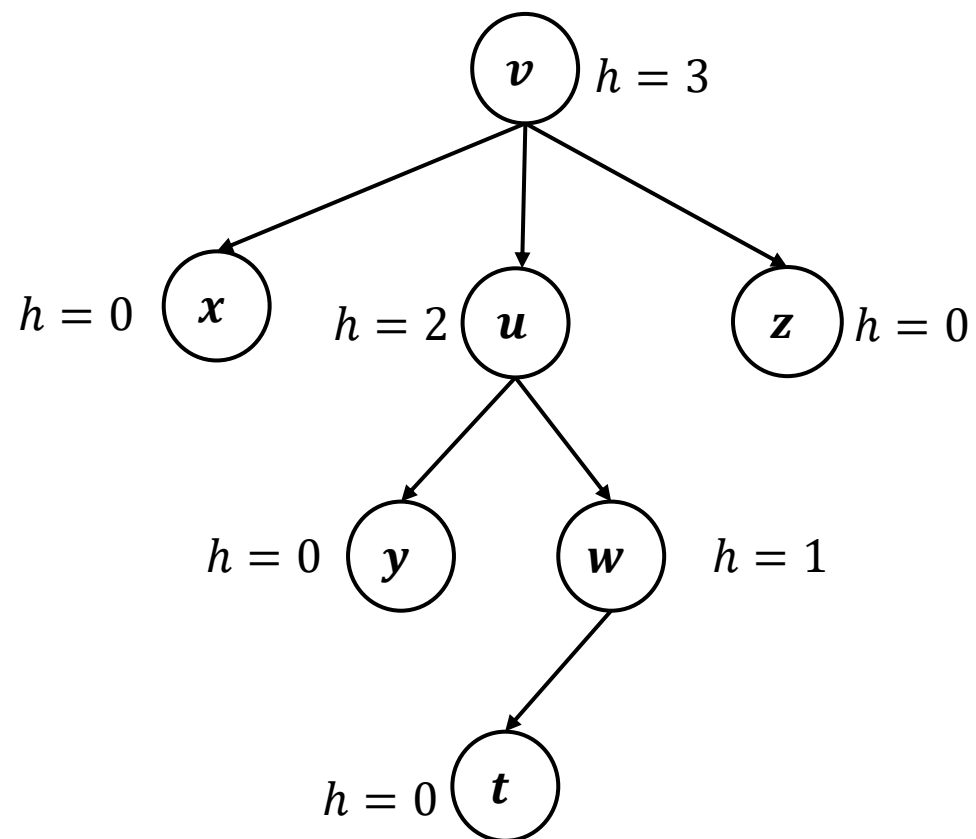
Если  $k = 1$ , то просто говорят, что дерево **сбалансировано**.



## Замечание

Если у вершины  $v$  только одно поддерево, то считаем, что второе поддерево имеет высоту минус 1.

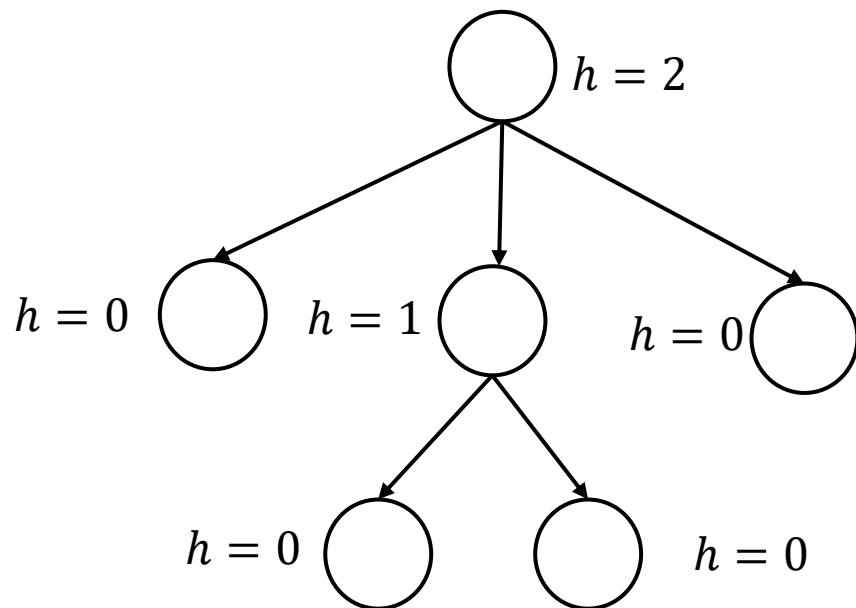
## Пример



$$k = 2$$

дерево 2-сбалансировано по высоте

## Пример

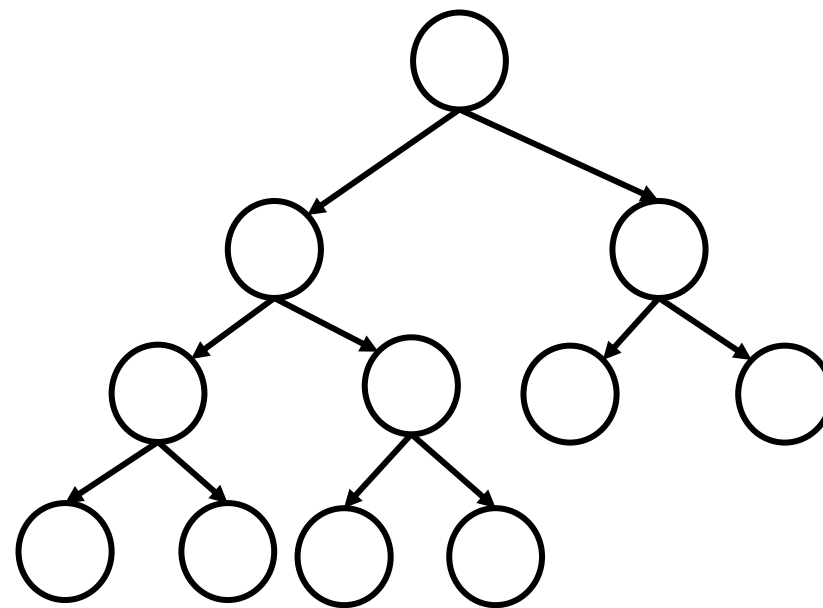


$$k = 1$$

дерево сбалансировано

## Полное бинарное дерево —

это такое корневое дерево, в котором каждая вершина имеет не более двух сыновей, а заполнение вершин осуществляется в порядке от верхних уровней к нижним, причём на одном уровне заполнение вершинами производится слева направо. Пока уровень полностью не заполнен, к следующему уровню не переходят. Последний уровень в полном бинарном дереве может быть заполнен не полностью.



$$k = 1$$

полное бинарное дерево всегда  
сбалансировано

Высота полного бинарного дерева  $h = O(\log n)$ , где  $n$  – количество вершин.

# Идеально сбалансированные деревья

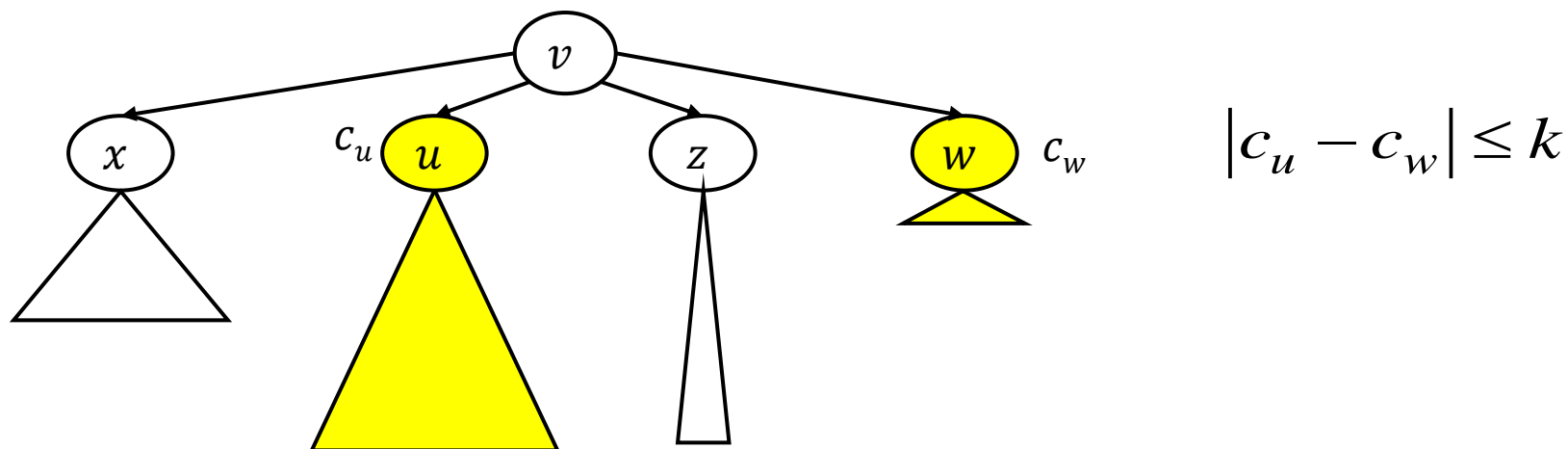


# Определение

Корневое дерево называется ***k-идеально сбалансированным по количеству вершин***, если для каждой её вершины  $v$

количество вершин в её максимальном (по количеству вершин) поддереве отличается от количества вершин в её минимальном (по количеству вершин) поддереве **не более, чем на  $k$** .

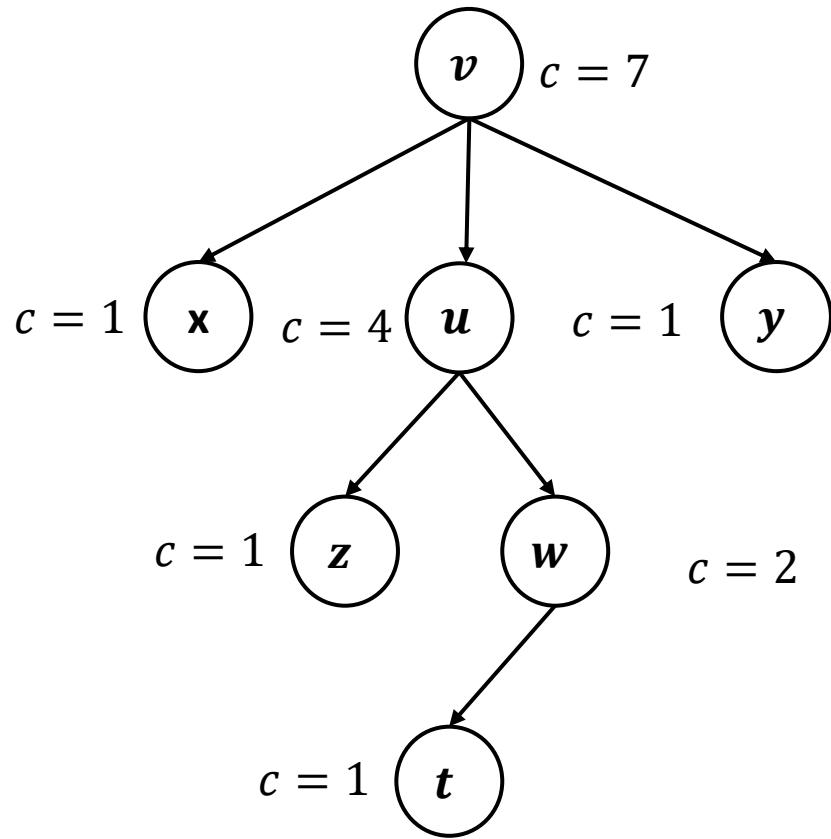
Если  $k = 1$ , то говорят, что дерево ***идеально сбалансировано***.



## Замечание

Если у вершины  $v$  только одно поддерево, то считаем, что не существующее второе поддерево имеет 0 вершин.

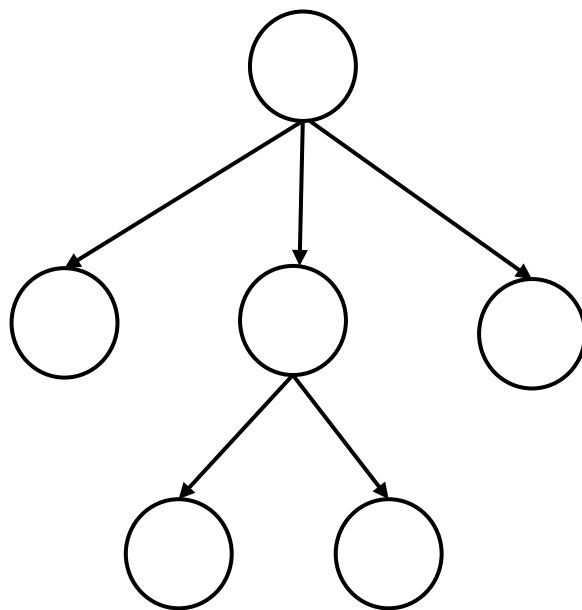
Пример



$$k = 3$$

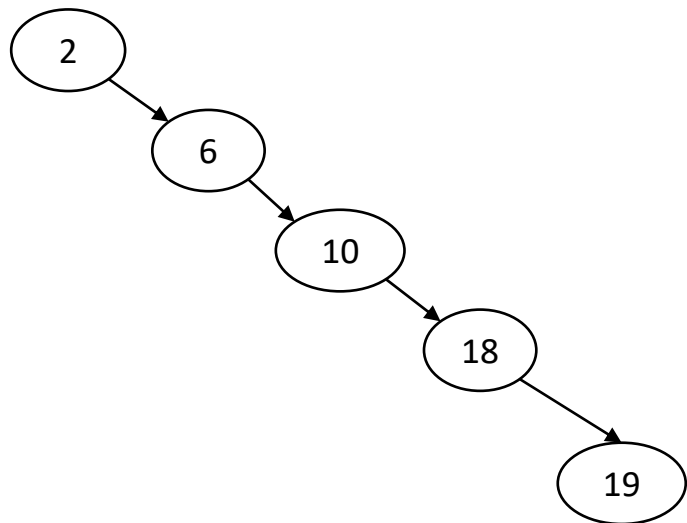
3-идеально сбалансировано  
по количеству вершин

Каждое идеально-сбалансированное дерево является сбалансированным. Обратное верно не всегда.



Дерево на рисунке – сбалансировано, но не является идеально-сбалансированным.

## Оценки для бинарных поисковых деревьев



в худшем случае высота дерева

$$h = n - 1$$

поиск элемента с  
заданным ключом  $x$

$$O(h)$$

добавление элемента с  
заданным ключом  $x$

$$O(h)$$

построение дерева для  
последовательности из  
 $n$  элементов

$$O(n \cdot h)$$

удаление элемента с  
заданным ключом  $x$

$$O(h)$$

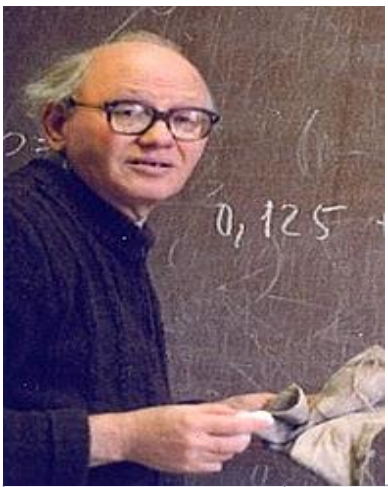
обход дерева из  $n$   
вершин

$$O(n)$$

В **1962** году советские учёные

Г. М. **А**дельсон-**В**ельский

Е. М. **Л**андис



Георгий  
Максимович  
Адельсон-  
Вельский

Дата рождения	8 января 1922
Место рождения	<a href="#">Самара</a> , <a href="#">РСФСР</a>
Дата смерти	26 апреля 2014 (92 года)
Место смерти	<a href="#">Гиватаим</a> , <a href="#">Израиль</a>
Страна	<a href="#">СССР</a> → <a href="#">Израиль</a>
Научная сфера	<a href="#">математик</a>
Место работы	<a href="#">Институт теоретической и экспериментальной физики</a>
<a href="#">Альма-матер</a>	<a href="#">МГУ (мехмат)</a>
Учёная степень	<a href="#">кандидат ф.-м. наук</a>



Евгений  
Михайлович  
Ландис

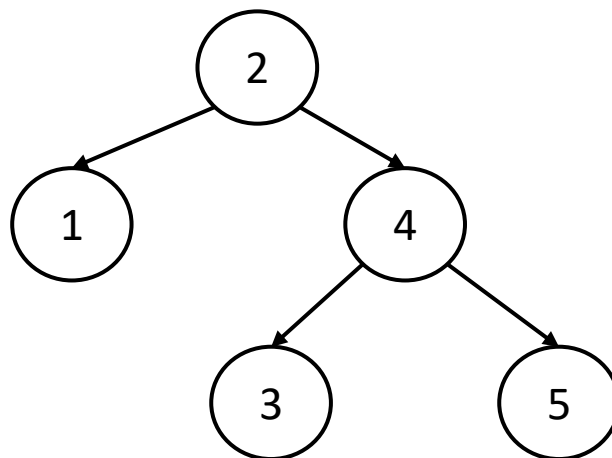
Дата рождения	6 октября 1921
Место рождения	Харьков
Дата смерти	12 декабря 1997 (76 лет)
Место смерти	<a href="#">Москва</a> , <a href="#">Россия</a>
Страна	<a href="#">СССР</a> → <a href="#">Россия</a>
Научная сфера	математика
Место работы	<a href="#">Московский государственный университет</a>
<a href="#">Альма-матер</a>	<a href="#">МГУ (мехмат)</a>
Учёная степень, звание	<a href="#">доктор ф.-м. наук</a> , профессор

предложили структуру данных  
сбалансированного поискового дерева:  
**АВЛ**-дерево

**АВЛ** — аббревиатура, образованная первыми буквами фамилий создателей

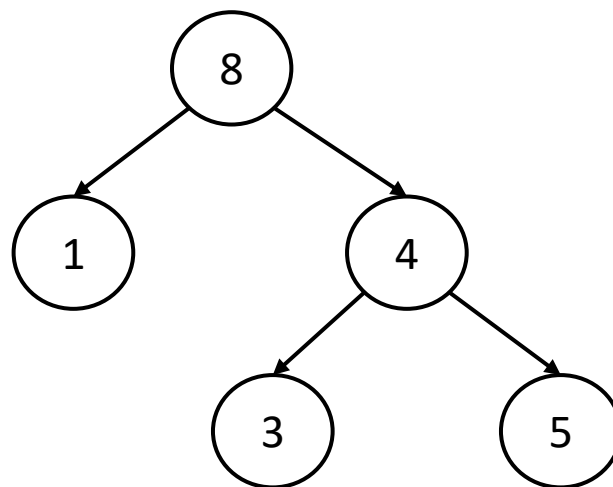
## **АВЛ-дерево –**

это бинарное поисковое дерево, которое является сбалансированным по высоте.



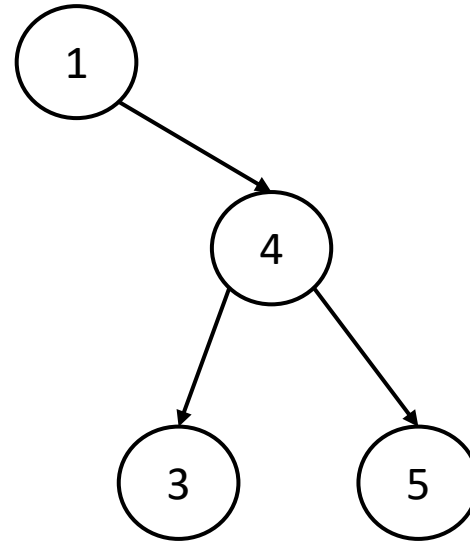


# АВЛ-дерево ?



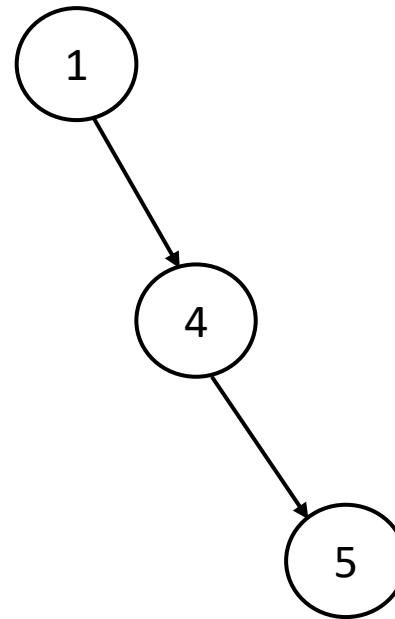
**НЕТ**

# АВЛ-дерево ?



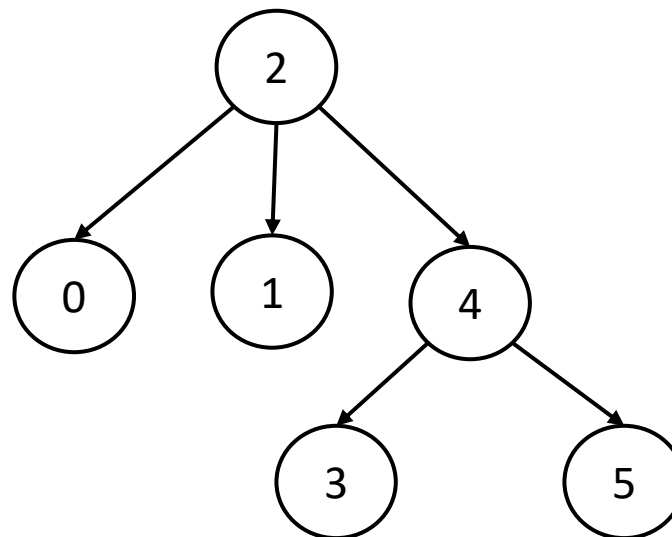
**НЕТ**

# АВЛ-дерево ?



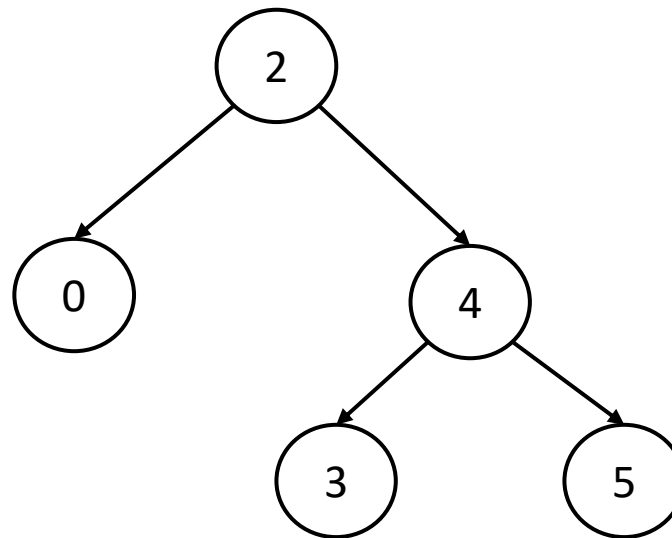
**НЕТ**

# АВЛ-дерево ?



**НЕТ**

# АВЛ-дерево ?



**ДА**

## ТЕОРЕМА

Пусть  $n$  – число внутренних вершин АВЛ-дерева,  
 $h$  – его высота.

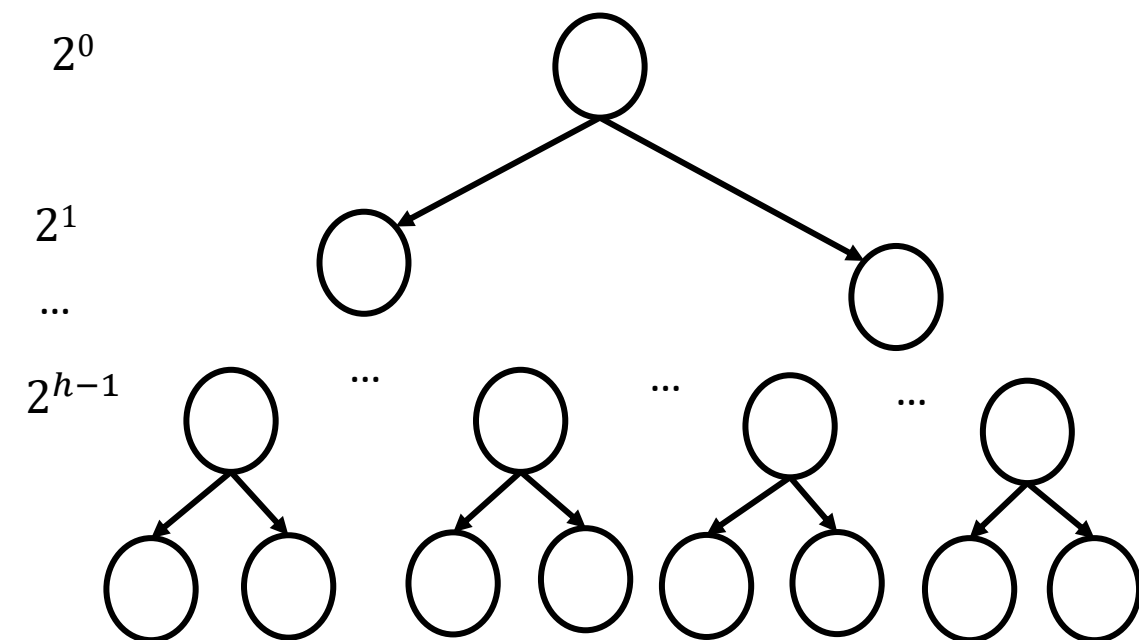
Тогда справедливы следующие неравенства:

$$\log_2(n + 1) \leq h < 1,4404 \cdot \log_2(n + 2) - 0,328$$

Для доказательства утверждения оценивают **максимальное и минимальное число внутренних вершин**.

**Максимальное число внутренних вершин** оценивается достаточно просто:

так как AVL-дерево является бинарным деревом, то подсчитаем максимальное число внутренних вершин у полного бинарного дерева высоты  $h$ .



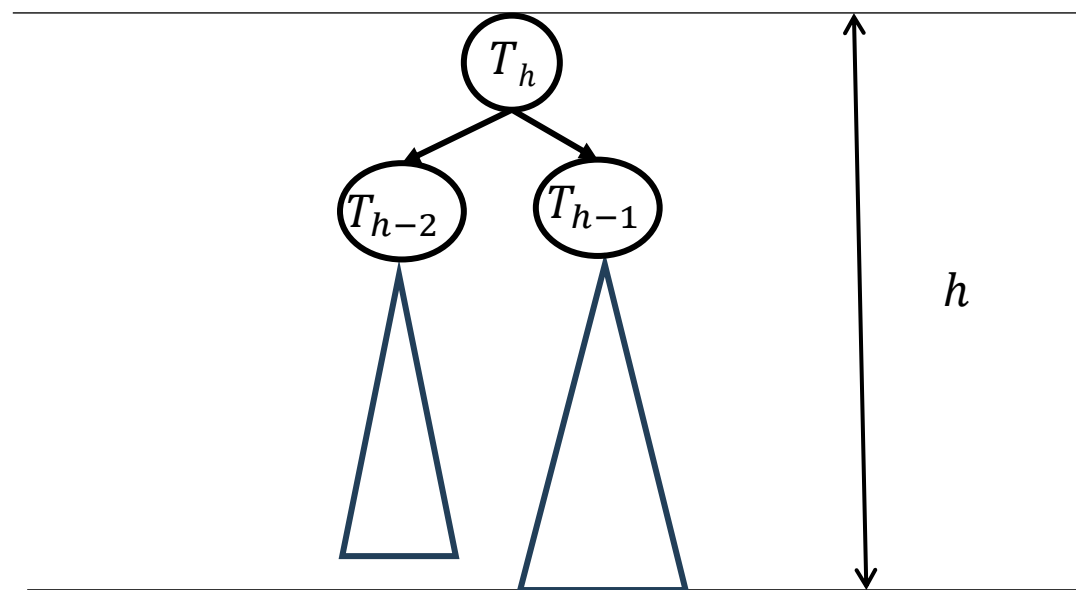
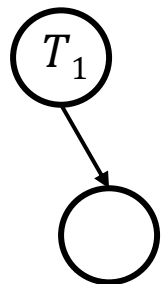
$$n \leq 2^0 + 2^1 + \dots + 2^{h-1} = 2^h - 1$$

$$h \geq \log_2(n + 1).$$

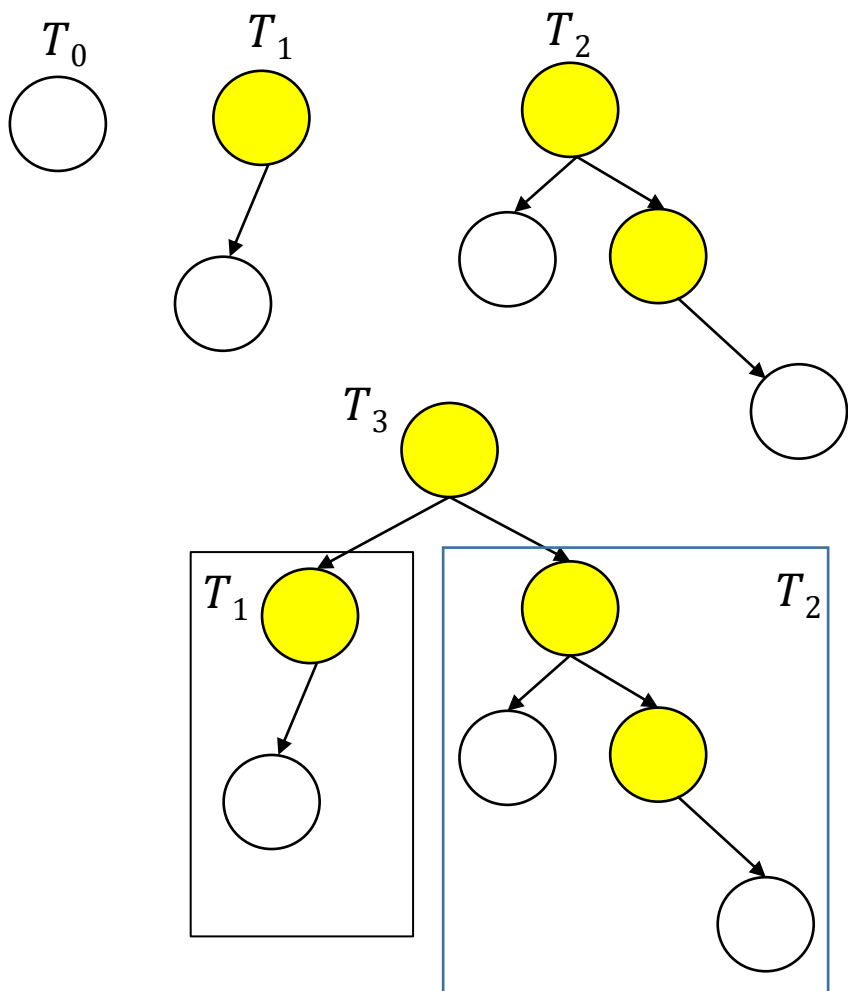
$\log_2(n + 1) \leq h$

 $< 1,4404 \cdot \log_2(n + 2) - 0,328$

Для оценки минимального числа внутренних вершин рассмотрим AVL-деревья высоты  $h$  с минимальным числом внутренних вершин.







Внутренние вершины на рисунке имеют желтую заливку).

Поскольку принцип построения деревьев напоминает построение чисел Фибоначчи, то такие деревья обычно называют **деревьями Фибоначчи**.

Леонардо Пизанский

итал. *Leonardo Pisano*



1170-1250

первый крупный математик средневековой Европы

подписывался Боначчи или  
Леонардо Биголло (странник, бездельник)

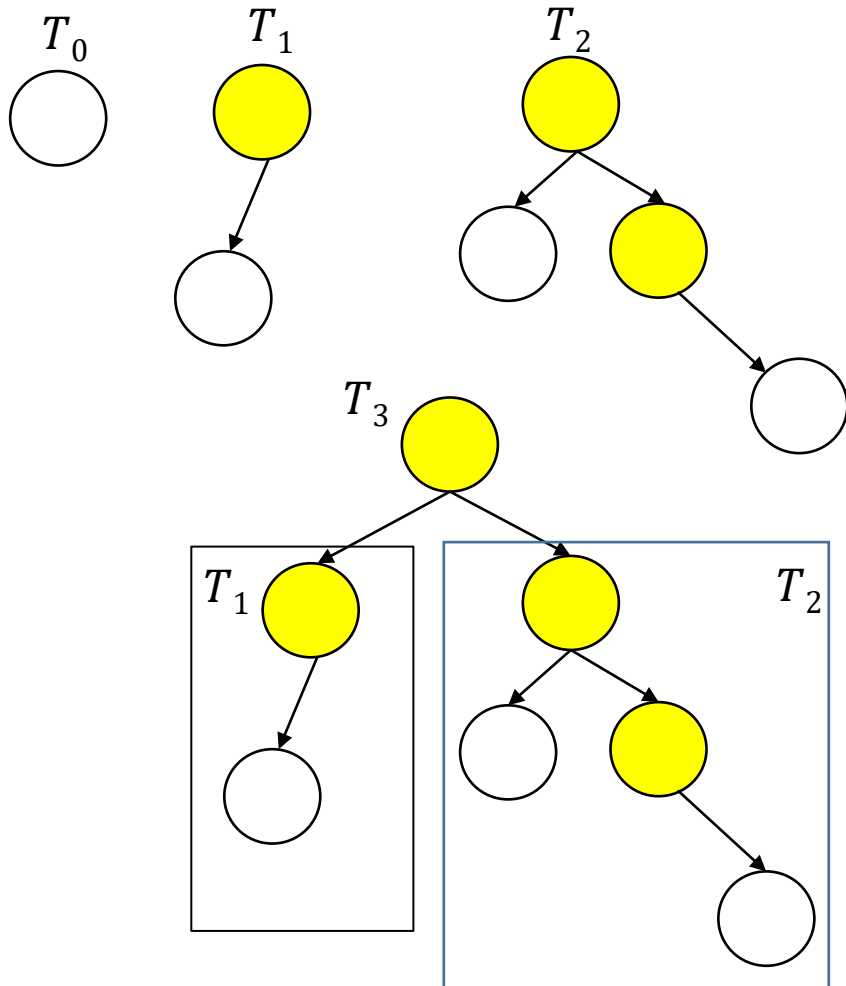
известен под прозвищем **Фибоначчи** (*Lionardo Fibonacci*)

? сын Боначчи («*filius Bonacci*»)

? прозвище «боначчи» — удачливый);

Для оценки **минимального числа внутренних вершин** используются свойства чисел Фибоначчи.

Пусть  $N_h$  — число внутренних вершин AVL-дерева высоты  $h$  с минимальным числом внутренних вершин (внутренние вершины на рисунке имеют желтую заливку).



$$N_{h+1} = N_h + N_{h-1} + 1$$

$$(N_{h+1} + 1) = (N_h + 1) + (N_{h-1} + 1)$$

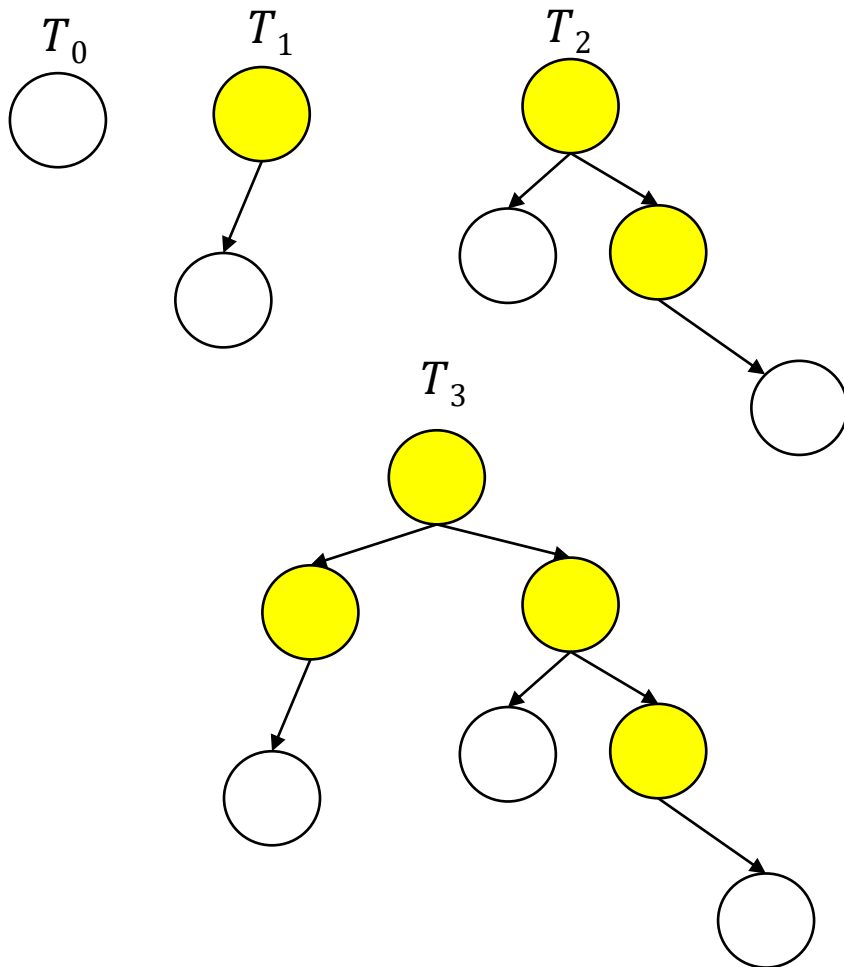
выполним замену переменной:

$$F'_i = N_i + 1$$



$$F'_{h+1} = F'_h + F'_{h-1}$$

Какая связь между  $F'_i$  и  $F_i$  — числом Фибоначчи?



$h$	$N_h$	$F'_h$	$F_h$
0	0	1	
1	1	2	1
2	2	3	1
3	4	5	2
4	7	8	3
5	12	13	5
6	...	...	8
7			13

$$F_{h+2} = F'_h = N_h + 1$$

$$N_h = F_{h+2} - 1$$

$$F_h = \frac{\Phi^h - \hat{\Phi}^h}{\sqrt{5}}, \Phi = \frac{1 + \sqrt{5}}{2}, \hat{\Phi} = \frac{1 - \sqrt{5}}{2}$$

$$n \geq N_h = \frac{\Phi^{h+2} - \hat{\Phi}^{h+2}}{\sqrt{5}} - 1 \geq \frac{\left(\frac{1 + \sqrt{5}}{2}\right)^{h+2}}{\sqrt{5}} - 2$$

$$\log_2(n + 1) \leq h < 1,4404 \cdot \log_2(n + 2) - 0,328$$

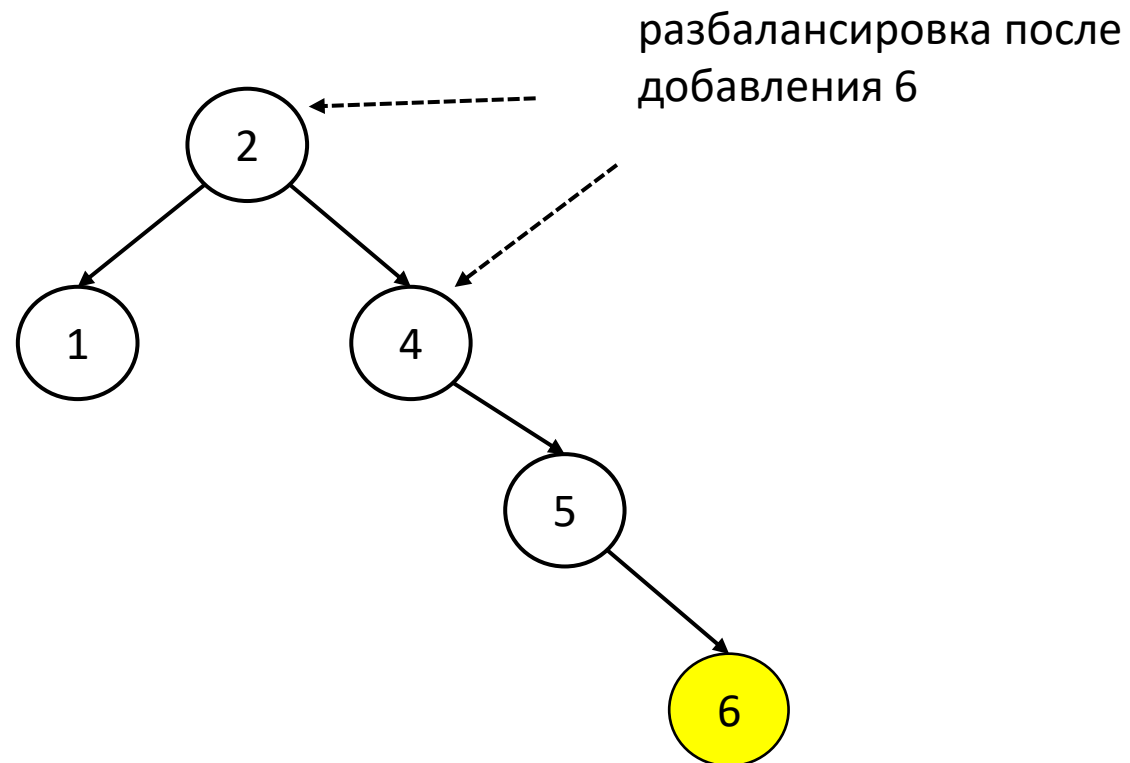
Теорема доказана.

Операции поиска, добавления и удаления элементов для AVL-деревьев осуществляются точно также, как и для бинарных поисковых деревьев.

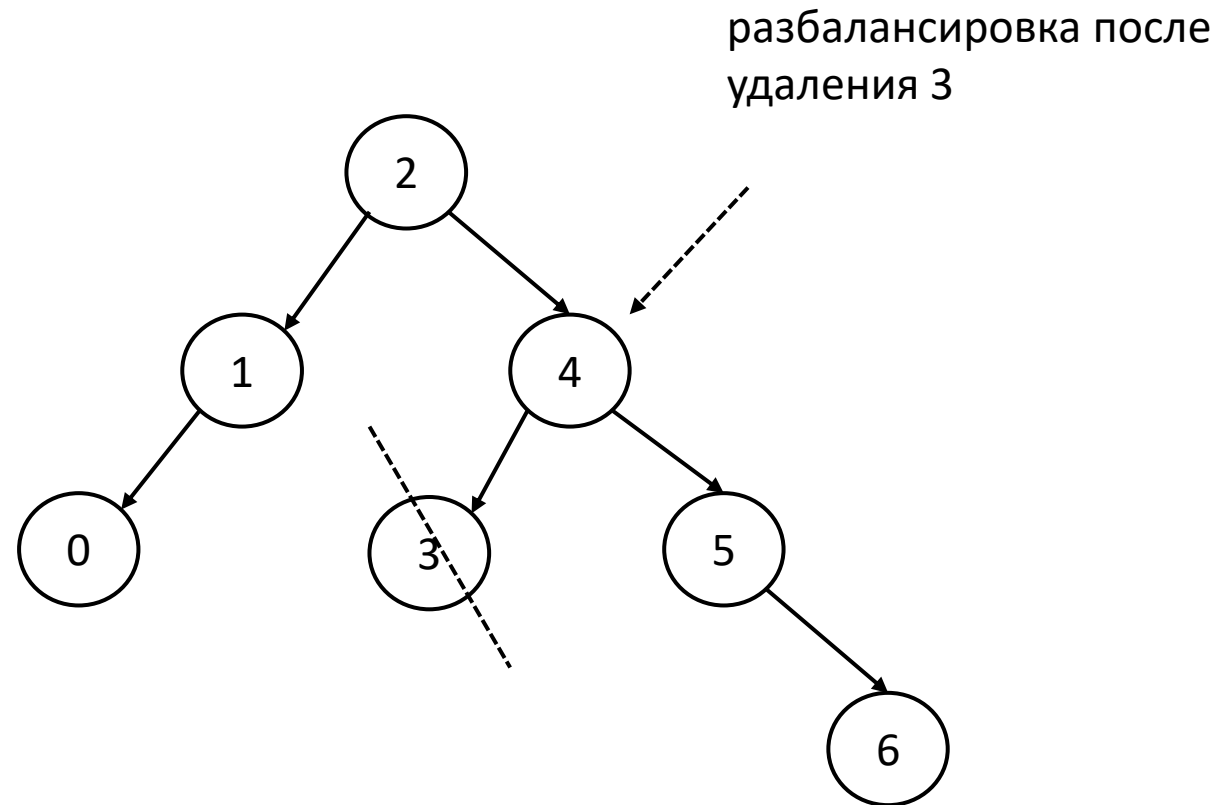
Однако, после добавления/удаления элемента может нарушиться свойство сбалансированности по высотам и его нужно восстановить.

Восстановление выполняют каждый раз, как только происходит нарушение сбалансированности.

# Разбалансировка после добавления элемента



# Разбалансировка после удаления элемента



# Балансировки

**LL** поворот (малое правое вращение, одинарный правый поворот)

**RR** поворот (малое левое вращение, одинарный левый поворот)

**LR** поворот (большое правое вращение, двойной правый поворот)

**RL** поворот (большое левое вращение, двойной левый поворот)

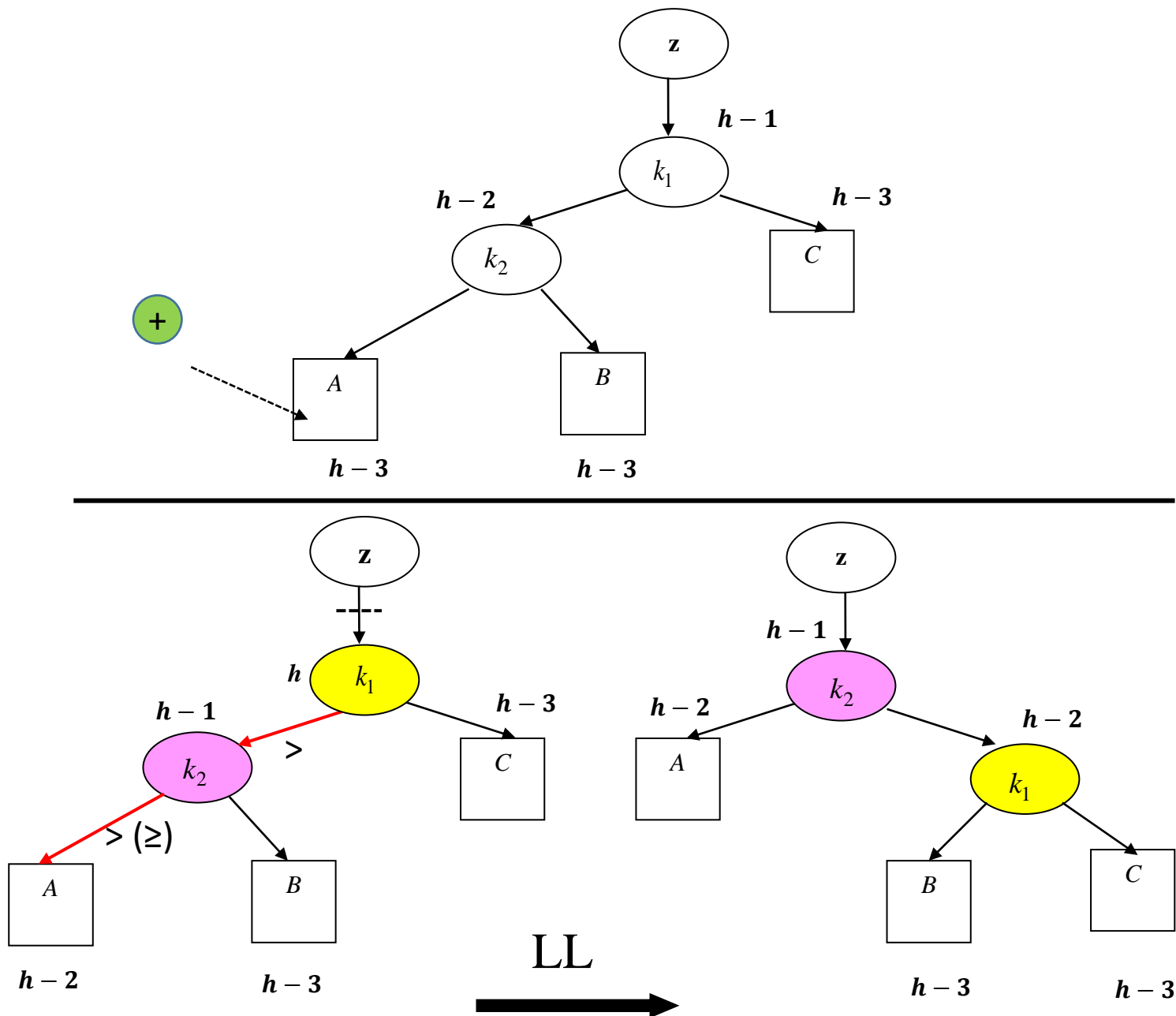


# LL –поворот

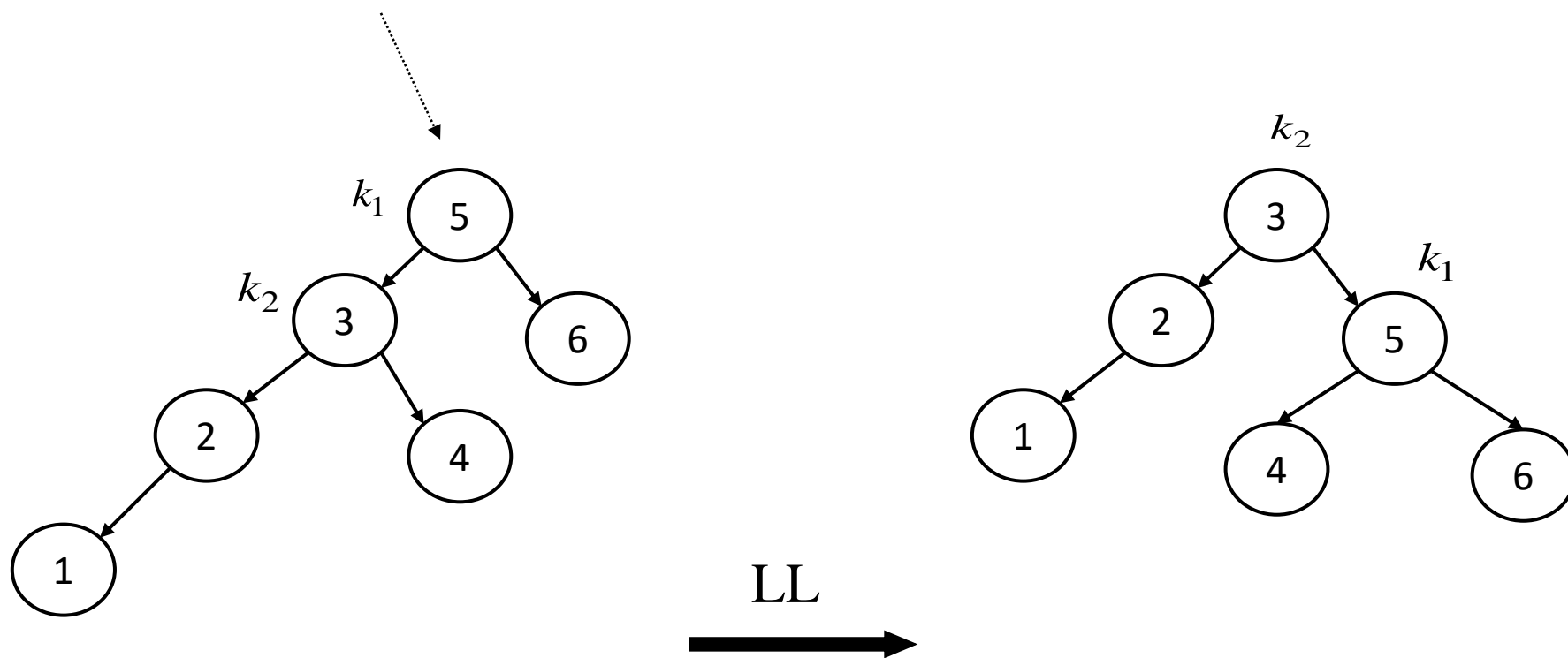
(малое правое вращение)

пусть  $k_1$  – вершина на максимальной глубине, для которой произошла разбалансировка и высота ее левого поддерева больше высоты правого поддерева на 2;

пусть  $k_2$  – левый сын вершины  $k_1$  и высота его левого поддерева (A) больше (или равна) высоты его правого поддерева (B);



# LL-поворот

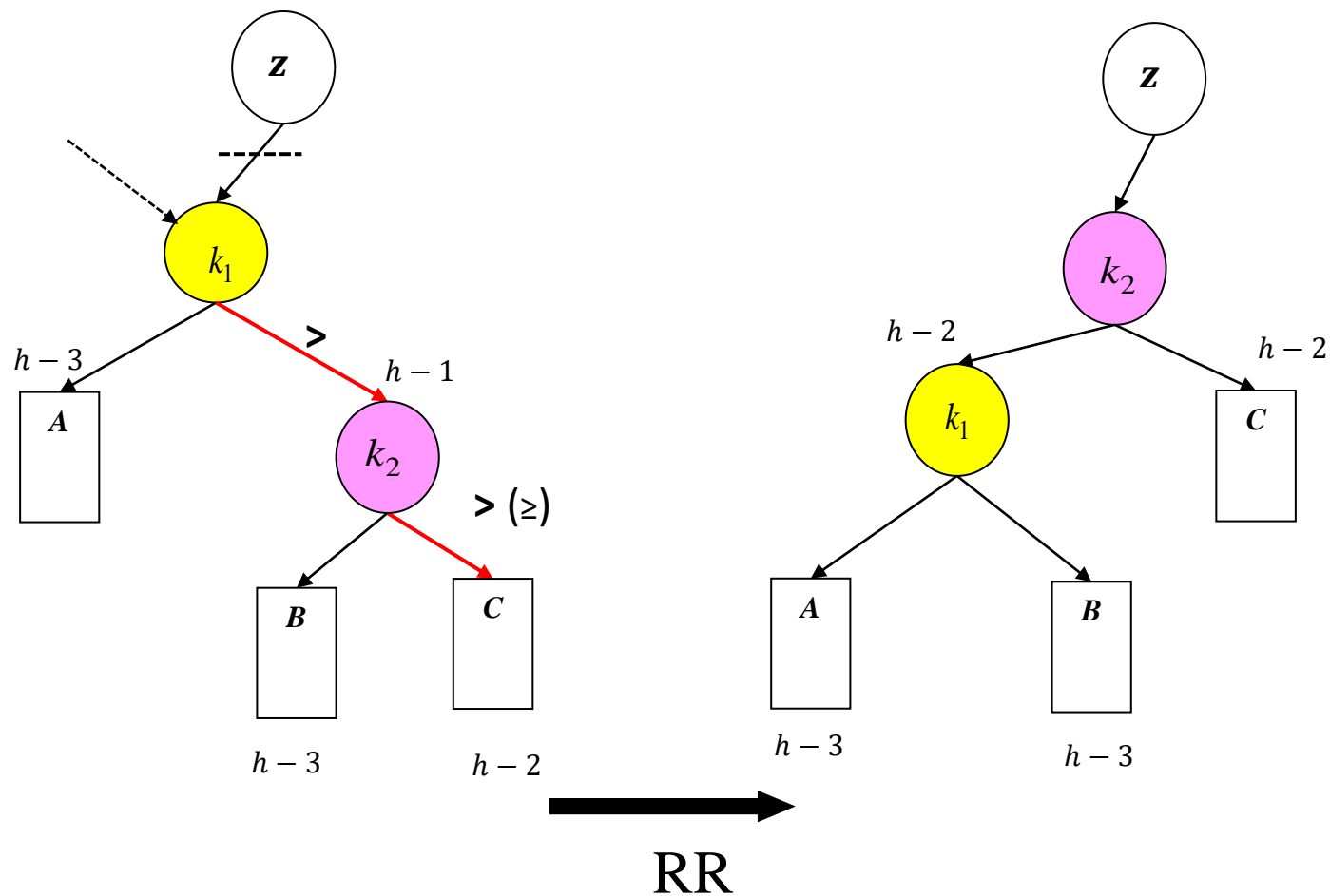


# RR –поворот

(малое левое вращение)

пусть  $k_1$  – вершина на максимальной глубине, для которой произошла разбалансировка и высота ее правого поддерева больше высоты левого поддерева на 2;

пусть  $k_2$  – правый сын вершины  $k_1$  и высота его правого поддерева (C) больше (или равна) высоты его левого поддерева (B);

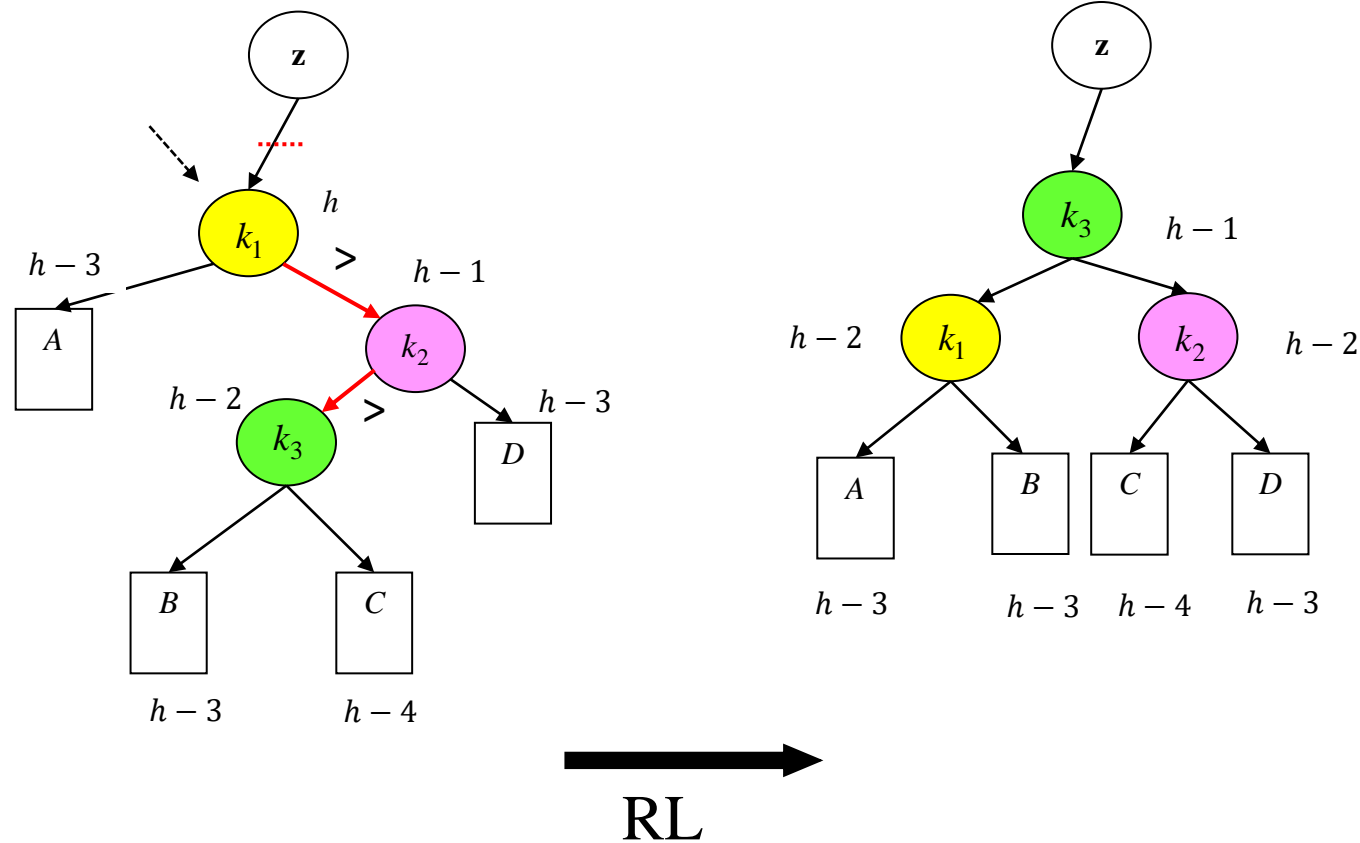


# RL –поворот

(большое левое  
вращение)

пусть  $k_1$  – вершина на максимальной глубине, для которой произошла разбалансировка и высота ее правого поддерева больше высоты левого поддерева на 2;

пусть  $k_2$  – правый сын вершины  $k_1$  и высота его левого поддерева (с корнем в вершине  $k_3$ ) больше высоты его правого поддерева ( $D$ );

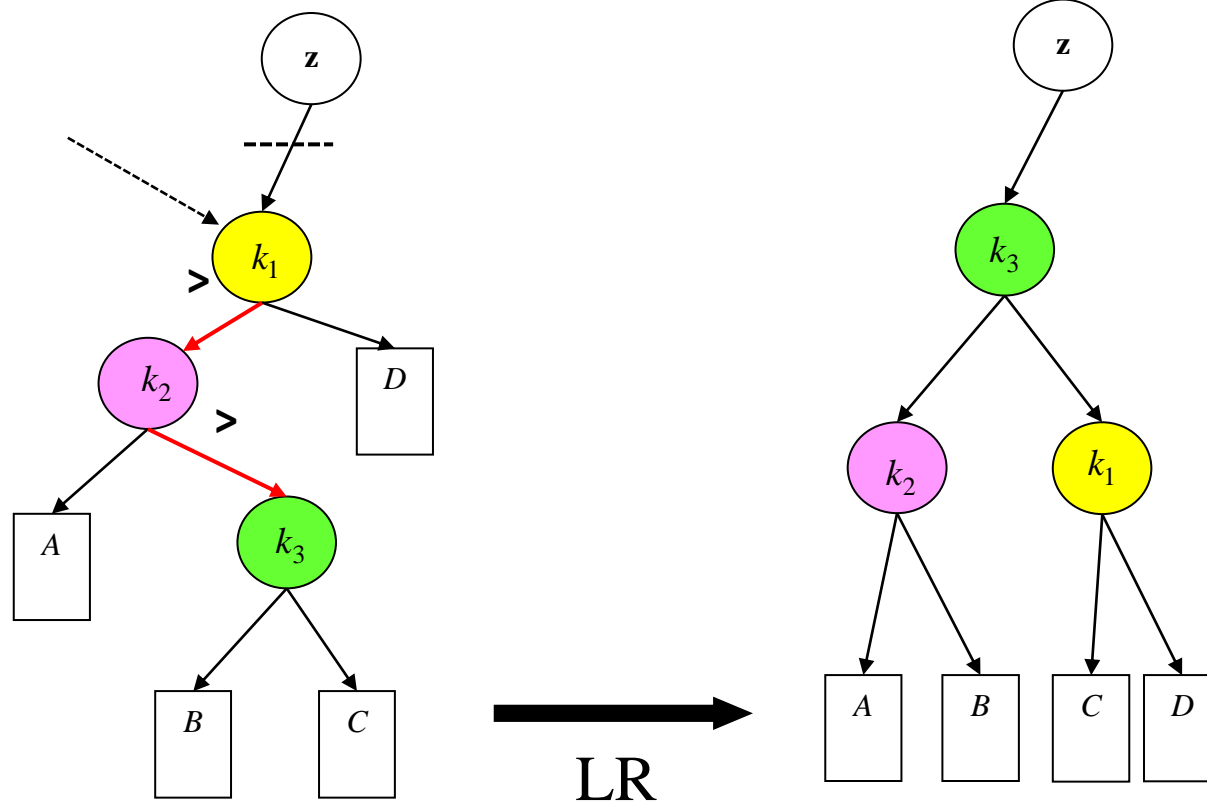


# LR –поворот

(большое правое вращение)

пусть  $k_1$  – вершина на максимальной глубине, для которой произошла разбалансировка и высота ее левого поддерева больше высоты правого поддерева на 2;

пусть  $k_2$  – левый сын вершины  $k_1$  и высота его правого поддерева (с корнем в вершине  $k_3$ ) больше высоты его левого поддерева ( $A$ );



# ОЦЕНКИ

Каждый из поворотов (LL, RR, LR, RL) выполняется за  $O(1)$ , если известна ссылка на разбалансированную вершину.

После выполнения операции **добавления** **элемента** разбалансировка может произойти сразу у нескольких вершин (эти вершины лежат на пути от корня дерева к отцу добавляемой вершины):

- ✓ сначала необходимо найти ту из разбалансированных вершин, которая наиболее удалена от корня дерева и выполнить для неё один из поворотов;
- ✓ в результате одной балансировки для всех вершин дерева будет выполняться свойство сбалансированности по высотам.

Таким образом, на весь процесс восстановления свойства сбалансированности будет потрачено время  **$O(\log n)$** .

Процедура **добавления элемента**:

- ✓ поиск отца для вершины  $x$  ;
- ✓ добавление вершины  $x$ ;
- ✓ поиск разбалансированной вершины;
- ✓ один из поворотов для восстановления свойства сбалансированности по высотам;

будет выполнена за время  **$O(\log n)$** .



При **удалении элемента  $x$**  разбалансировка может произойти только у одной вершины:

- ✓ найдём разбалансированную вершину и выполним для неё поворот;
- ✓ однако, после поворота может появиться ещё одна разбалансированная вершина и т.д.;
- ✓ выполним повторные балансировки (число повторных балансировок ограничено высотой дерева, так как каждый раз балансируемая вершина находится на большей высоте).

Так как удаление одного элемента из бинарного поискового дерева выполняется за  $O(\log n)$ , одна балансировка – за  $O(1)$ , а число повторных балансировок ограничено высотой дерева  $h = O(\log n)$ , то вся процедура удаления элемента выполняется за время –  $O(\log n)$ .

**ПРИМЕР**

Построить AVL-дерево для последовательности чисел:

**7, 8, 2, 3, 4, 6, 1, 9, 10, 11, 5**

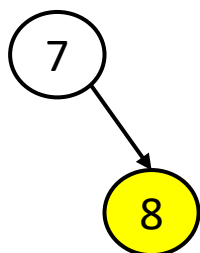
- ✓ построение осуществляется последовательным добавлением элементов;
- ✓ если на некотором шаге произошла разбалансировка, то для её восстановления выполнить поворот.

7 8 2 3 4 6 1 9 10 11 5

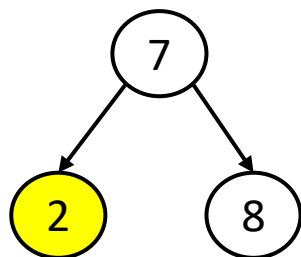
7:



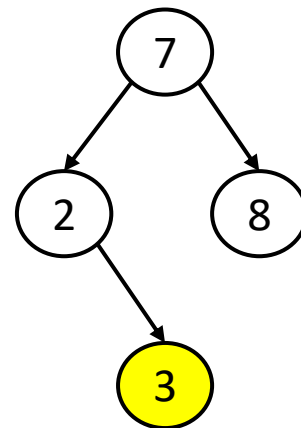
8:



2:

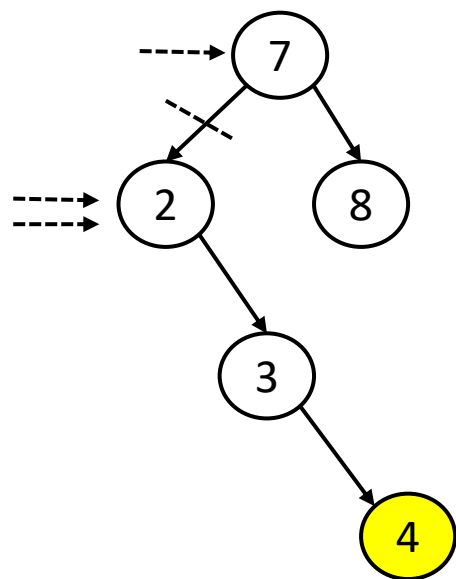


3:

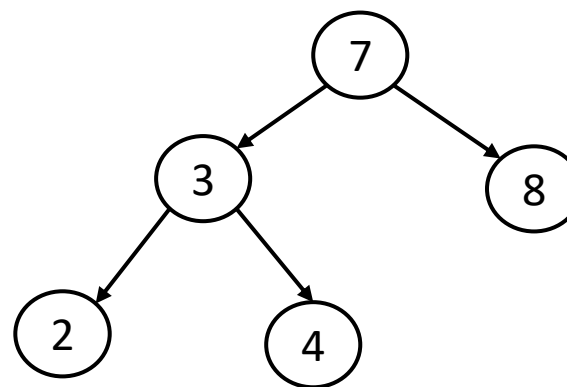


7 8 2 3 4 6 1 9 10 11 5

4:

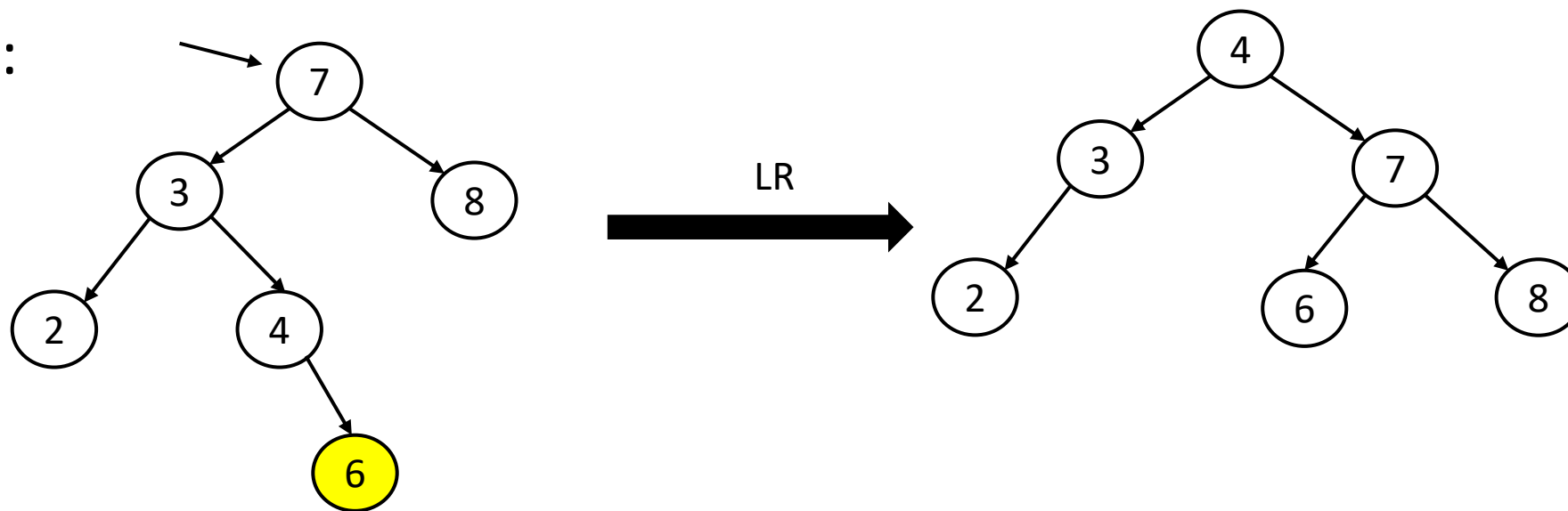


RR



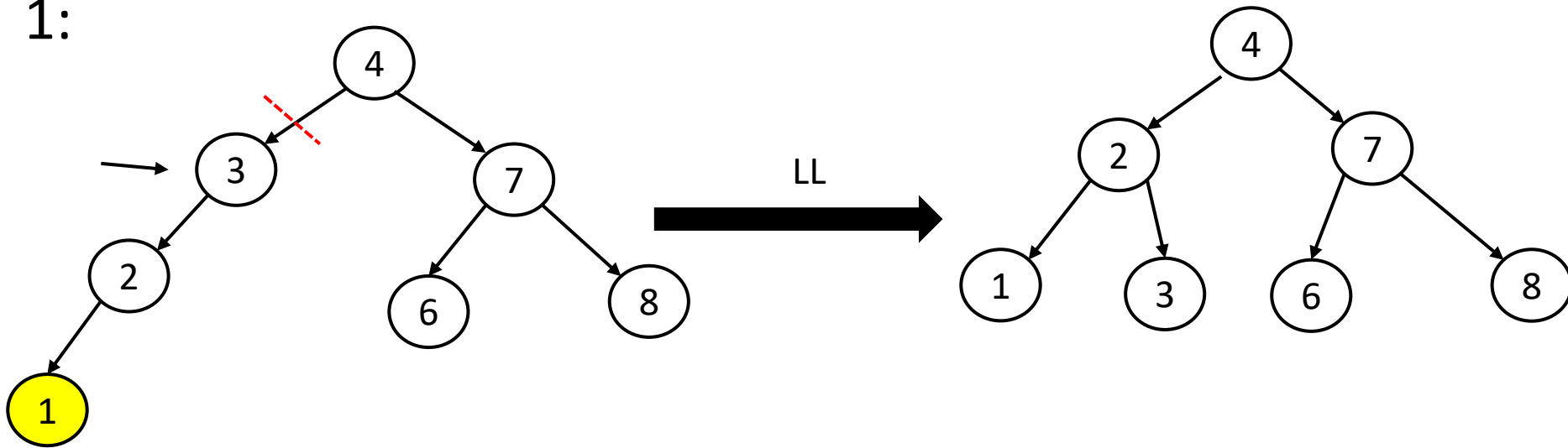
7 8 2 3 4 6 1 9 10 11 5

6:



7 8 2 3 4 6 1 9 10 11 5

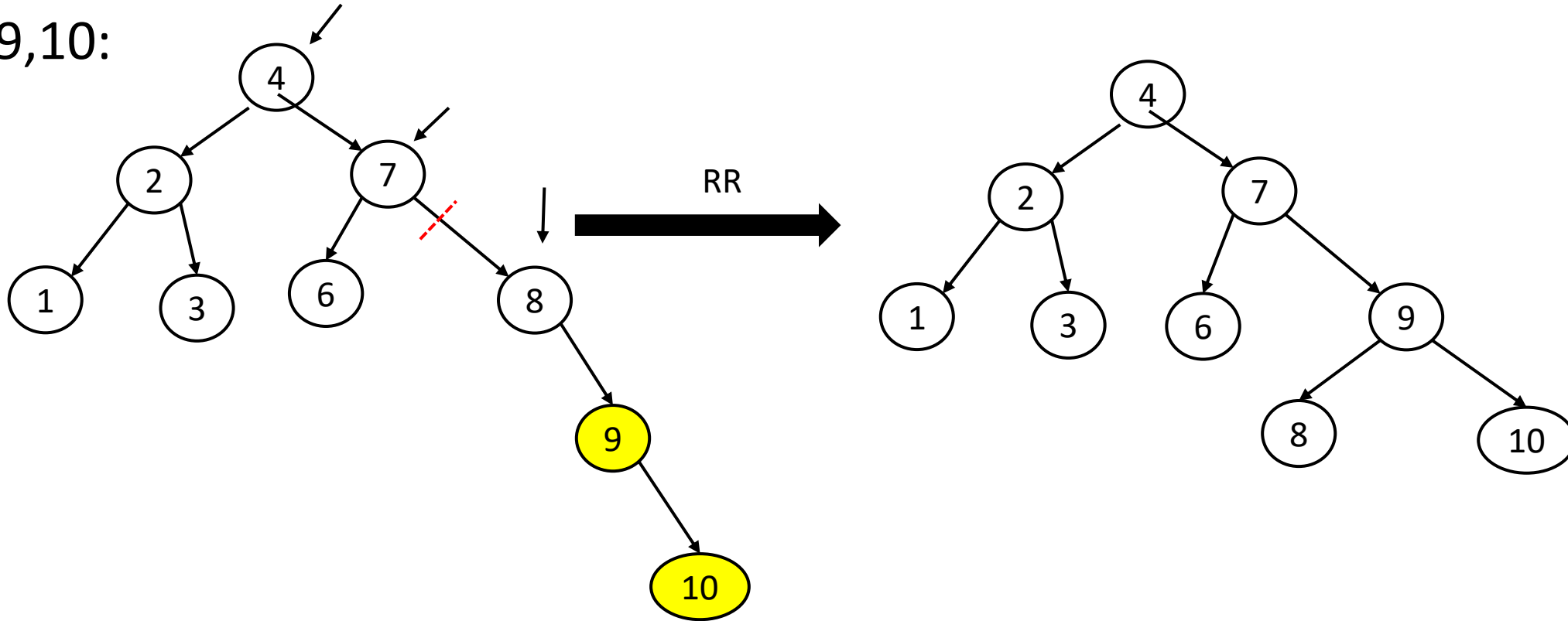
1:



Построить AVL-дерево для последовательности чисел:

7 8 2 3 4 6 1 9 10 11 5

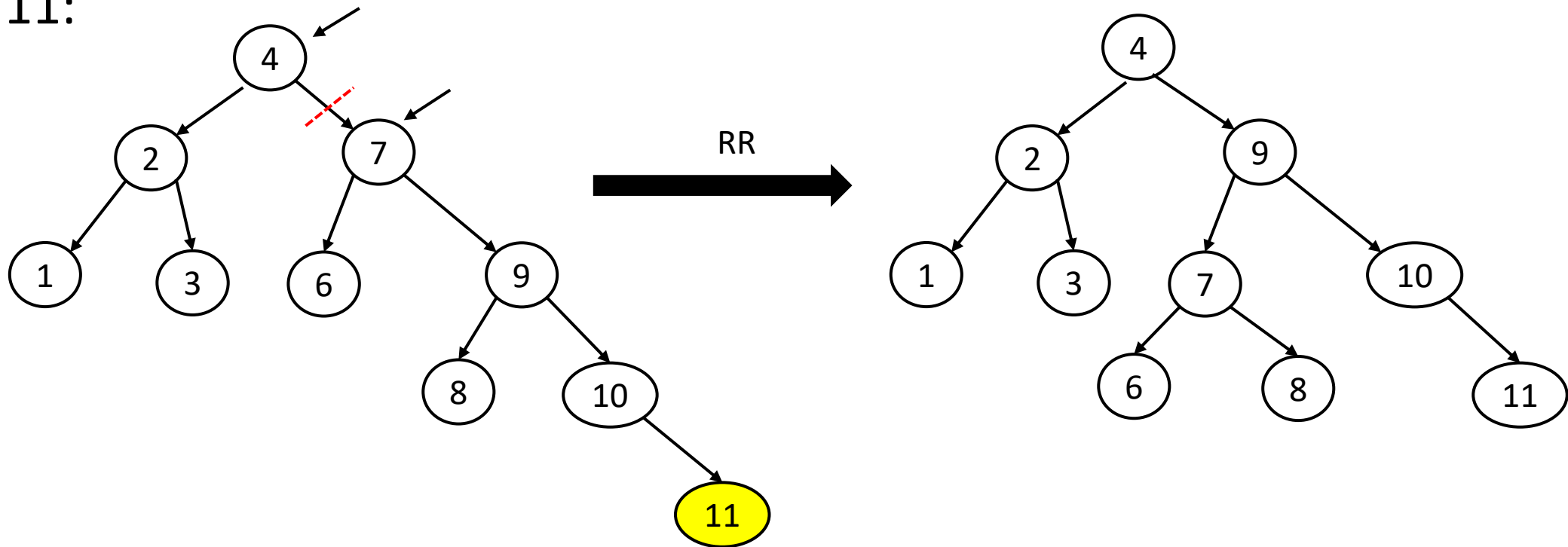
9,10:





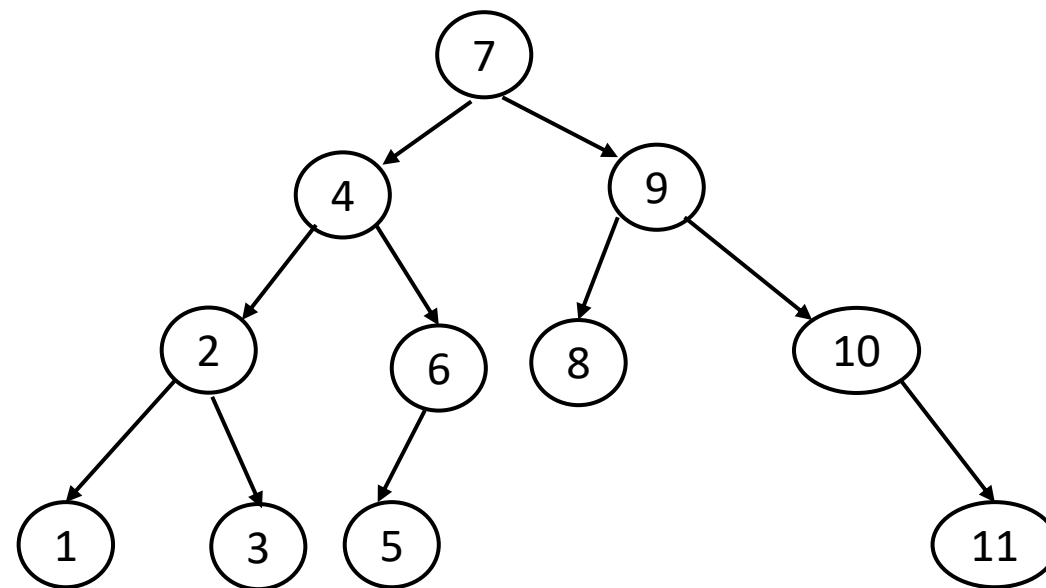
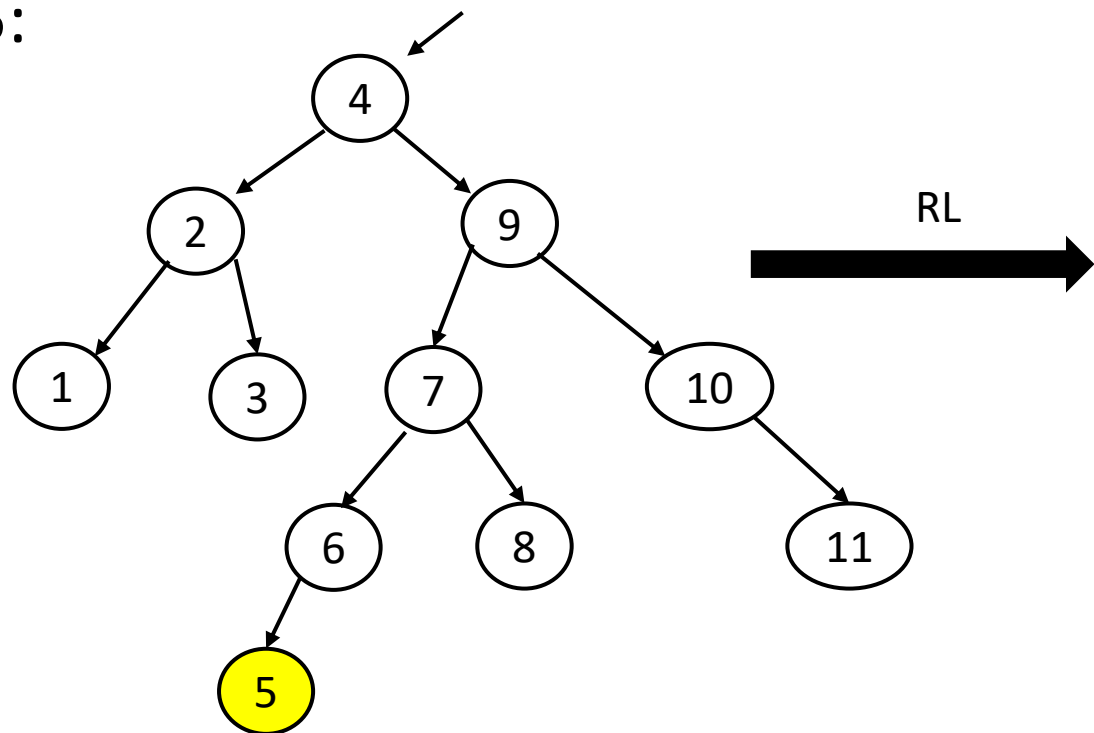
7 8 2 3 4 6 1 9 10 11 5

11:



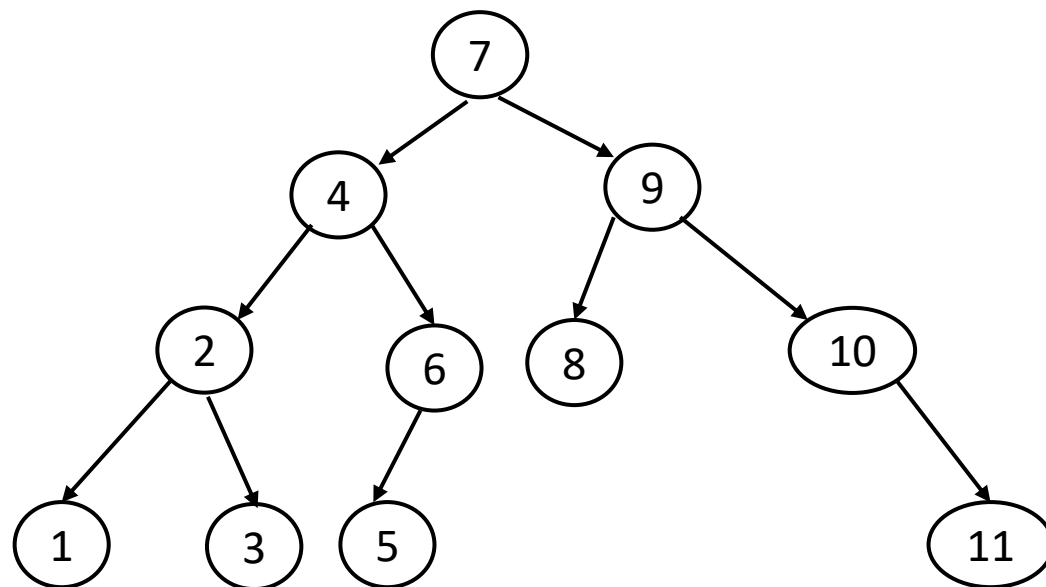
7 8 2 3 4 6 1 9 10 11 5

5:



задача решена

7, 8, 2, 3, 4, 6, 1, 9, 10, 11, 5



# Сортировка деревом

Предположим, что на вход поступаю числа, среди которых нет повторяющихся.

1. По последовательности чисел сначала построим AVL-дерево.

$$O(n \cdot \log n)$$

2. Выполним внутренний левый обход построенного дерева.

$$O(n)$$

Время работы алгоритма сортировки деревом:

$$O(n \cdot \log n)$$

# Абстрактный тип данных: множество (set)

**Множество** (англ. set) — хранит набор попарно различных объектов без определённого порядка.

Интерфейс множества включает три основные операции:

- 1) **Insert(x)** — добавить в множество ключ  $x$ ;
- 2) **Contains(x)** — проверить, содержится ли в множестве ключ  $x$ ;
- 3) **Remove(x)** — удалить ключ  $x$  из множества.

Для реализации интерфейса множества обычно используются такие структуры данных, как:

- **сбалансированные поисковые деревья**: например, AVL-деревья, 2-3-деревья, красно-чёрные деревья.
- **хеш-таблицы**.

В стандартной библиотеке **C++** есть контейнер `std::set`, который реализует множество на основе **сбалансированного дерева** (обычно красно-чёрного), и контейнер `std::unordered_set`, построенный на базе хеш-таблицы.

В языке **Java** определён интерфейс `Set`, у которого есть несколько реализаций, среди которых классы `TreeSet` (работает на **основе красно-чёрного дерева**) и `HashSet` (на основе **хеш-таблицы**).

В языке **Python** есть только встроенный тип `set`, использующий **хеширование**, но нет готового класса множества, построенного на сбалансированных деревьях.

# Абстрактный тип данных ассоциативный массив (**map**)

**Ассоциативный массив** (англ. associative array), или отображение (англ. map), или словарь (англ. dictionary), — хранит пары вида (ключ, значение), при этом каждый ключ встречается не более одного раза.

Название «ассоциативный» происходит от того, что значения ассоциируются с ключами.

Интерфейс ассоциативного массива включает операции:

- 1) **Insert(k,v)** — добавить пару, состоящую из ключа *k* и значения *v*;
- 2) **Find(k)** — найти значение, ассоциированное с ключом *k*, или сообщить, что значения, связанного с заданным ключом, нет;
- 3) **Remove(k)** — удалить пару, ключ в которой равен *k*.

Данный интерфейс реализуется на практике теми же способами, что и интерфейс множества. Реализация технически немного сложнее, чем множества, но использует те же идеи.

Для языка программирования **C++** в стандартной библиотеке доступен контейнер `std::map`, работающий на основе **сбалансированного дерева** (обычно красно-чёрного), и контейнер `std::unordered_map`, работающий на основе **хеш-таблицы**.

В языке **Java** определён интерфейс `Map`, который реализуется несколькими классами, в частности классом `TreeMap` (базируется на **красно-чёрном дереве**) и `HashMap` (базируется на **хеш-таблице**).

В языке **Python** очень широко используется встроенный тип `dict`. Этот словарь использует внутри **хеширование**.



БЕЛОРУССКИЙ  
ГОСУДАРСТВЕННЫЙ  
УНИВЕРСИТЕТ

Спасибо за внимание!