Информатика

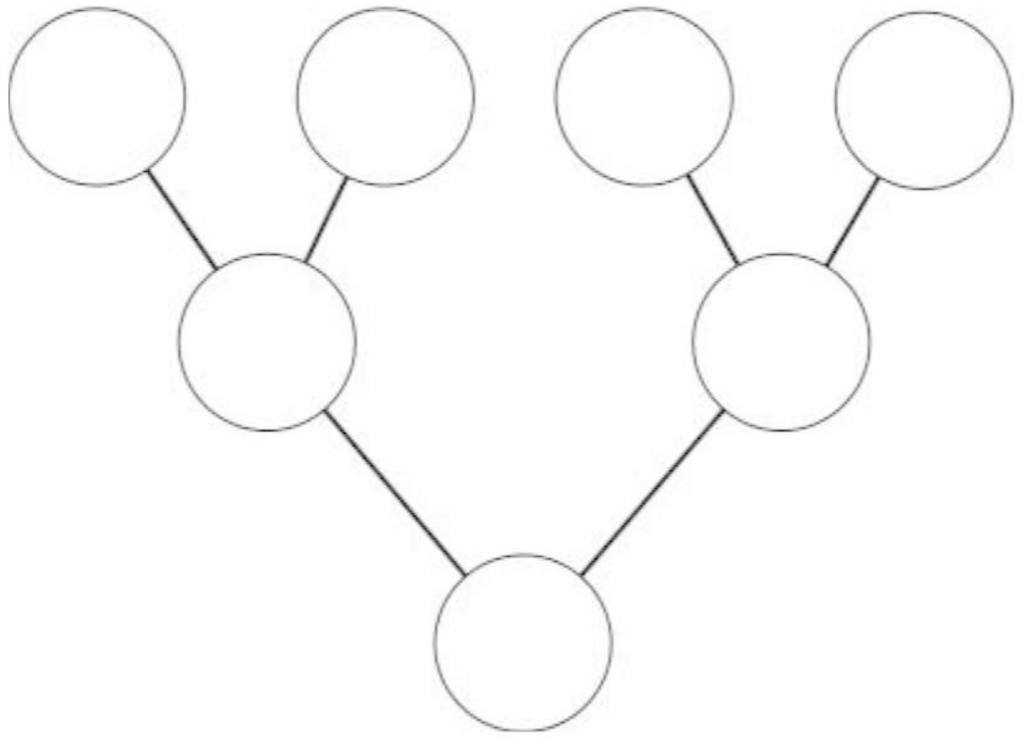
Осень 2024

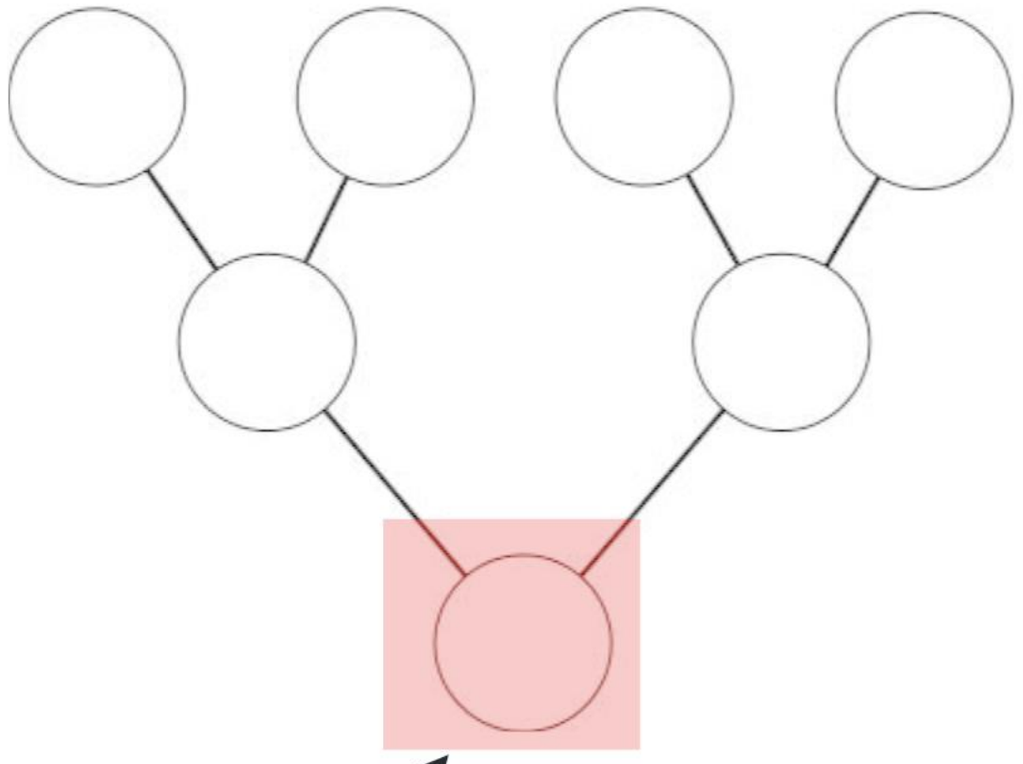
Баранов Максим Александрович

О ЧЁМ ПОГОВОРИМ?

1. АТД дерево
2. Реализация деревьев
3. Бинарное дерево
4. Бинарное дерево поиска
5. Сбалансированные деревья

АТД дерево

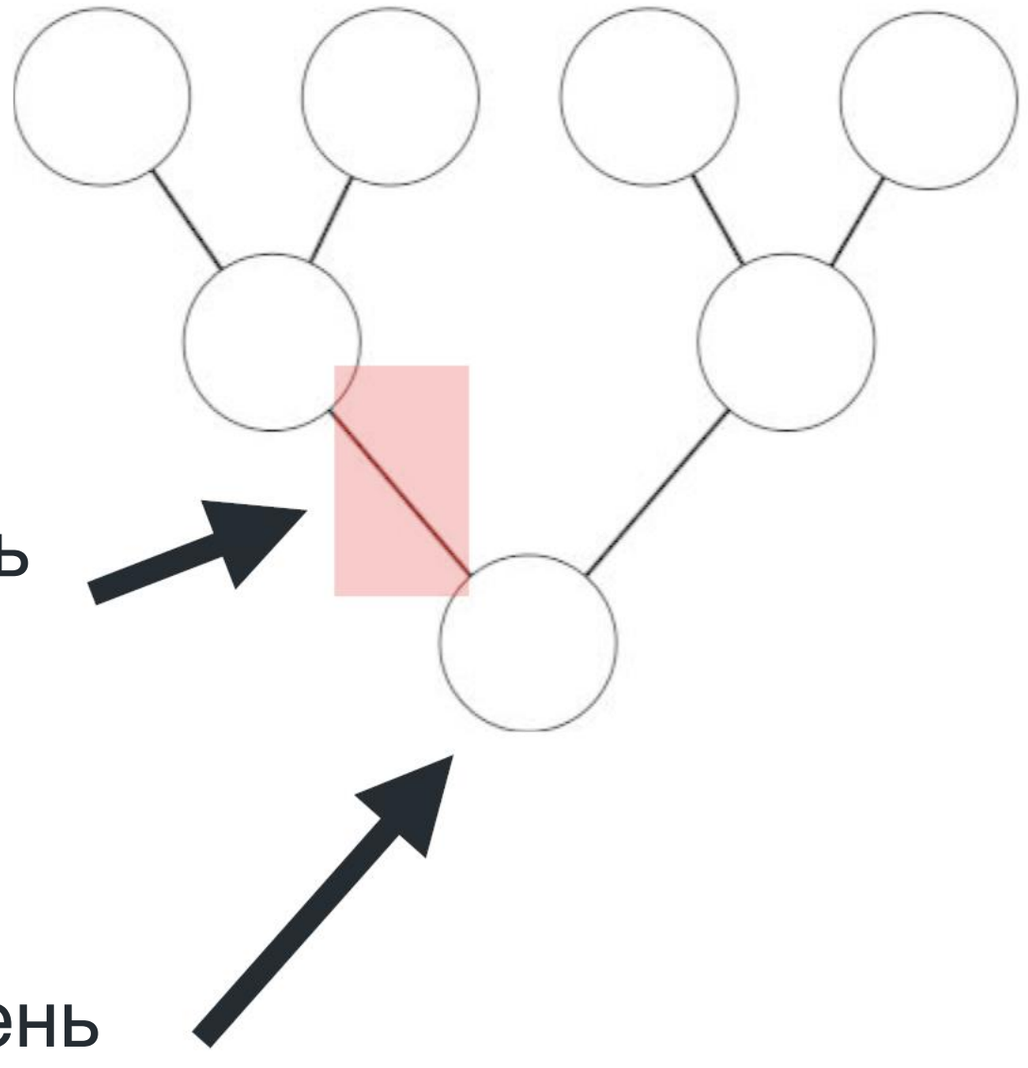




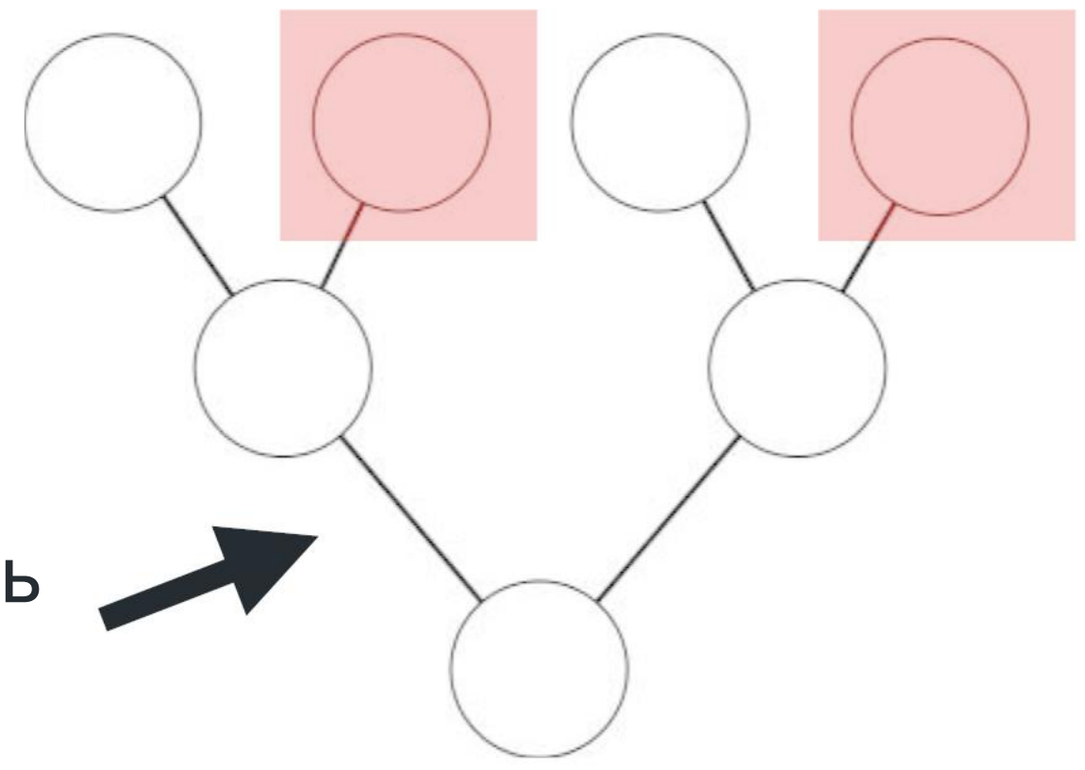
корень

ветвь

корень



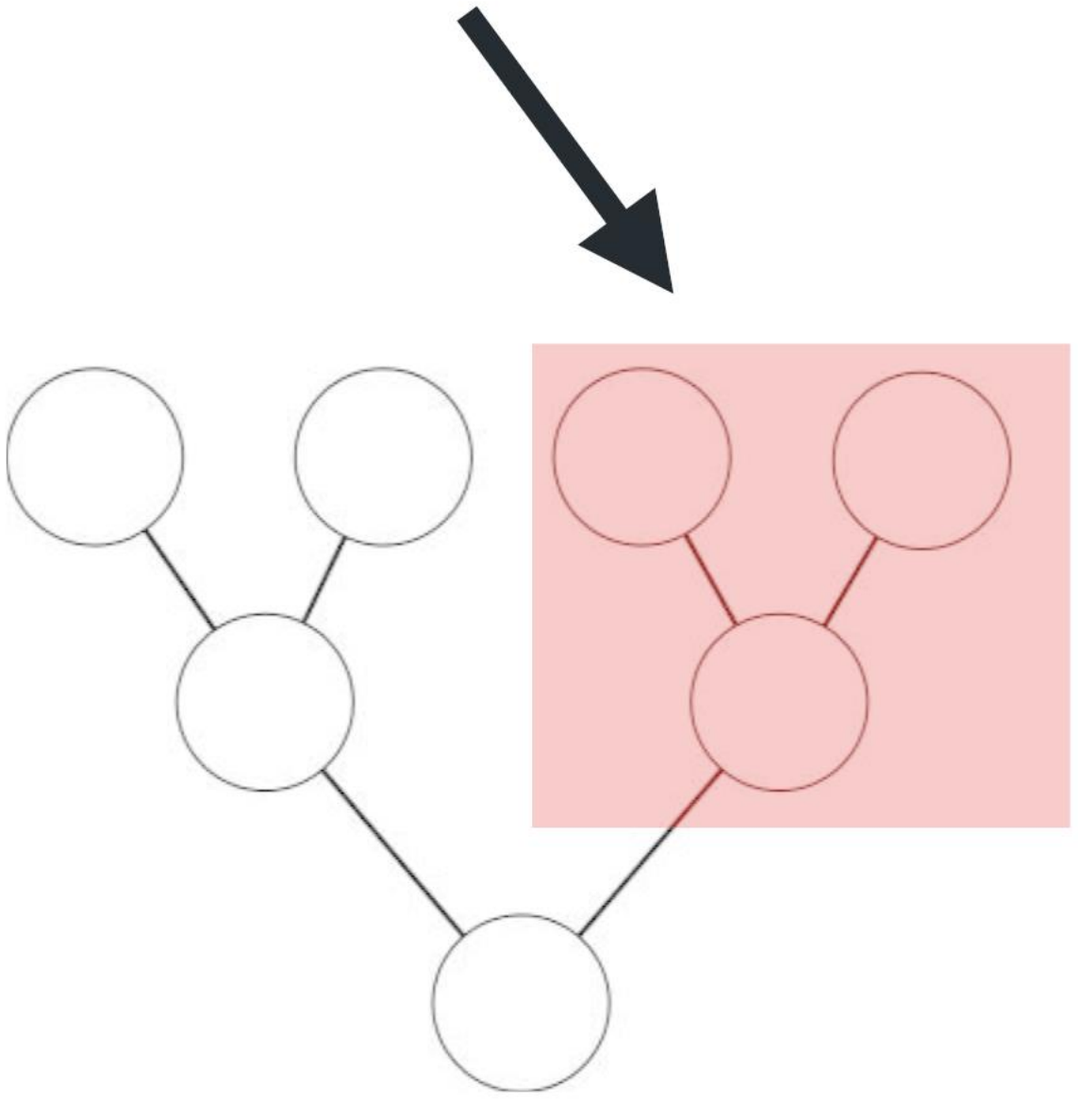
листья

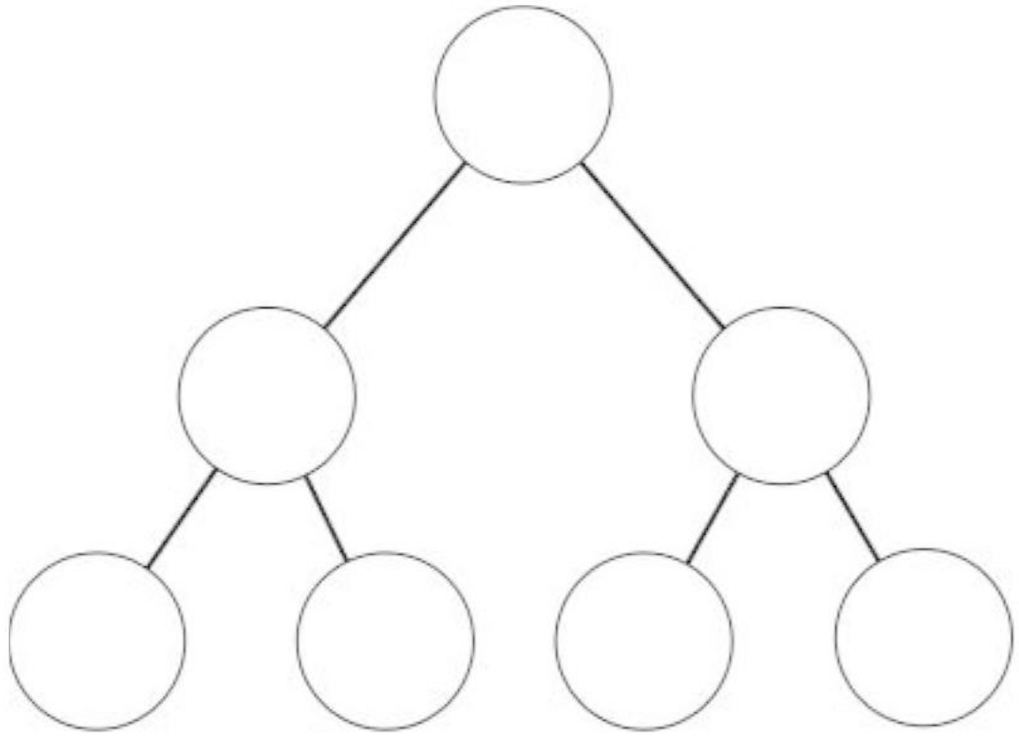


ветвь

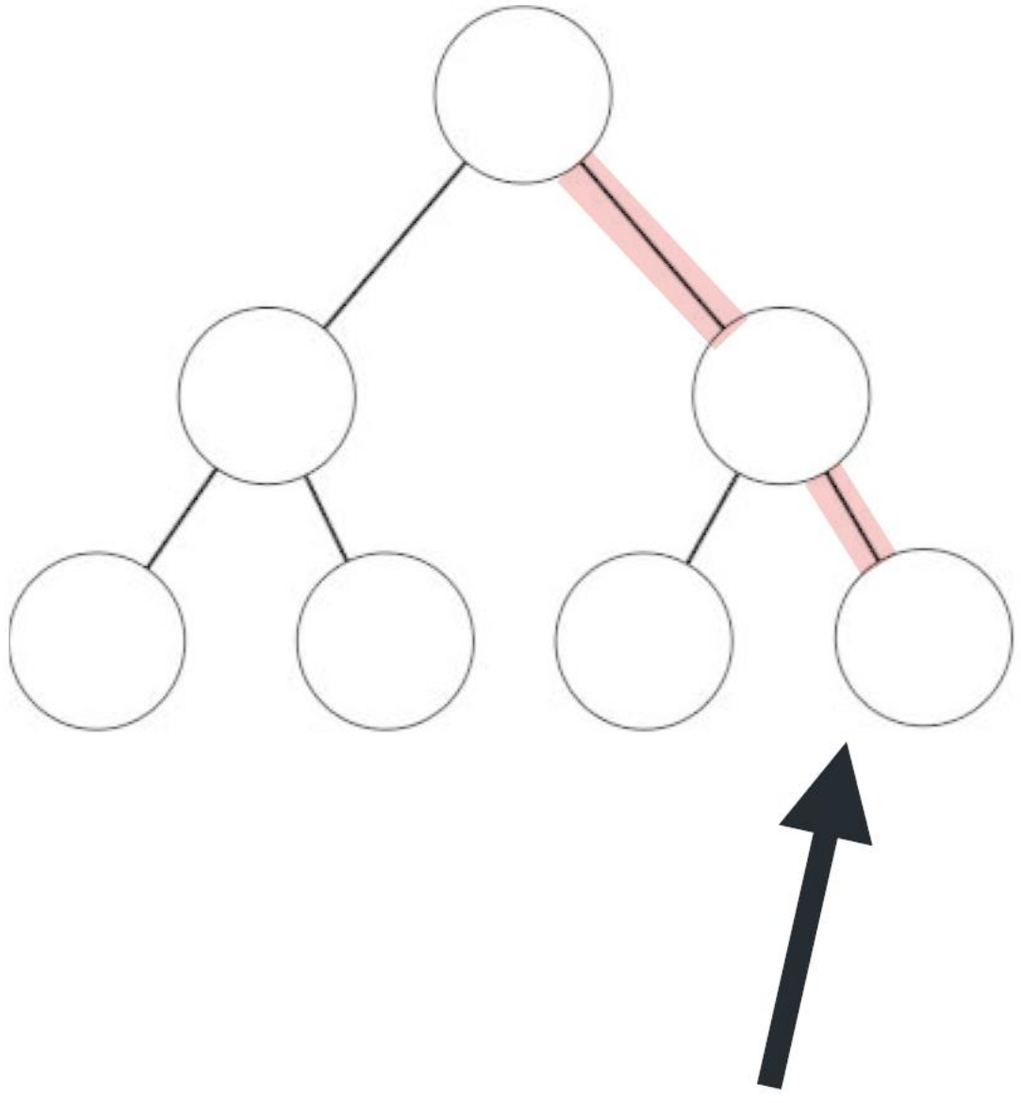
корень

поддерево

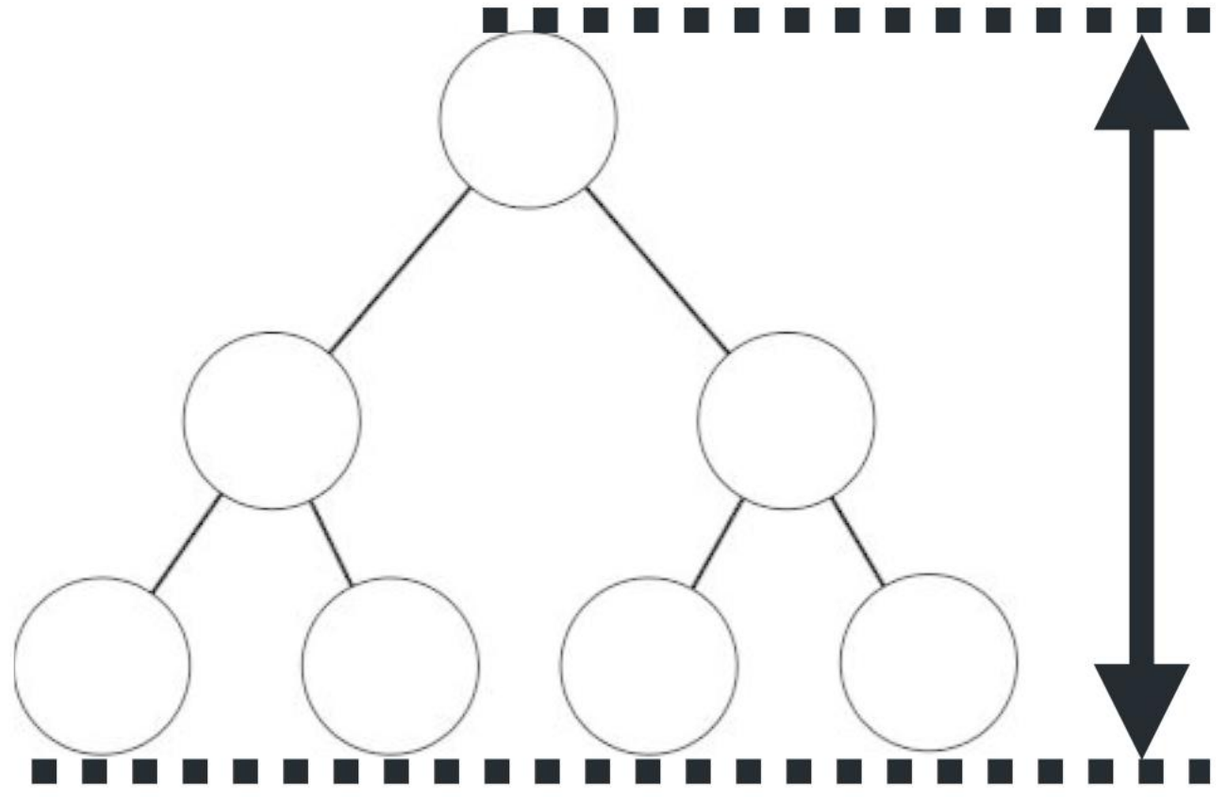




дерево (корневое дерево)

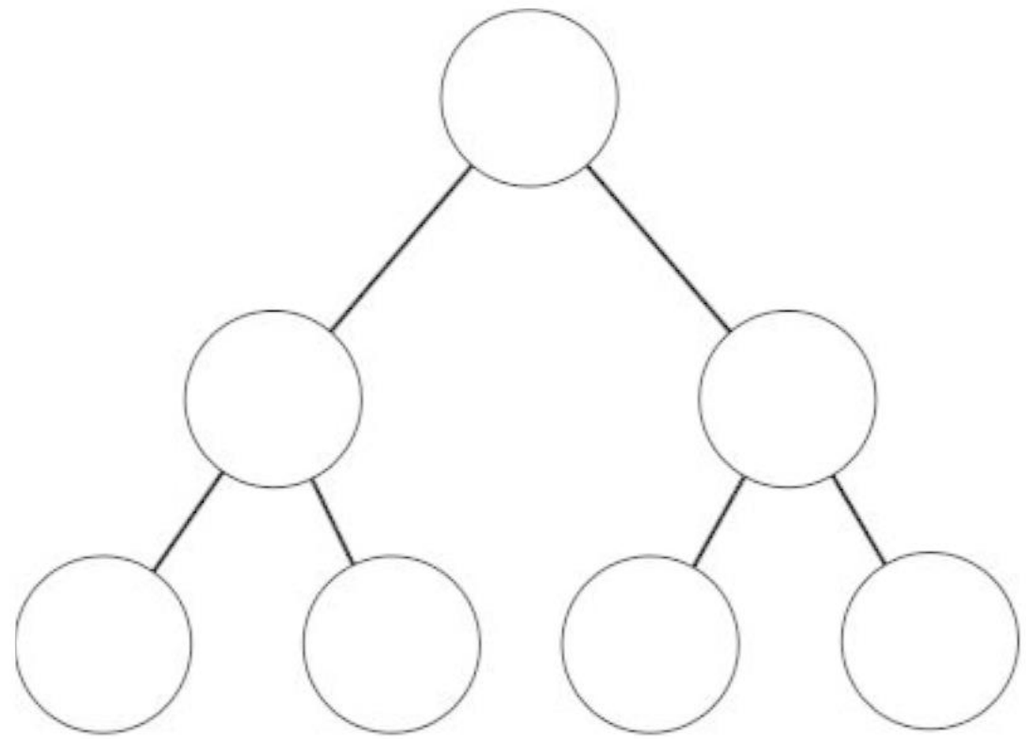


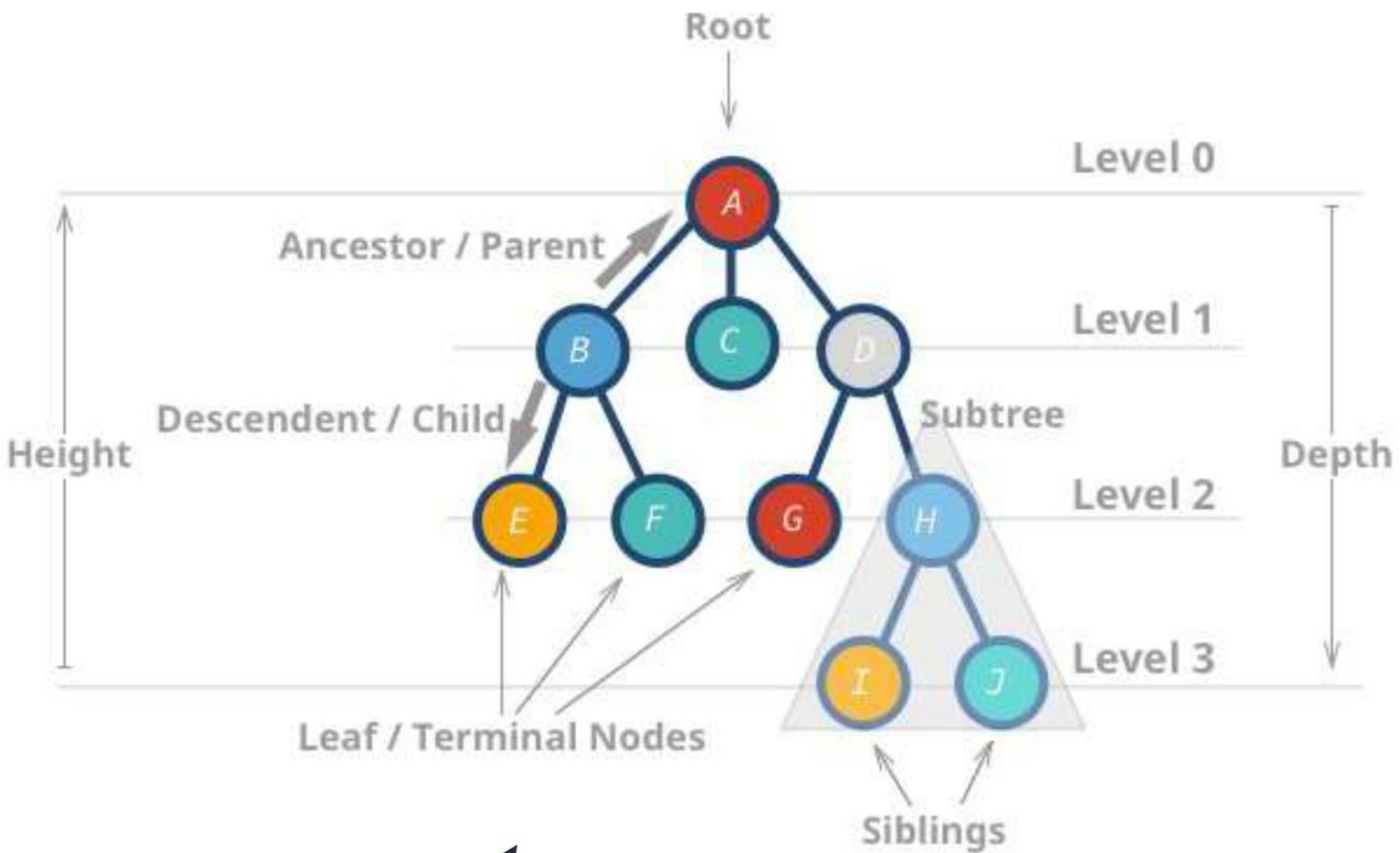
глубина вершины = длина пути от корня к вершине



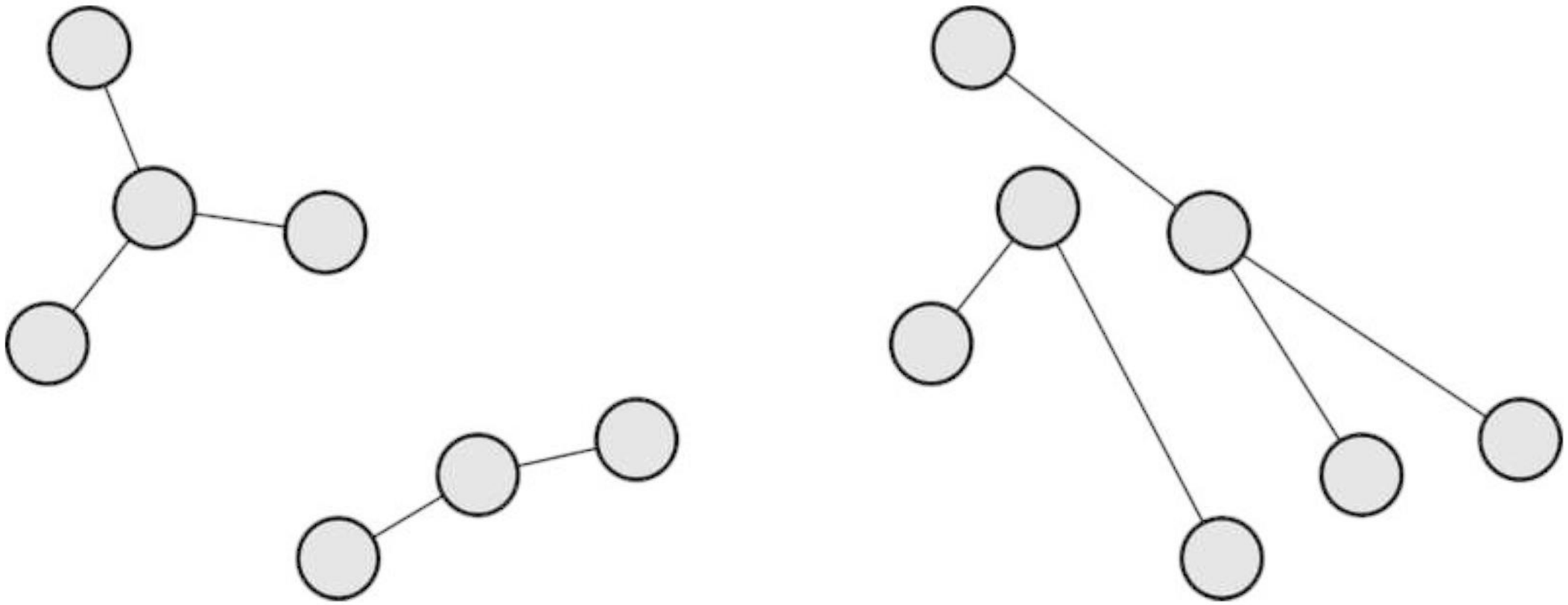
высота дерева = максимальная глубина

$$|\text{Ветвей}| = |\text{Вершин}| - 1$$

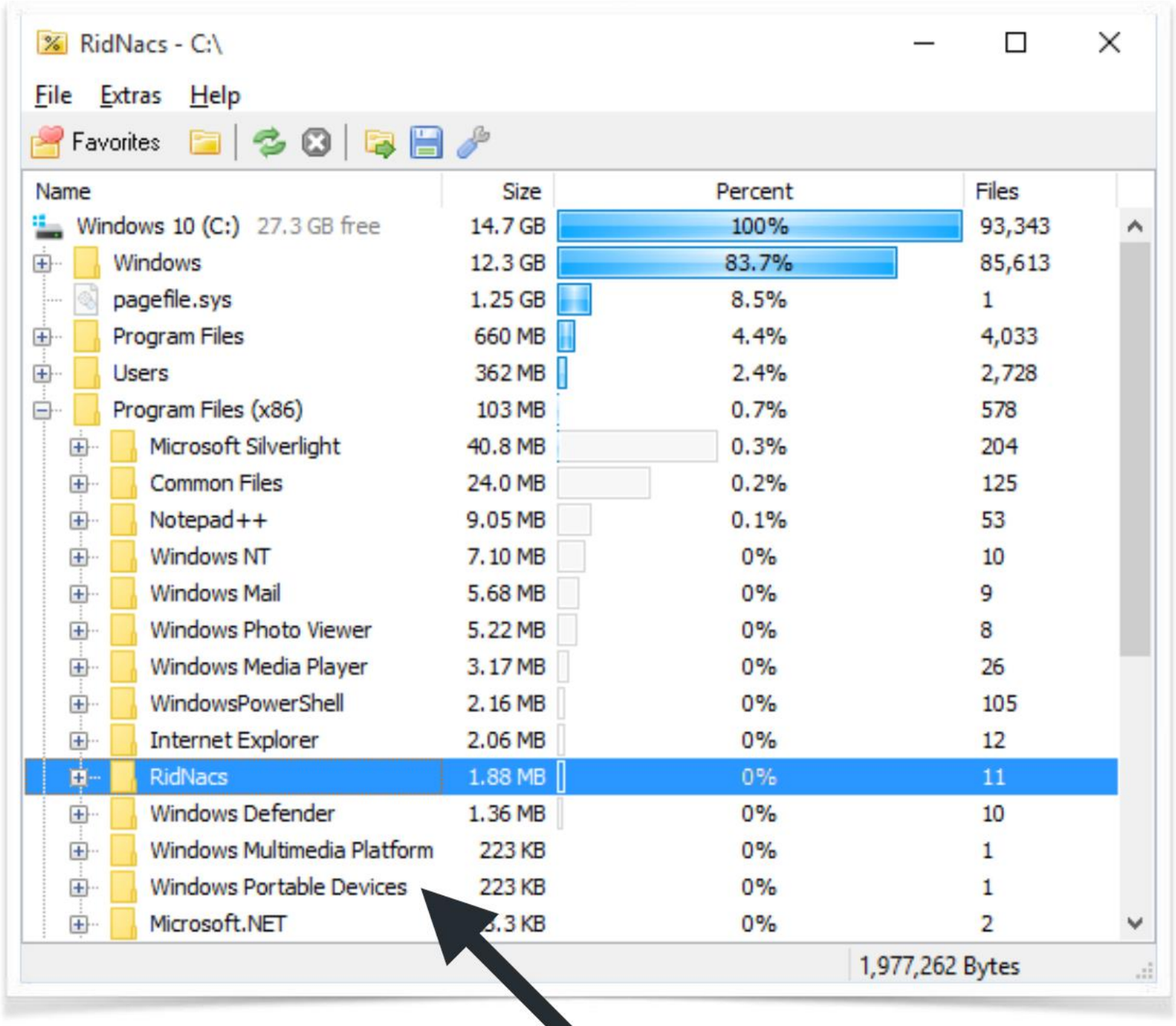




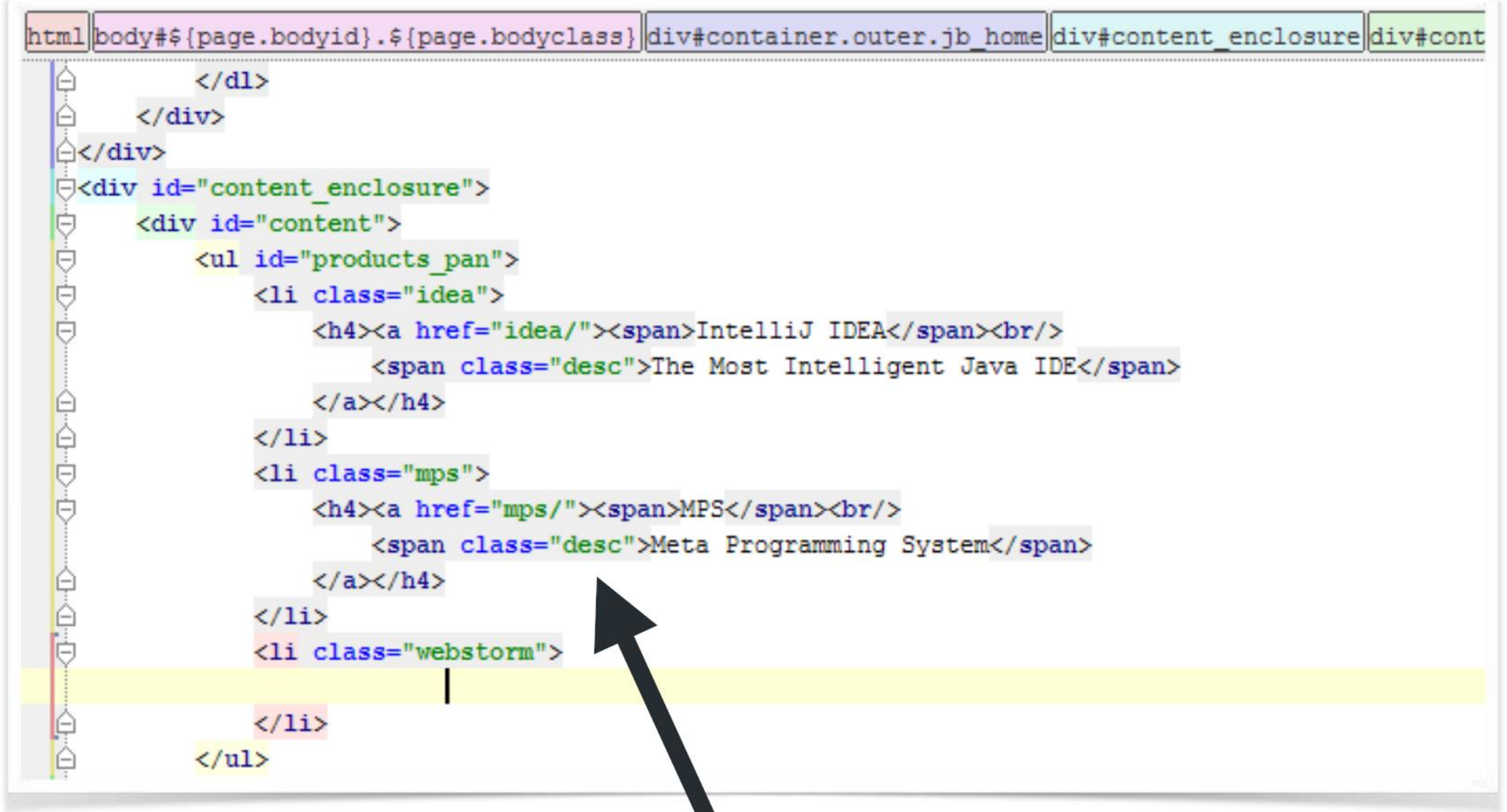
термины на английском



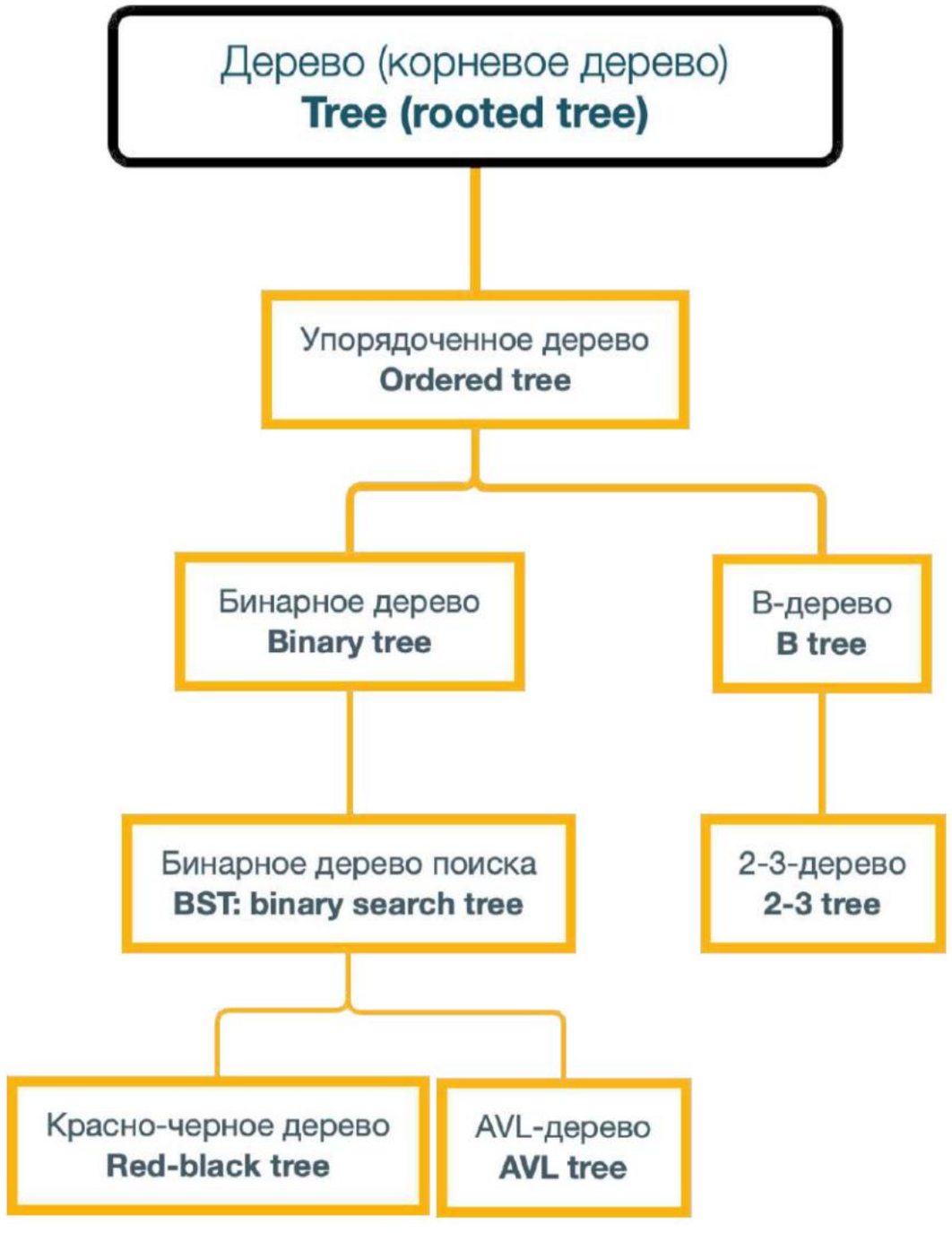
лес = набор деревьев (



деревья встречаются часто

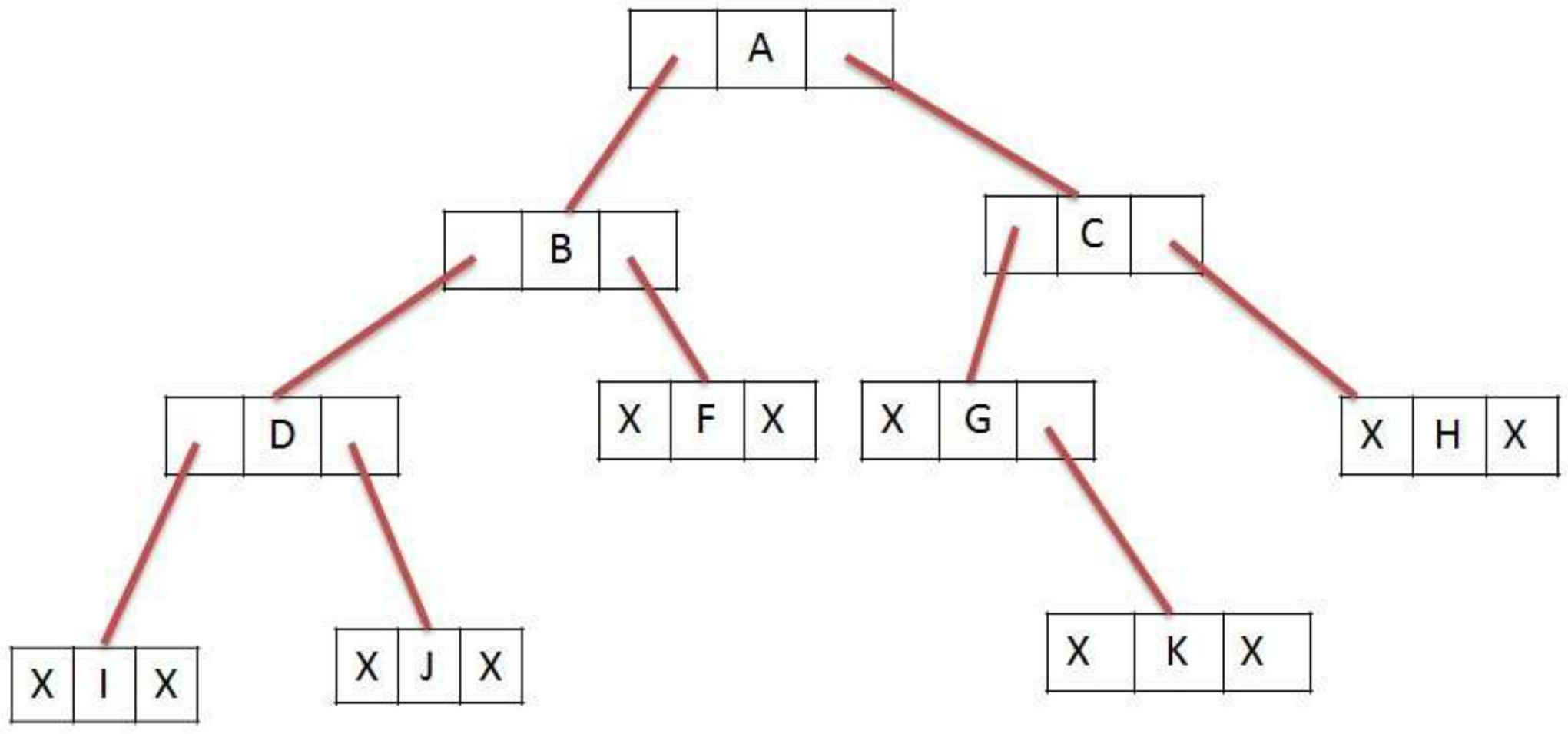


деревья встречаются часто

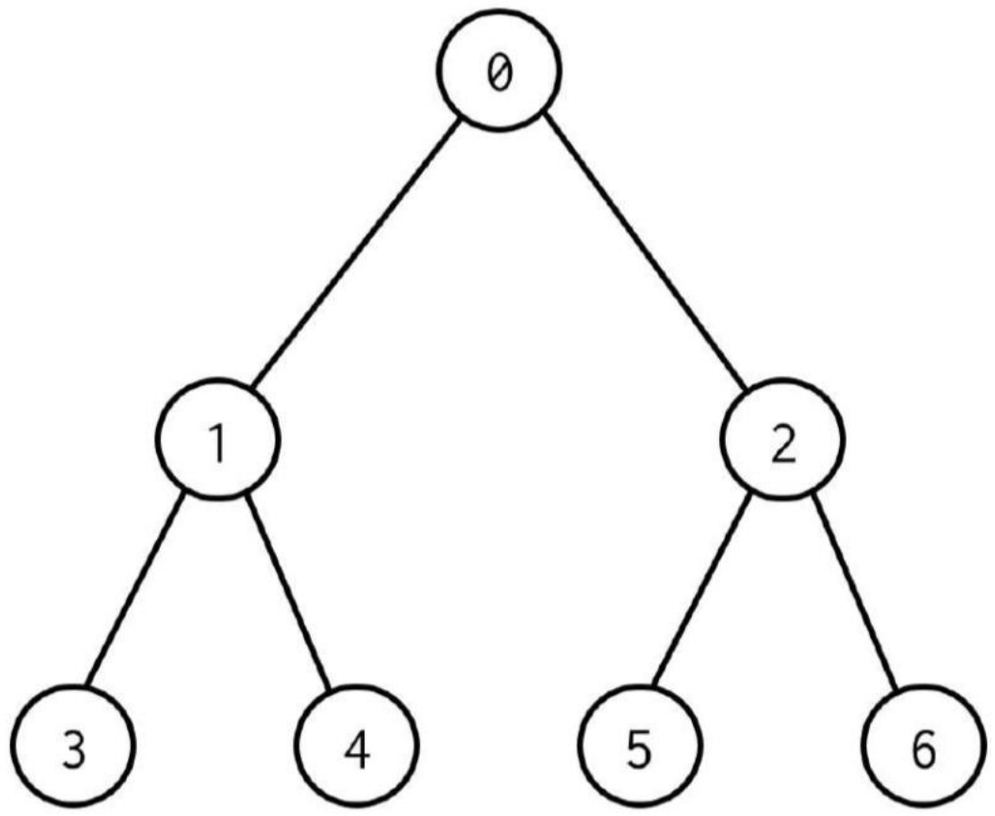

Дерево (корневое дерево)
Tree (rooted tree)
Упорядоченное дерево
Ordered tree
Бинарное дерево
Binary tree
B-дерево
B tree
Бинарное дерево поиска
BST: binary search tree
2-3-дерево
2-3 tree
Красно-черное дерево
Red-black tree
AVL-дерево
AVL tree

Реализация деревьев

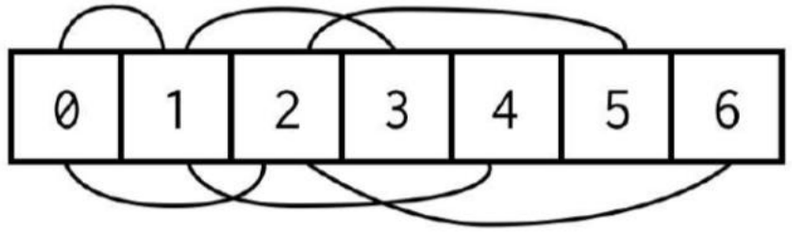
На связном списке



На массиве



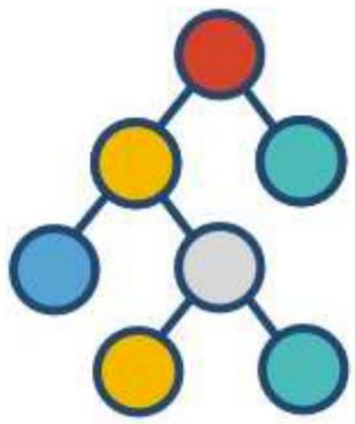
a[i] has children a[2*i+1] and a[2*i+2]



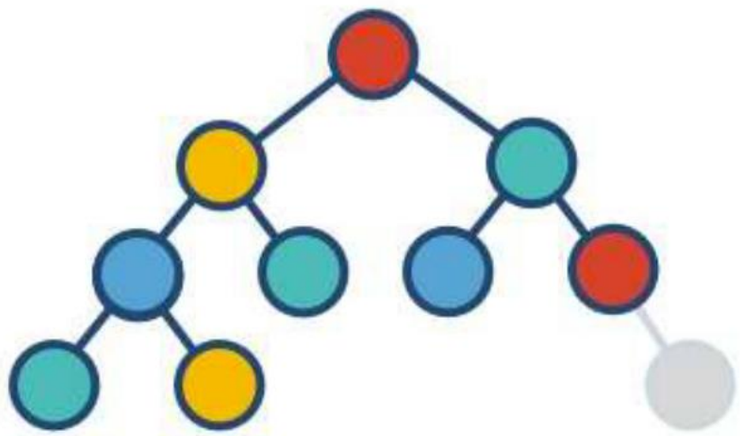
На связном списке в Python

```python
class Node:
    def __init__(self, value):
        self.value = value
        self.left_child = None
        self.right_child = None


a, b, c = Node(1), Node(2), Node(3)
a.left_child = b
a.right_child = c
```

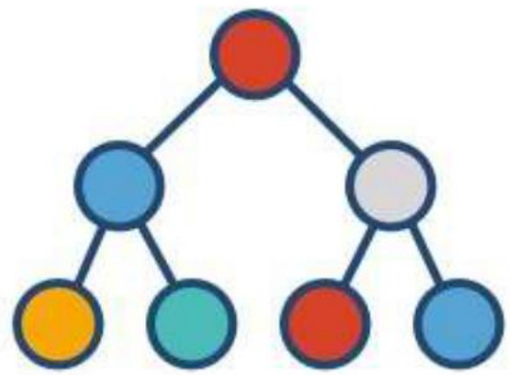
Бинарное дерево



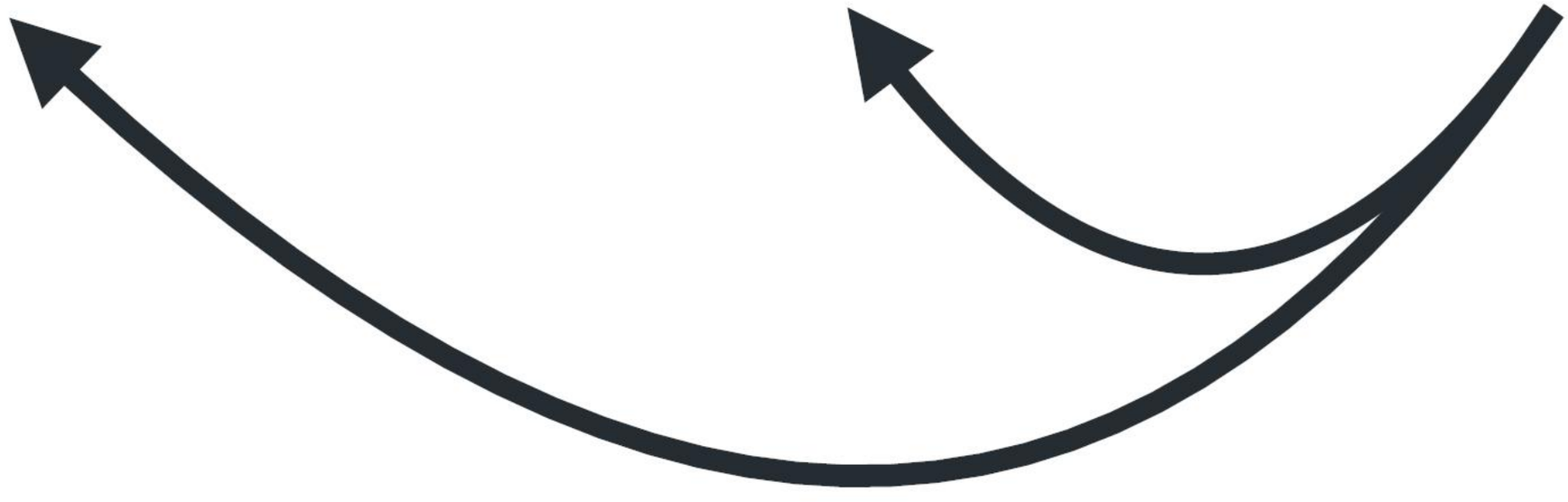
Full Binary Tree

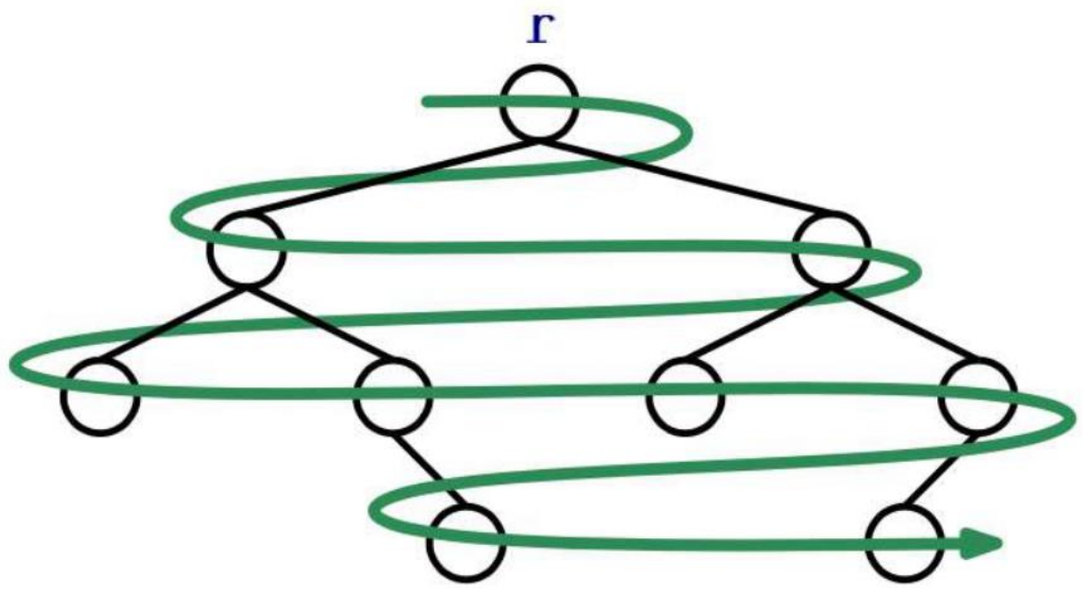


Complete Binary Tree



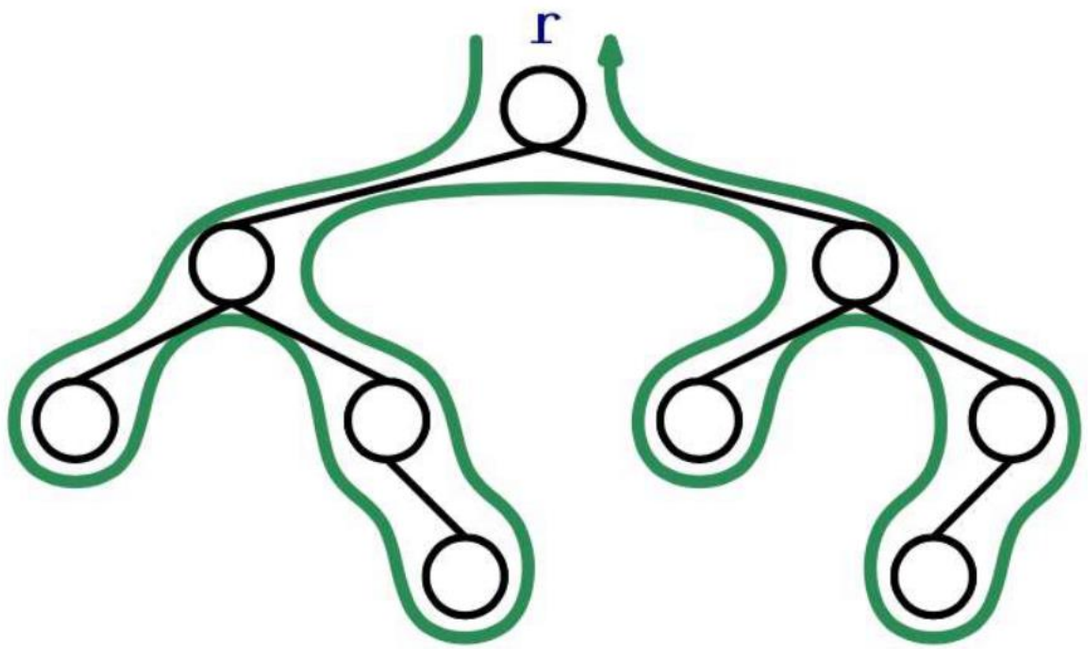
Perfect Binary Tree





Обход в ширину

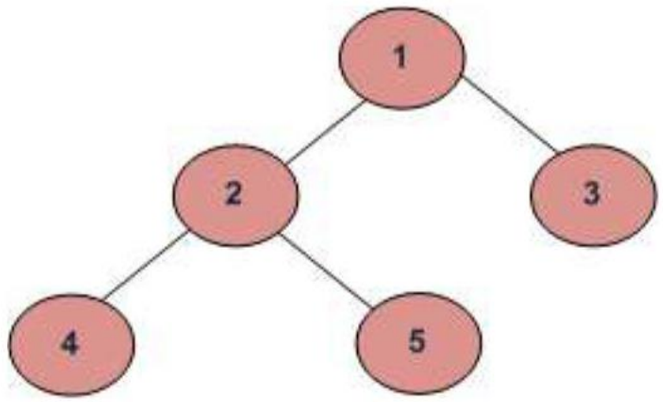
Breadth-First Traversal: BFT



Обход в глубину

Deep-First Traversal: DFT

Виды обходов в глубину



1. Префиксный (Preorder) 1 2 4 5 3
2. Симметричный (Inorder) 4 2 5 1 3
3. Постфиксный (Postorder) 4 5 2 3 1

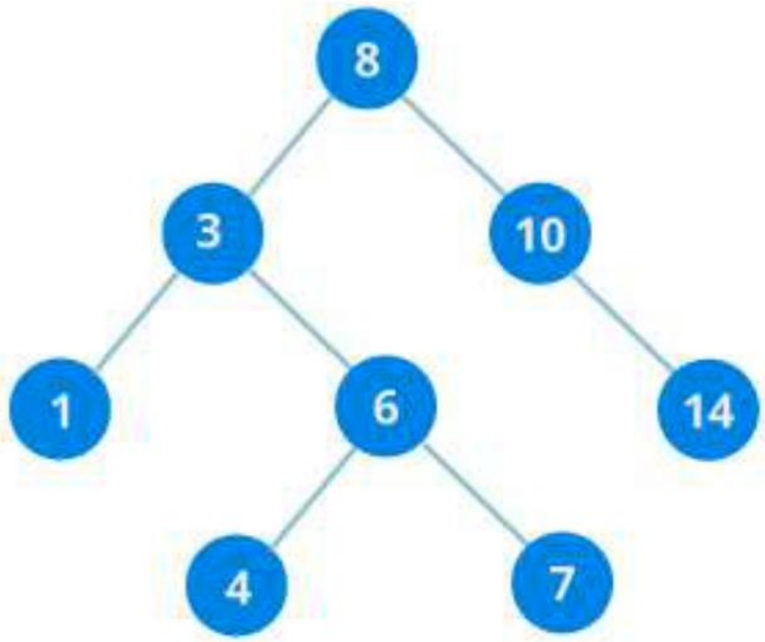
Виды обходов в глубину

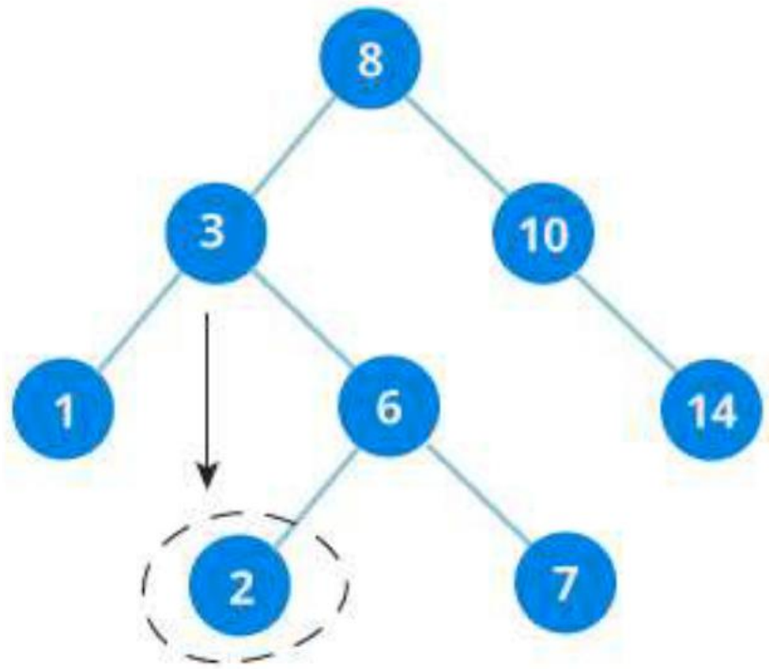
```python
def walk_preorder(node):
    if node:
        print(node.value)
        walk_preorder(node.left_child)
        walk_preorder(node.right_child)

def walk_inorder(node):
    if node:
        walk_inorder(node.left_child)
        print(node.value)
        walk_inorder(node.right_child)

def walk_postorder(node):
    if node:
        walk_postorder(node.left_child)
        walk_postorder(node.right_child)
        print(node.value)
```

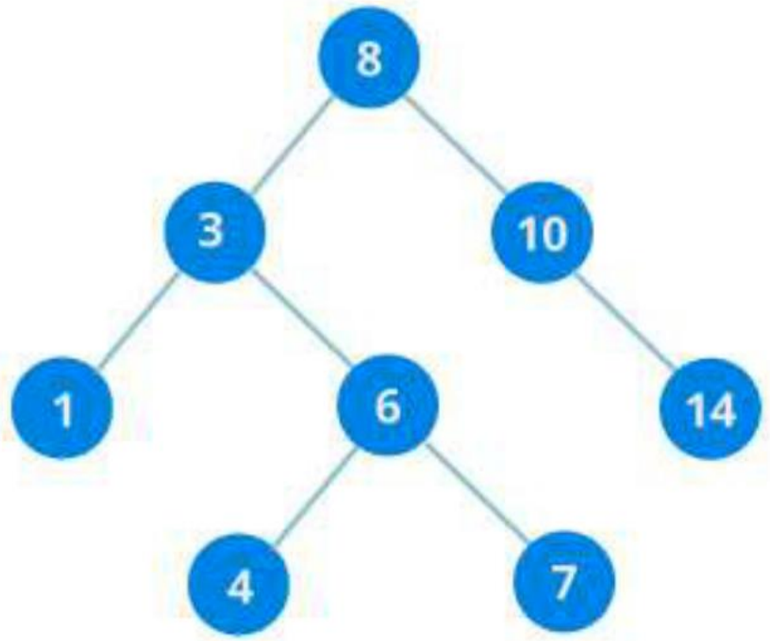
Бинарное дерево поиска

Ключи в двоичном дереве поиска хранятся с соблюдением **свойства упорядоченности** (binary-search-tree property):

Пусть x — произвольная вершина двоичного дерева поиска. Если вершина y находится в левом поддереве вершины x, то $key[y] \leqslant key[x]$. Если y находится в правом поддереве x, то $key[y] \geqslant key[x]$.


8
3
10
1
6
14
4
7


8
3
10
1
6
14
2
7

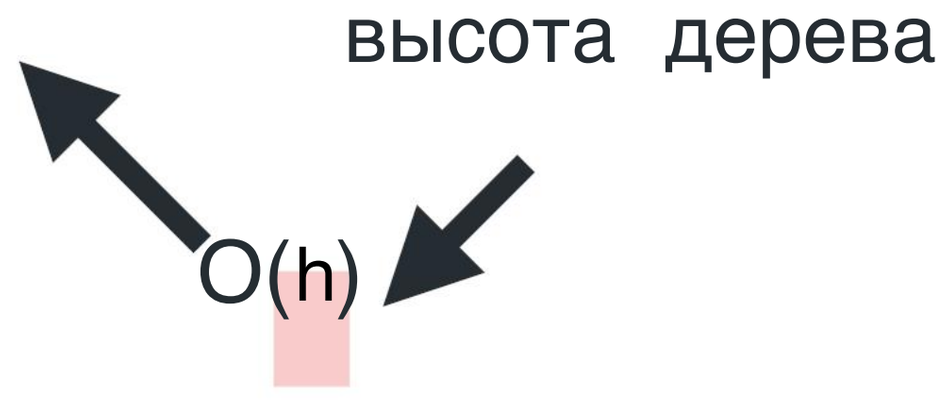


inorder-обход = неубывающий порядок

1 3 4 6 7 8 10 14

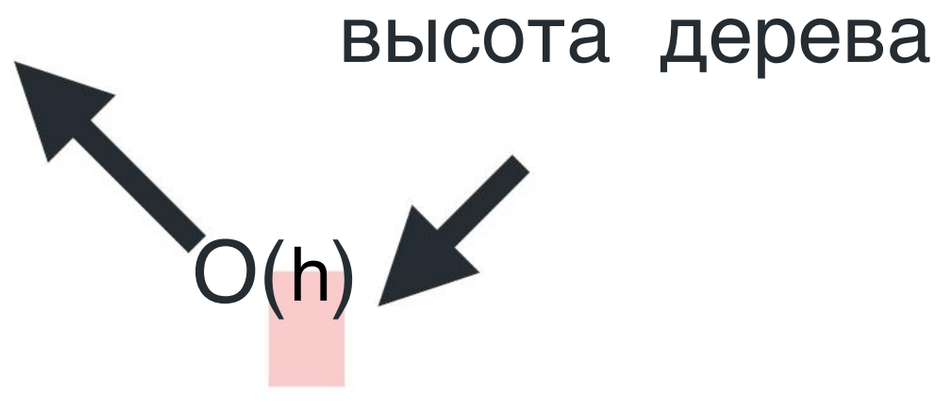
Поиск 🕵️

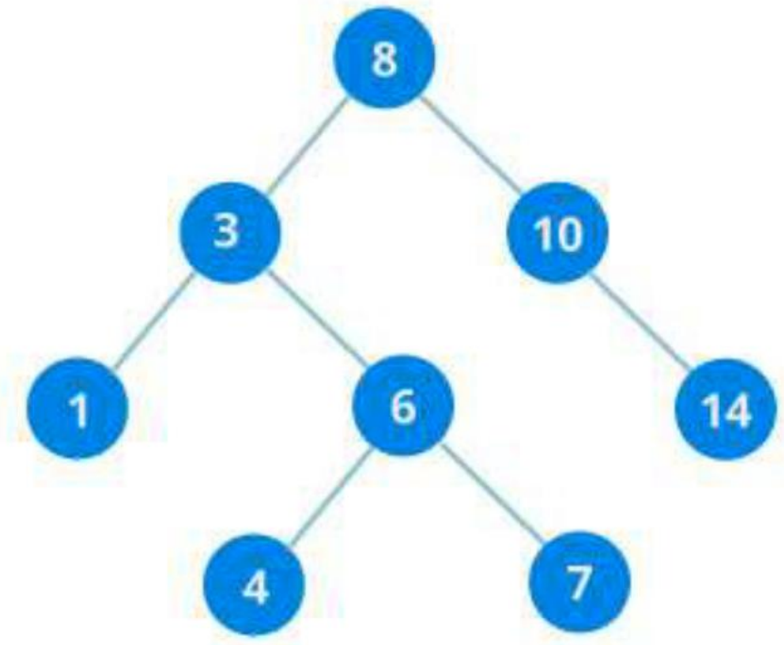
```python
def search(root, x):
    if root is None or root.value == x:
        return root
    if root.value < x:
        return search(root.right_child, x)
    return search(root.left_child, x)
```



Поиск 🕵️

```python
def search(root, x):
    while root is not None and root.value ≠ x:
        root = root.right_child if root.value < x else root.left_child
    return root
```

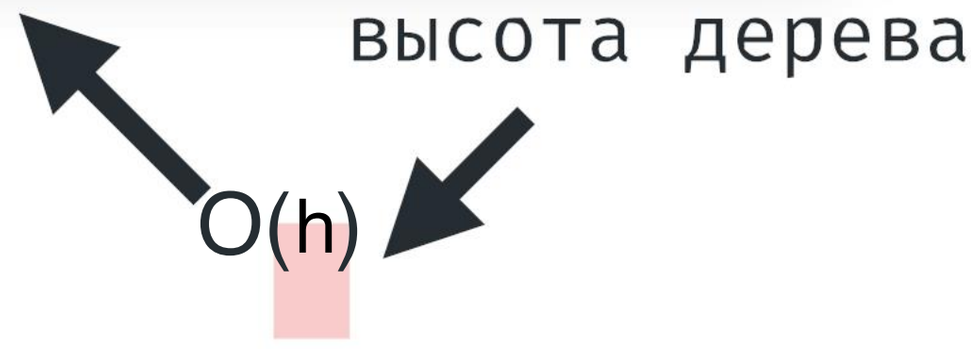




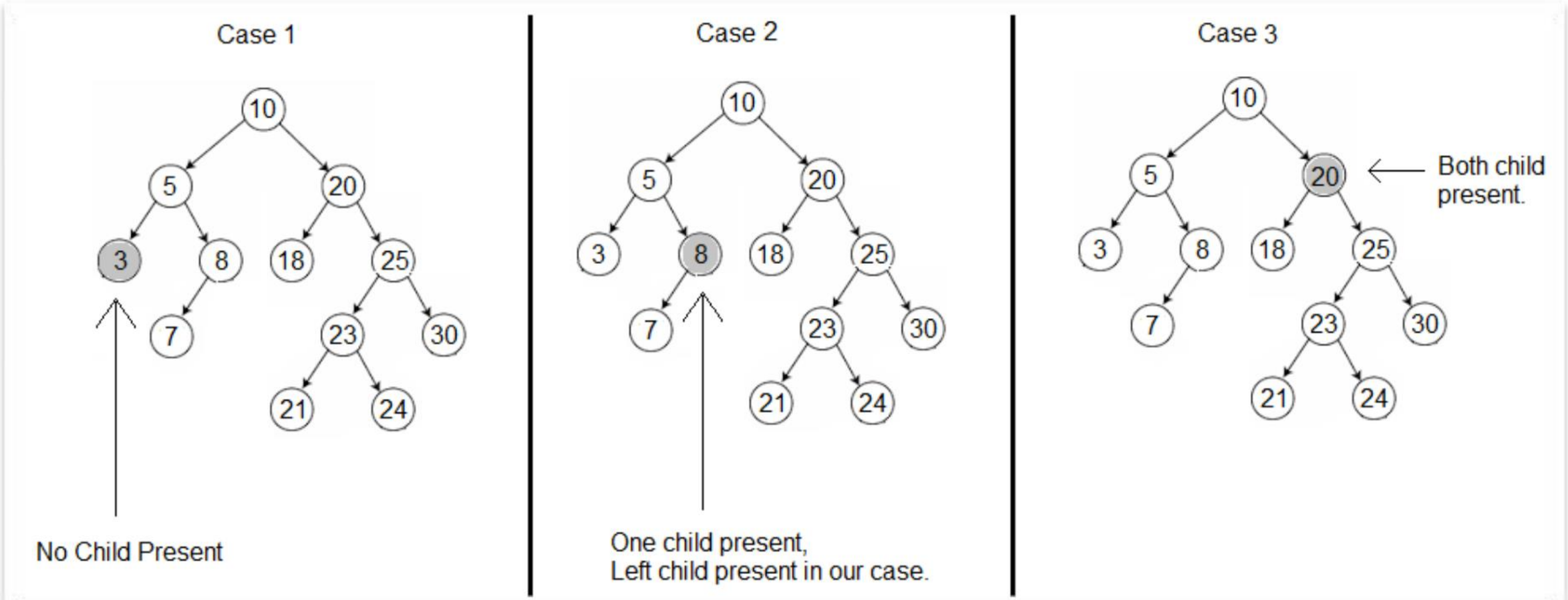
Как найти min и max?

Вставка

```python
def insert(root, x):
    if root is None:
        return Node(x)

    if root.value ⩽ x:
        root.right_child = insert(root.right_child, x)
    else:
        root.left_child = insert(root.left_child, x)
    return root
```

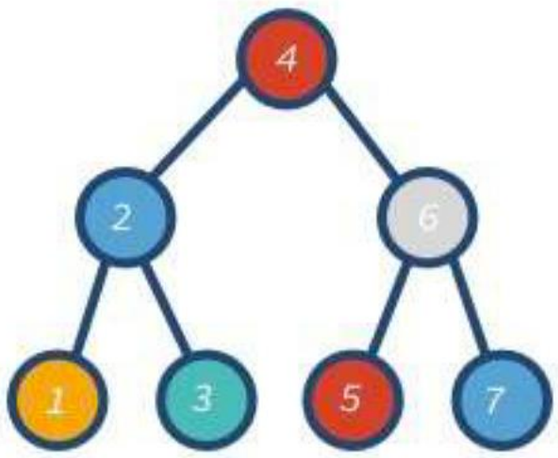


Удаление

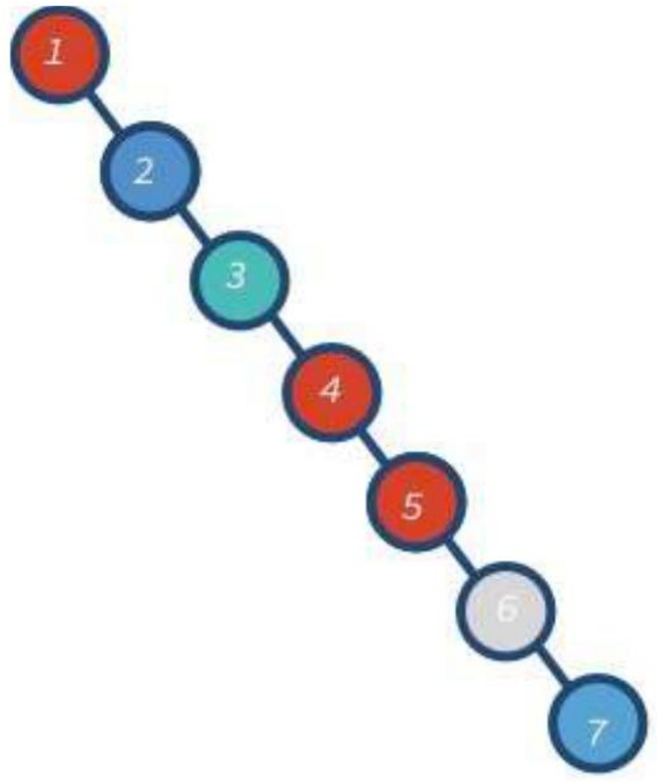

Case 1
10
5
20
3
8
18
25
7
23
30
21
24
No Child Present
Case 2
10
5
20
3
8
18
25
7
23
30
21
24
One child present,
Left child present in our case.
Case 3
10
5
20
Both child
present.
3
8
18
25
7
23
30
21
24

```python
def find_min_node(root):
    while root.left_child is not None:
        root = root.left_child
    return root

def delete(root, x):
    if root is None:  # Случай 0. Пустое дерево
        return root
    if x < root.value: # Рекурсивно опускаемся по дереву влево
        root.left_child = delete(root.left_child, x)
    elif x > root.value: # Рекурсивно опускаемся по дереву вправо
        root.right_child = delete(root.right_child, x)
    else: # Если нашли вершину ⟹ удаляем
        # Случаи 1 и 2. Нет детей или 1 ребенок
        if root.left_child is None:
            return root.right_child
        if root.right_child is None:
            return root.left_child
        # Случай 3. 2 ребенка
        min_node = find_min_node(root.right_child)
        root.value = min_node.value
        root.right_child = delete(root.right_child , min_node.value)
    return root
```

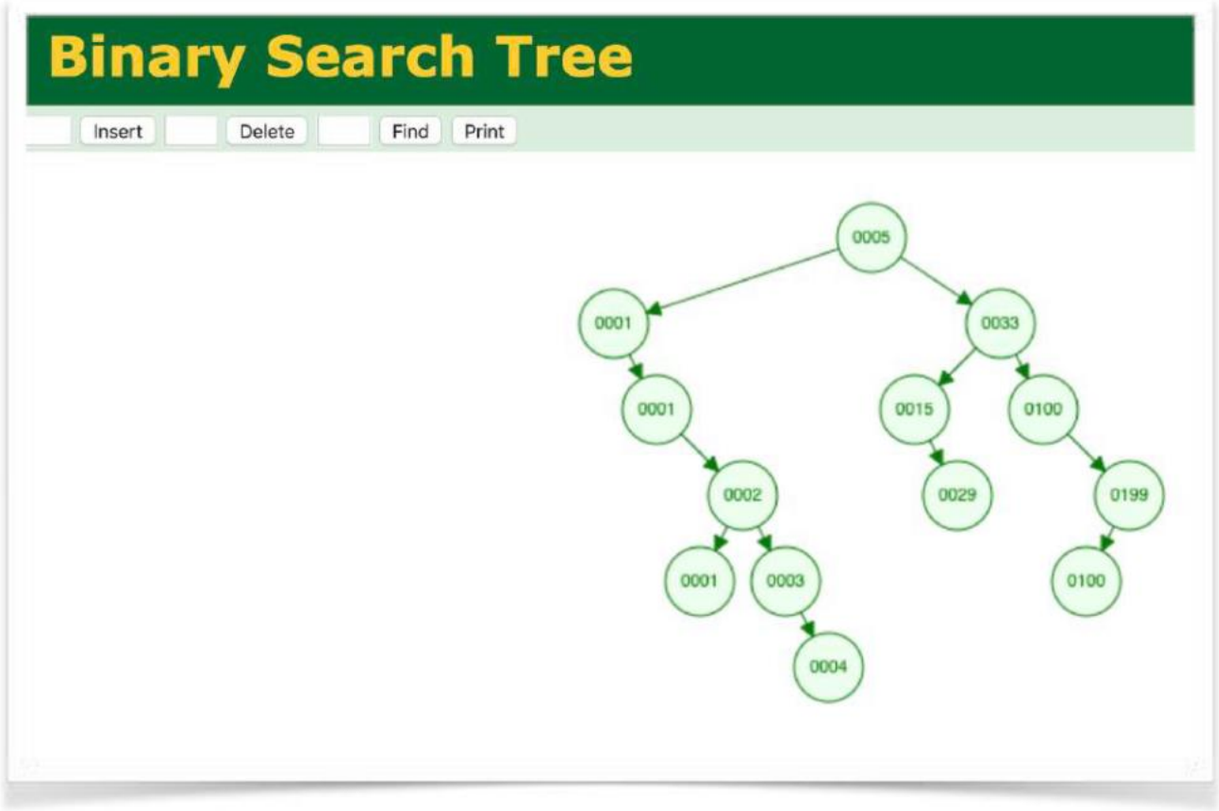
$$\lfloor \log_2 n \rfloor \leq h \leq n - 1$$



Balanced Tree



Non-Balanced Tree



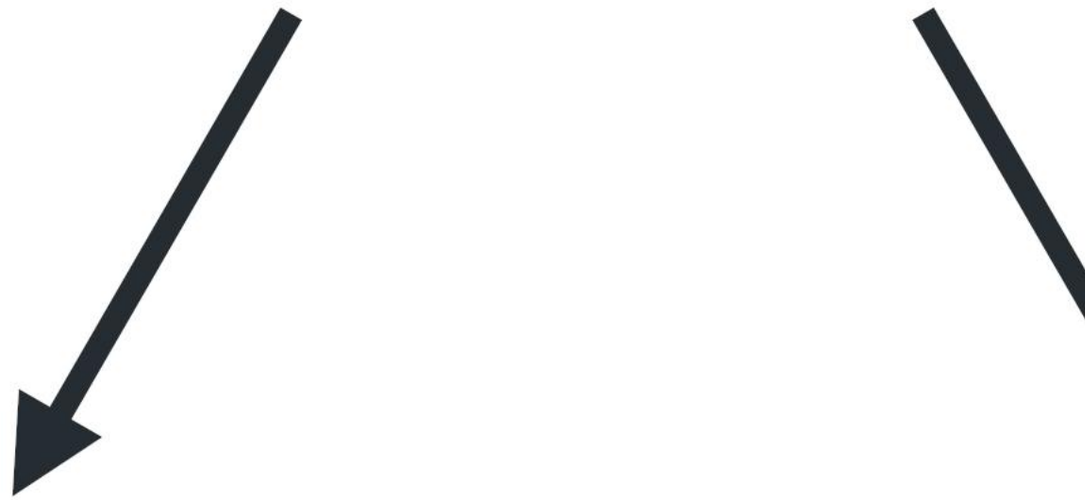
cs.usfca.edu/~galles/visualization/BST.html

Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

Сбалансированные деревья

Что делать с несбалансированным деревом?



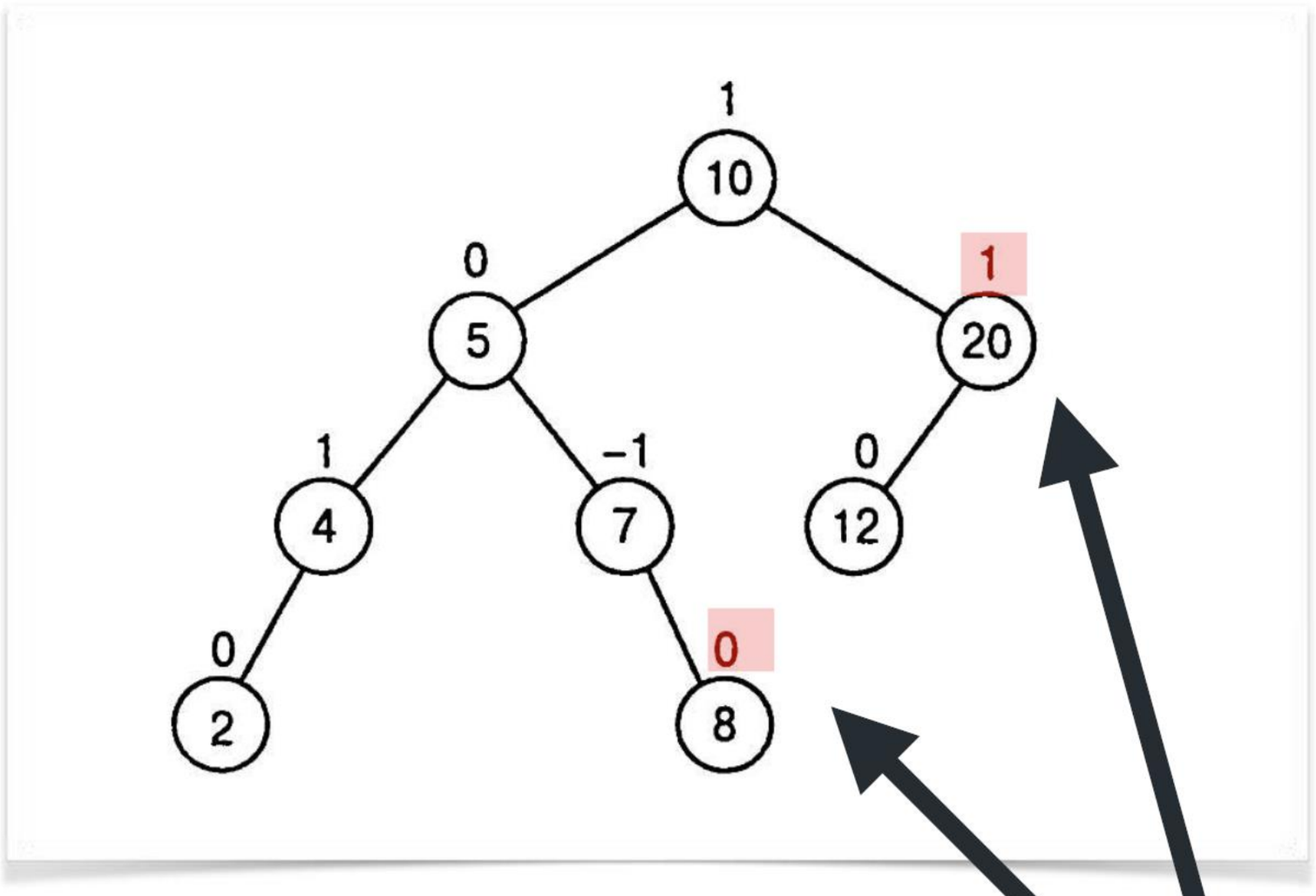
Сбалансировать (повороты дерева)

Модифицировать дерево

Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

AVL-дерево

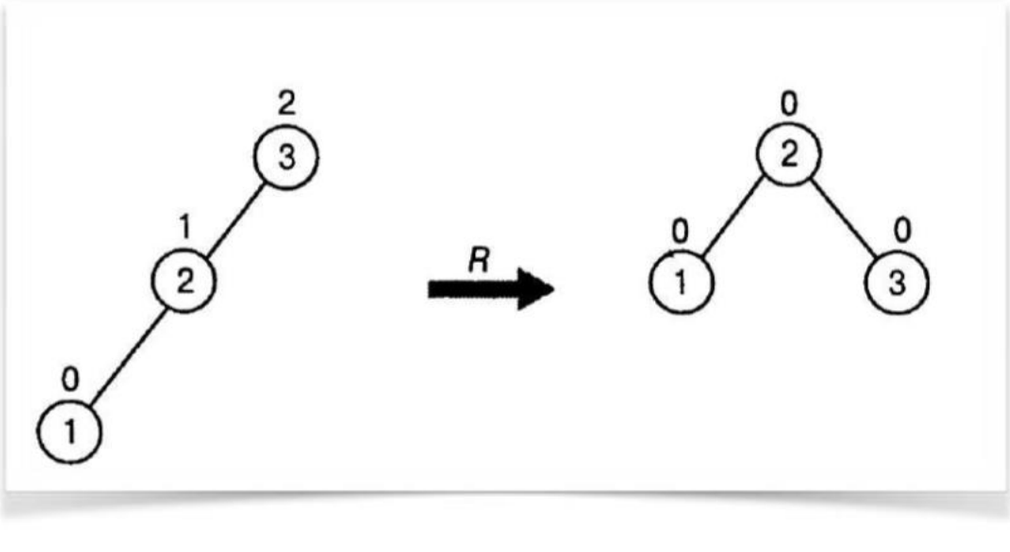


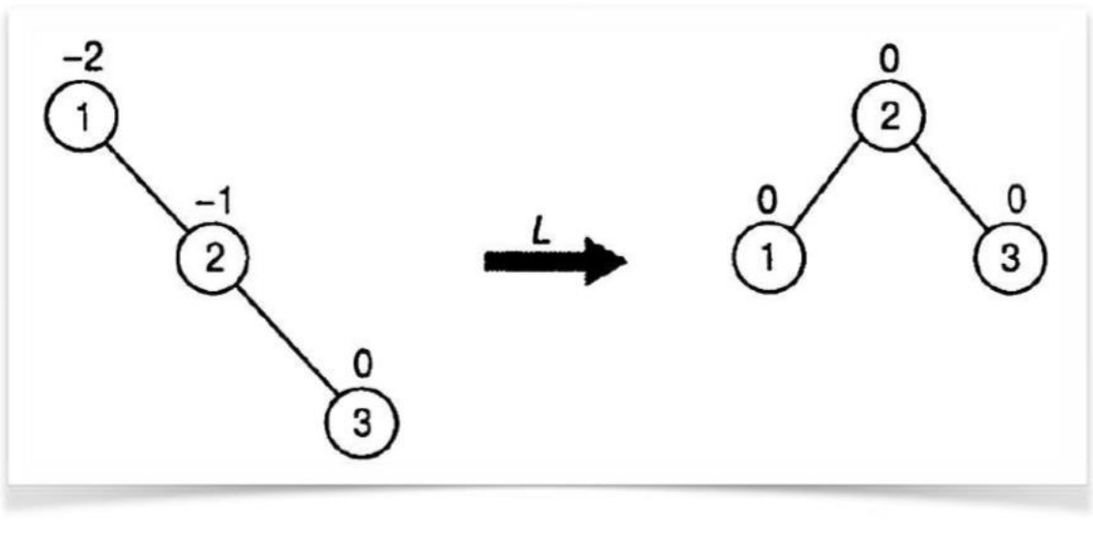
balance factor

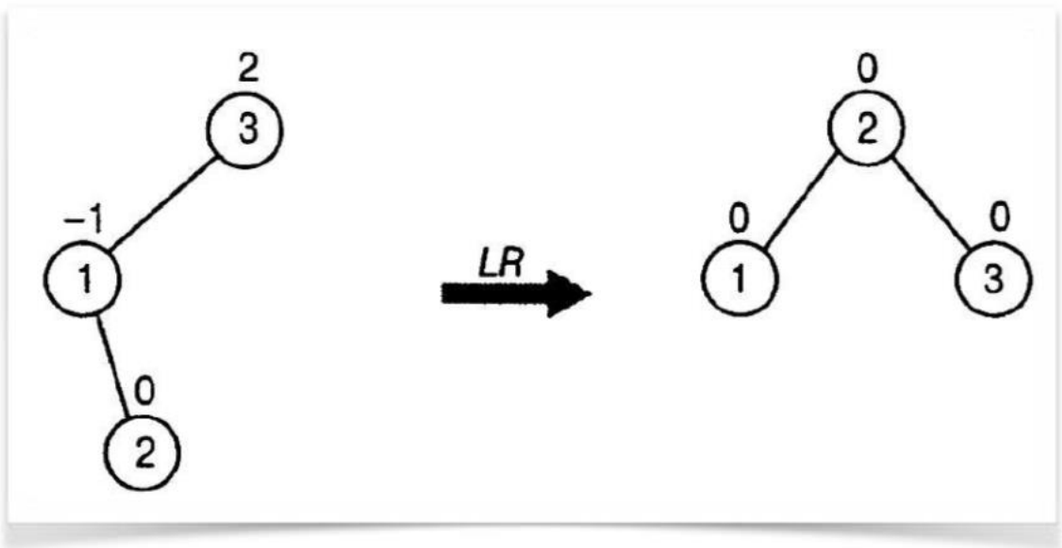
0 или ±1

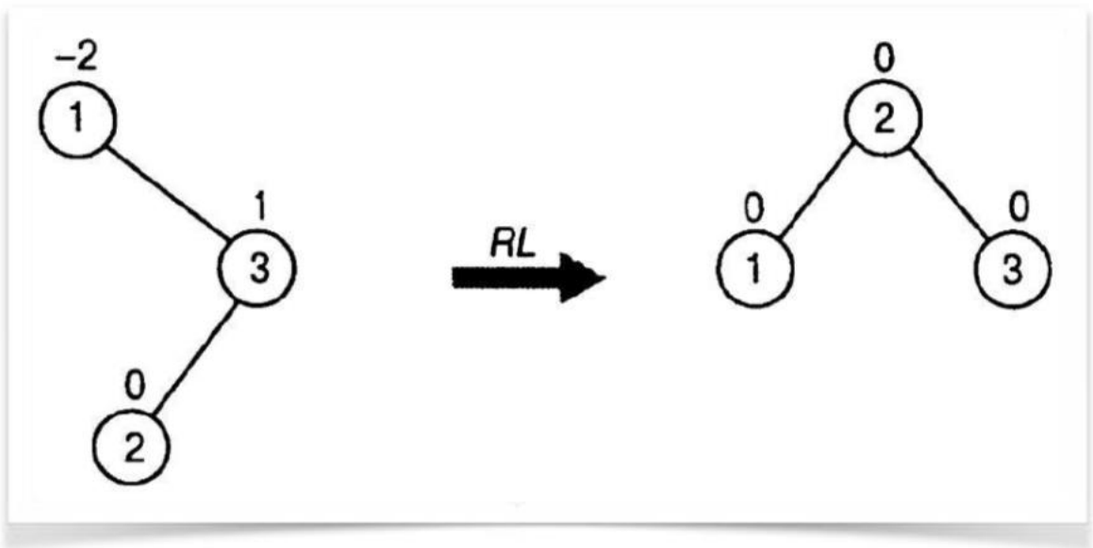
AVL-дерево

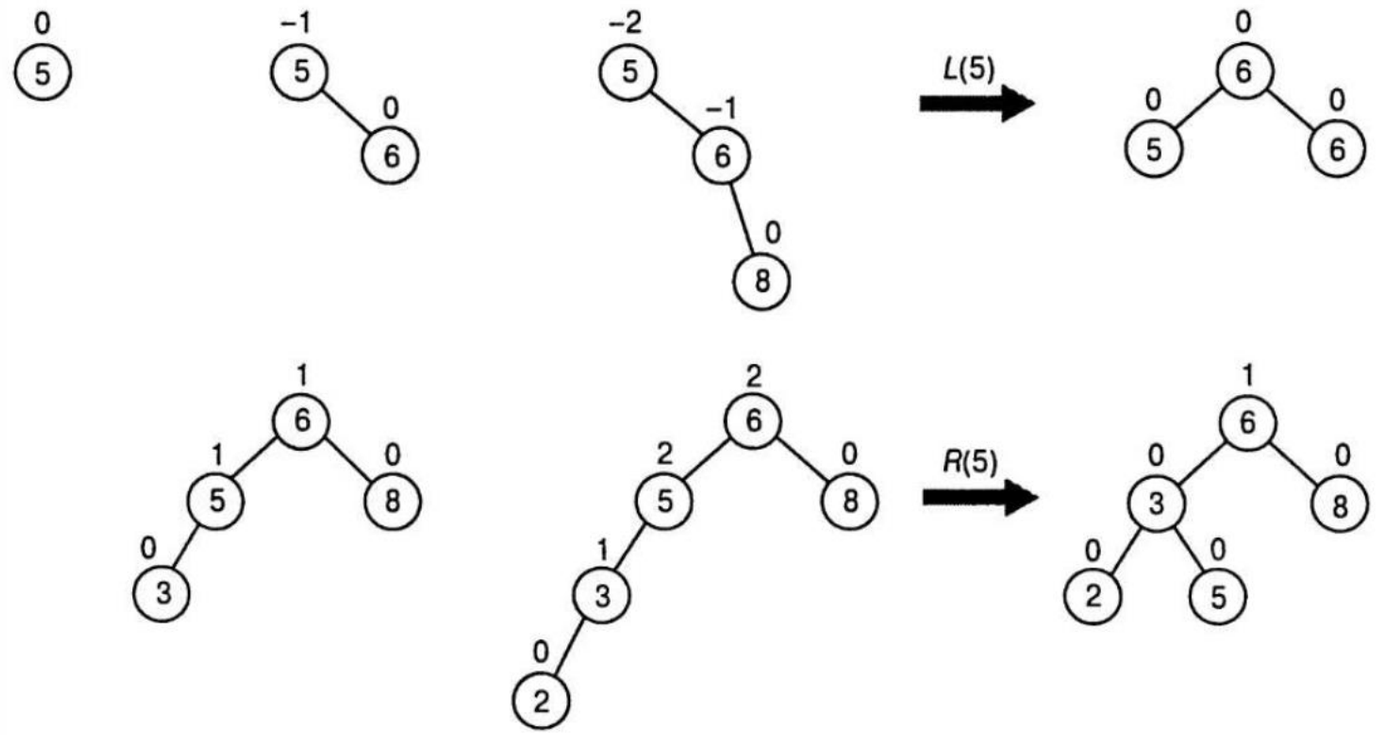
(rotations)

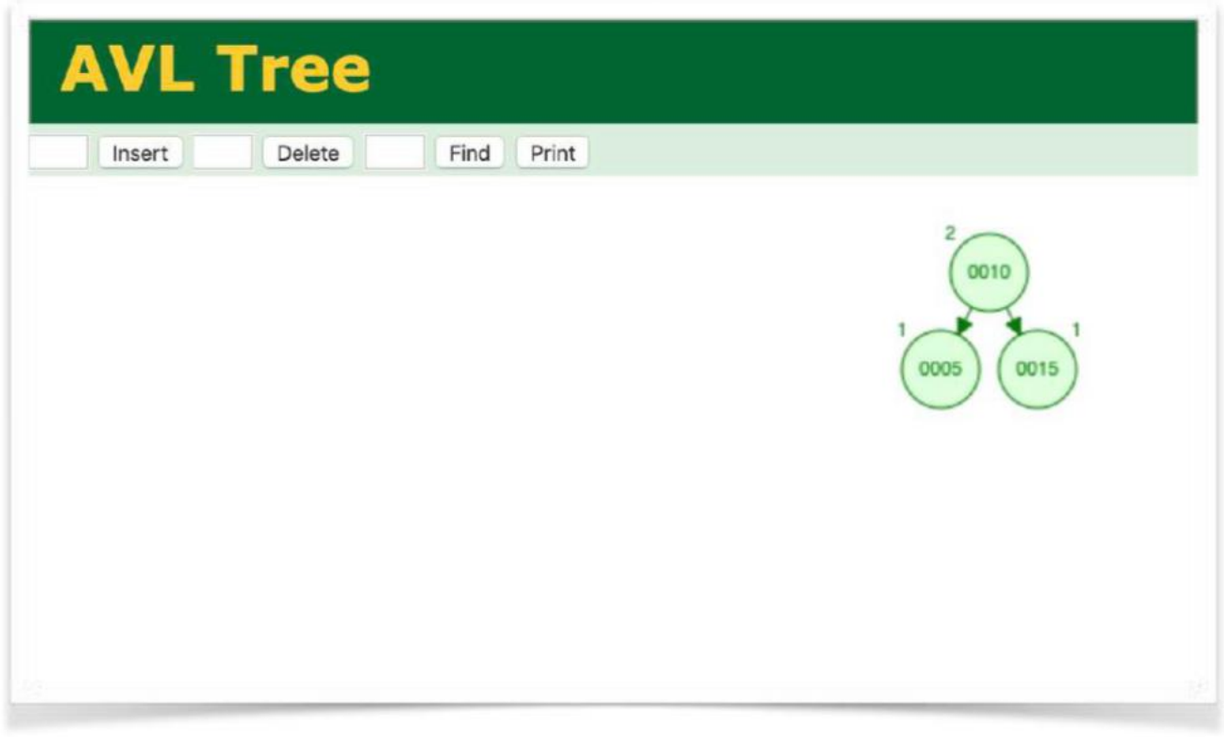









0
5
−1
5
0
6
−2
5
−1
6
0
8
L(5)
0
6
0
0
5
6
1
6
1
0
5
8
0
3
2
6
2
0
5
8
1
3
0
2
R(5)
1
6
0
0
3
8
0
2
0
5
2
6
−1
3
0
8
0
2
1
5
0
4
LR(6)
5
0
−1
3
6
0
0
0
2
4
8
1
5
0
−2
3
6
0
0
1
2
4
8
0
7
RL(6)
0
5
0
0
3
7
0
0
0
0
2
4
6
8

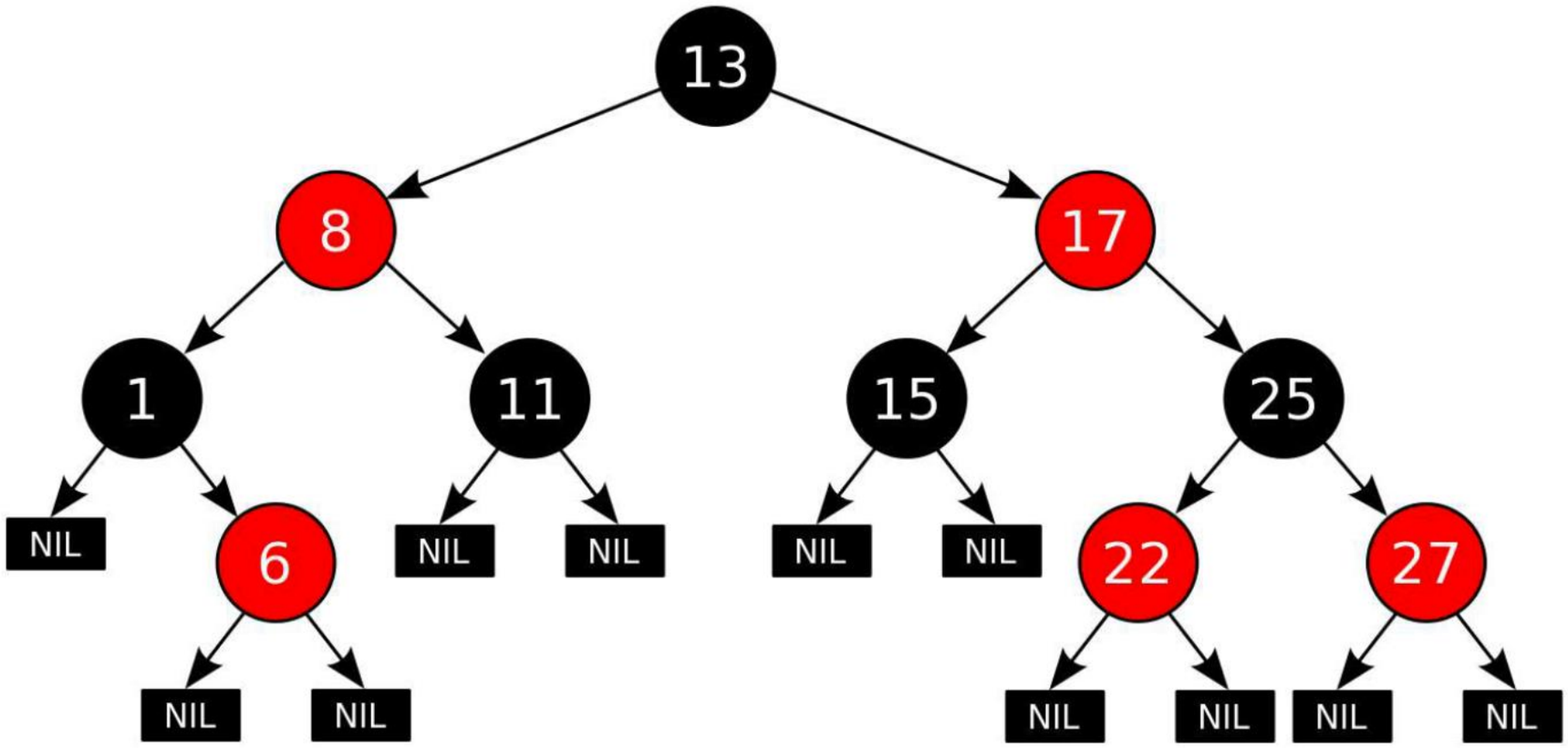


cs.usfca.edu/~galles/visualization/AVLtree.html

Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

Красно-черное дерево

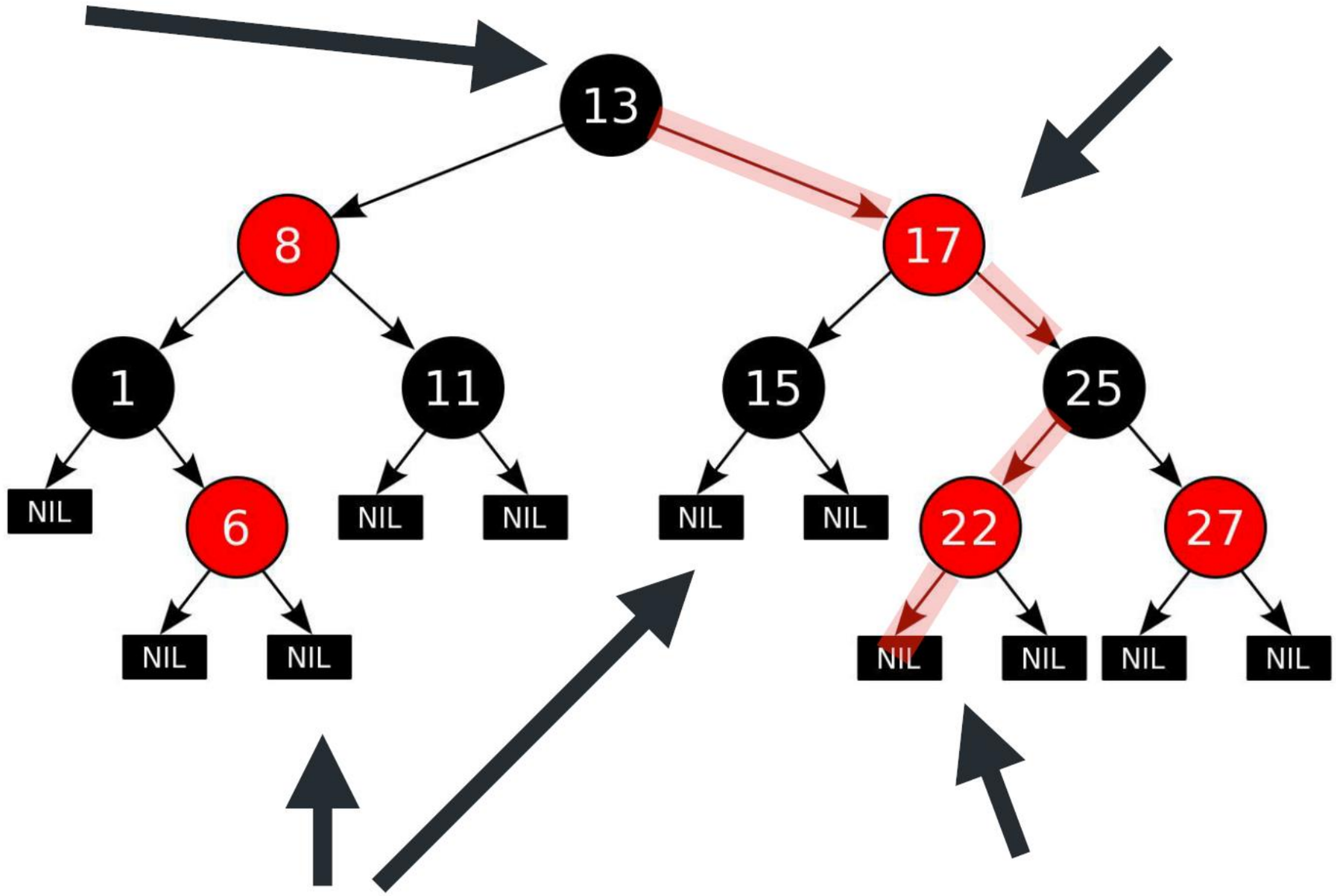


$$h \leq 2 \log(n + 1)$$

Красно-черное дерево

корень
черный

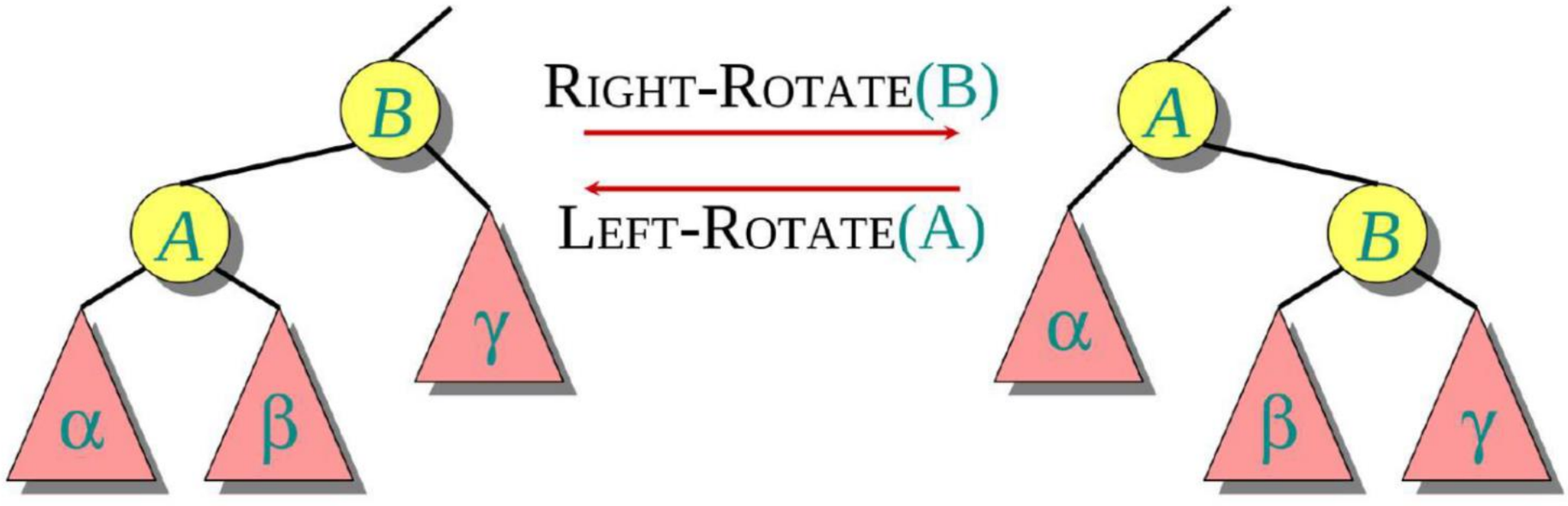
у красного
родителя
черные дети

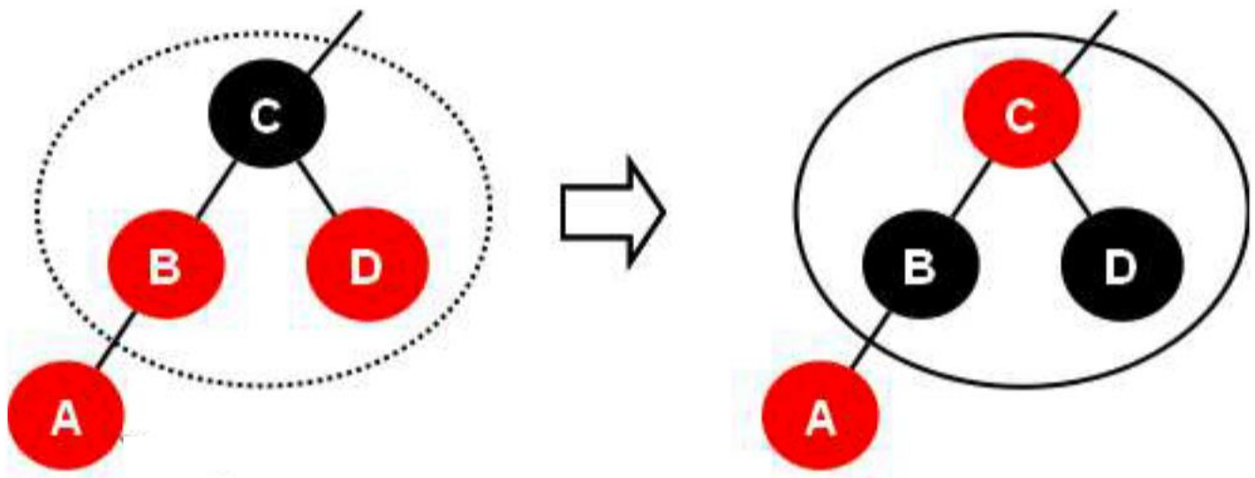


листья
черные

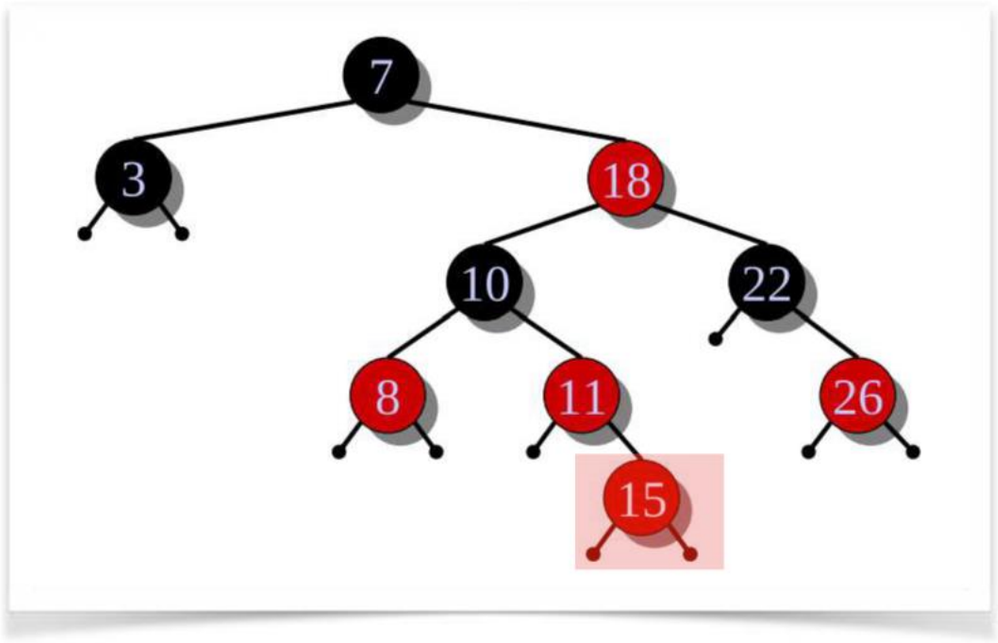
от корня к листьям
одинаковое количество
черных вершин

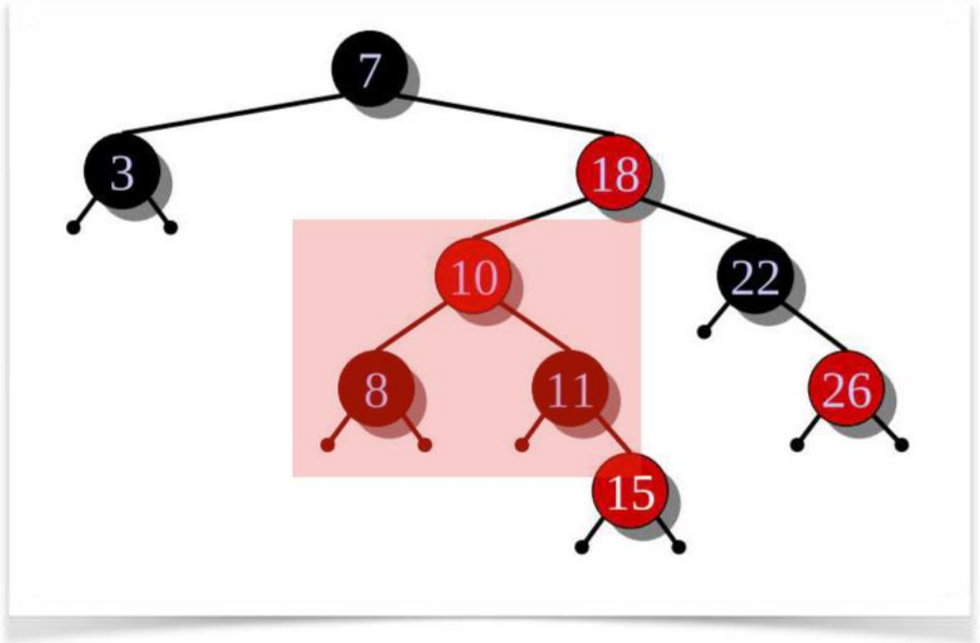
Красно-черное дерево

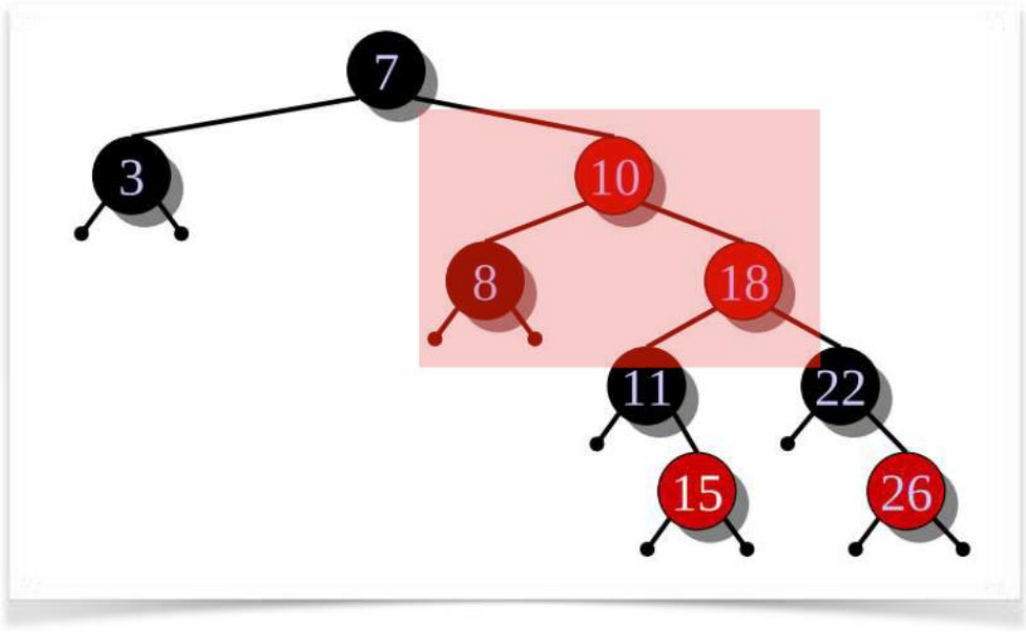


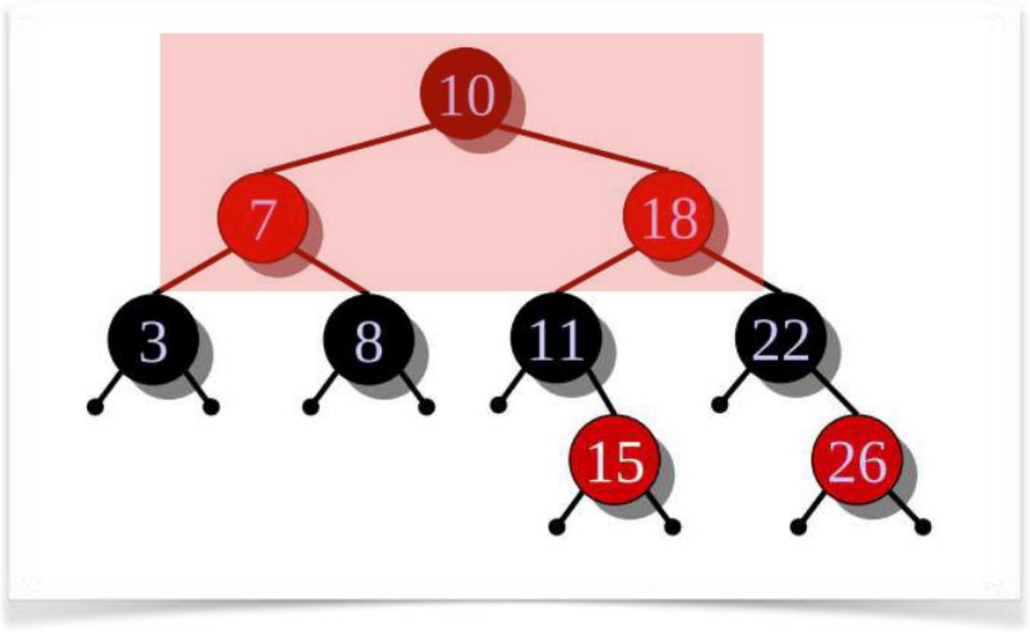


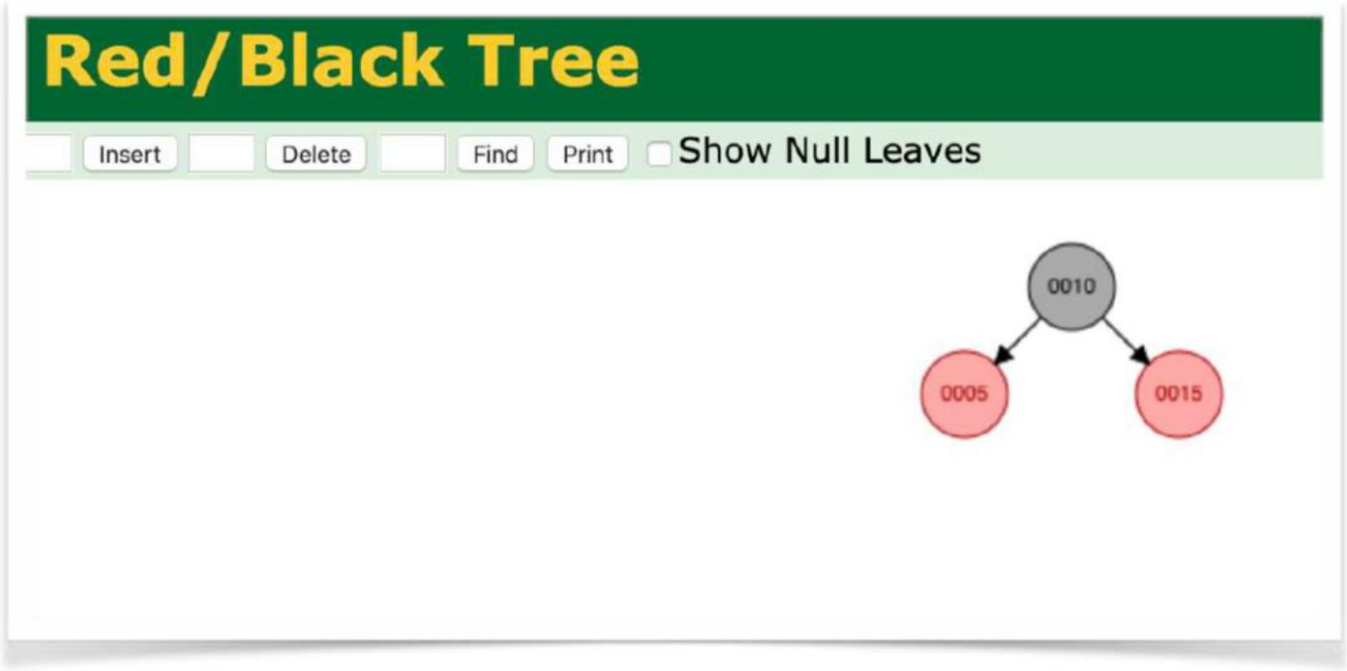
Красно-черное дерево









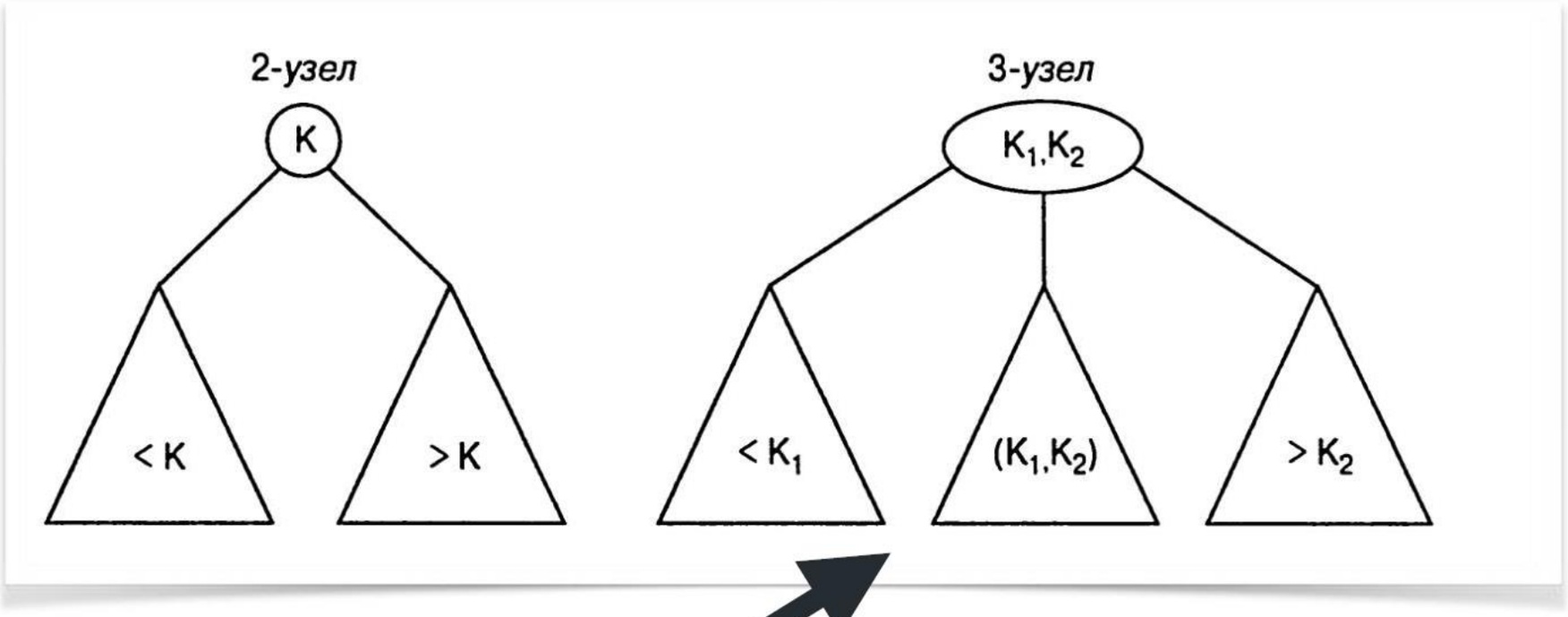


cs.usfca.edu/~galles/visualization/RedBlack.html

Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

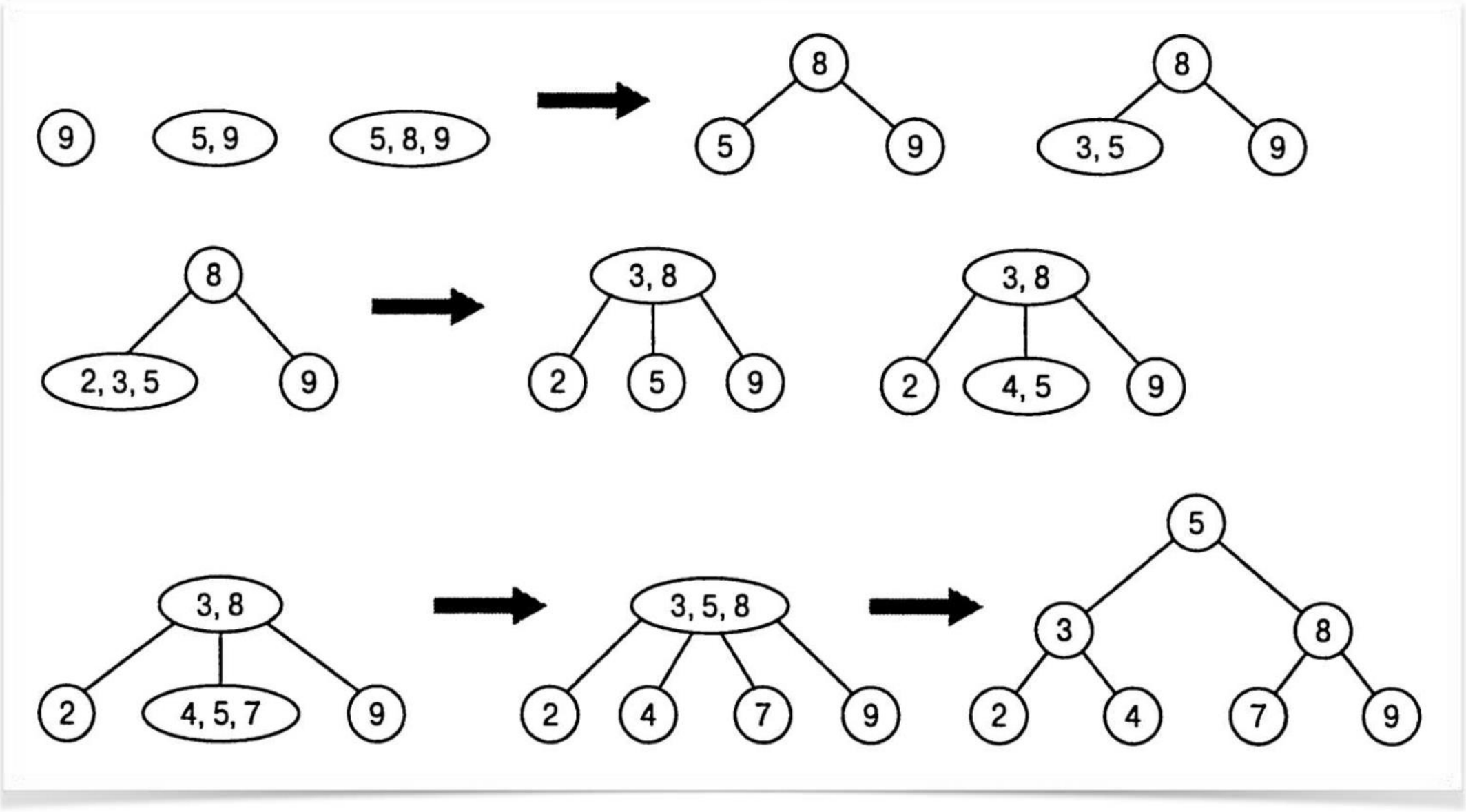
2-3—дерево



поддеревья

одной высоты

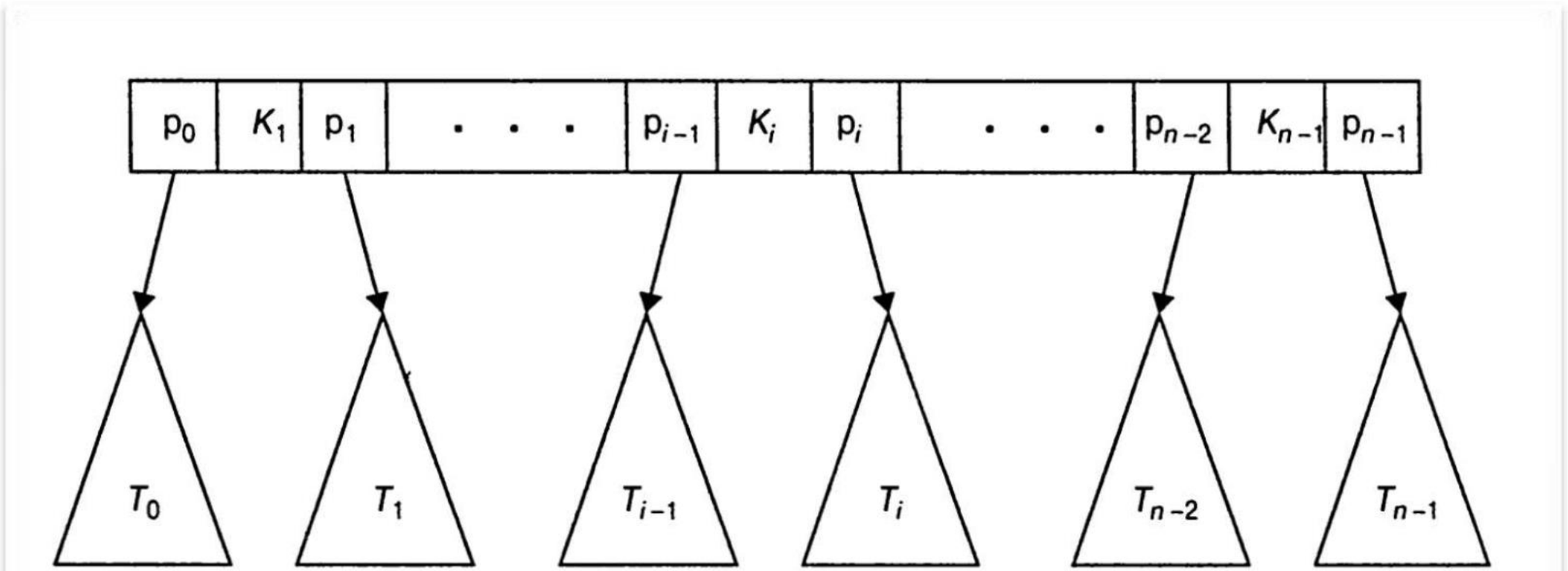
2-3—дерево


9
5, 9
5, 8, 9
8
5
9
8
3, 5
9
8
2, 3, 5
9
3, 8
2
5
9
3, 8
2
4, 5
9
3, 8
2
4, 5, 7
9
3, 5, 8
2
4
7
9
5
3
8
2
4
7
9

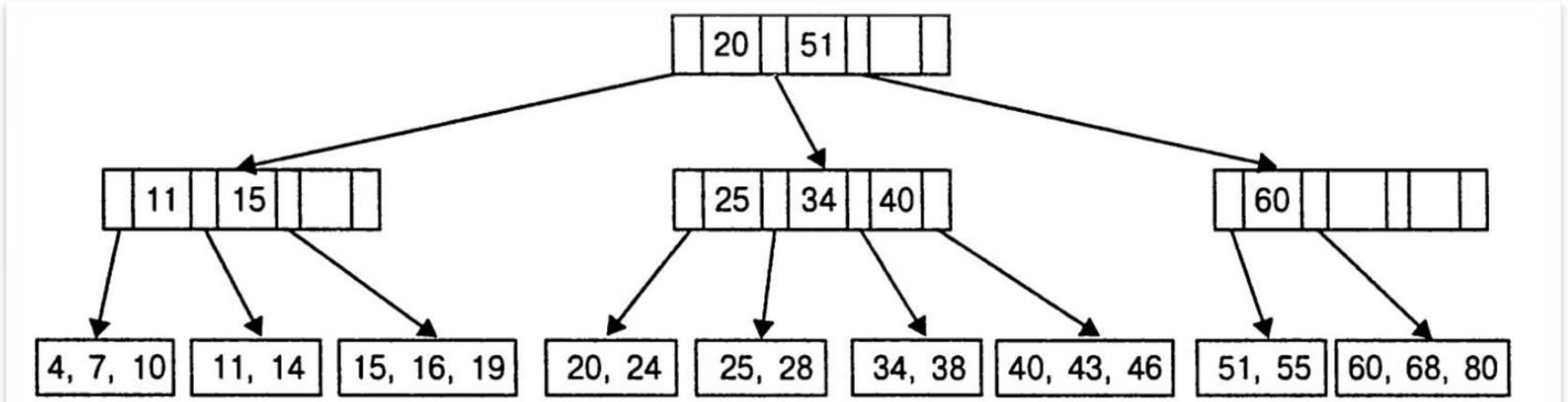
обобщение

2-3 — дерева

B — дерево

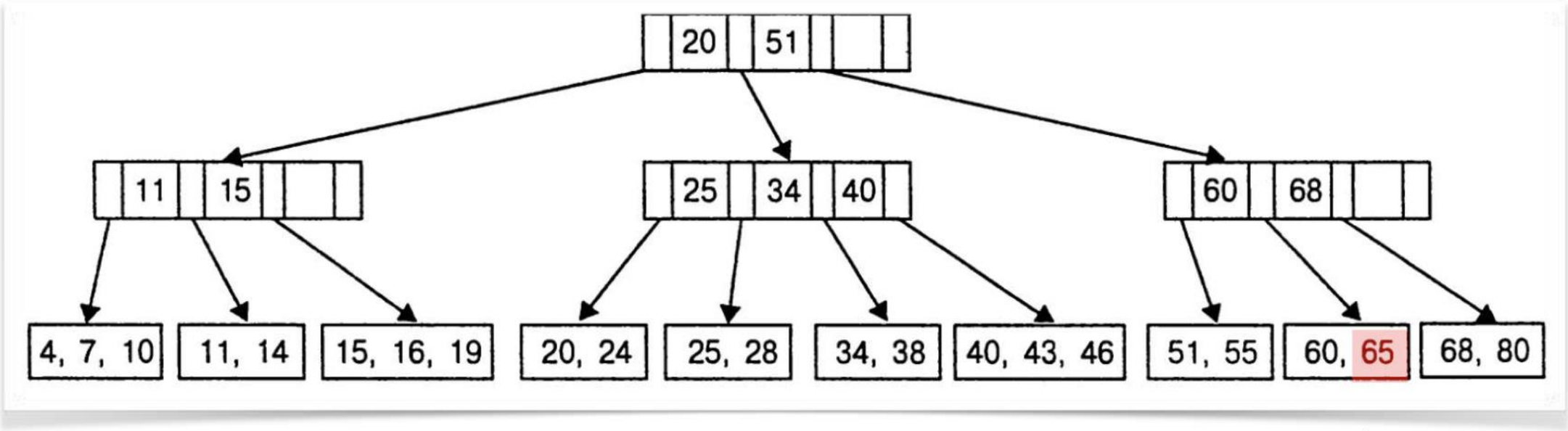


B—дерево



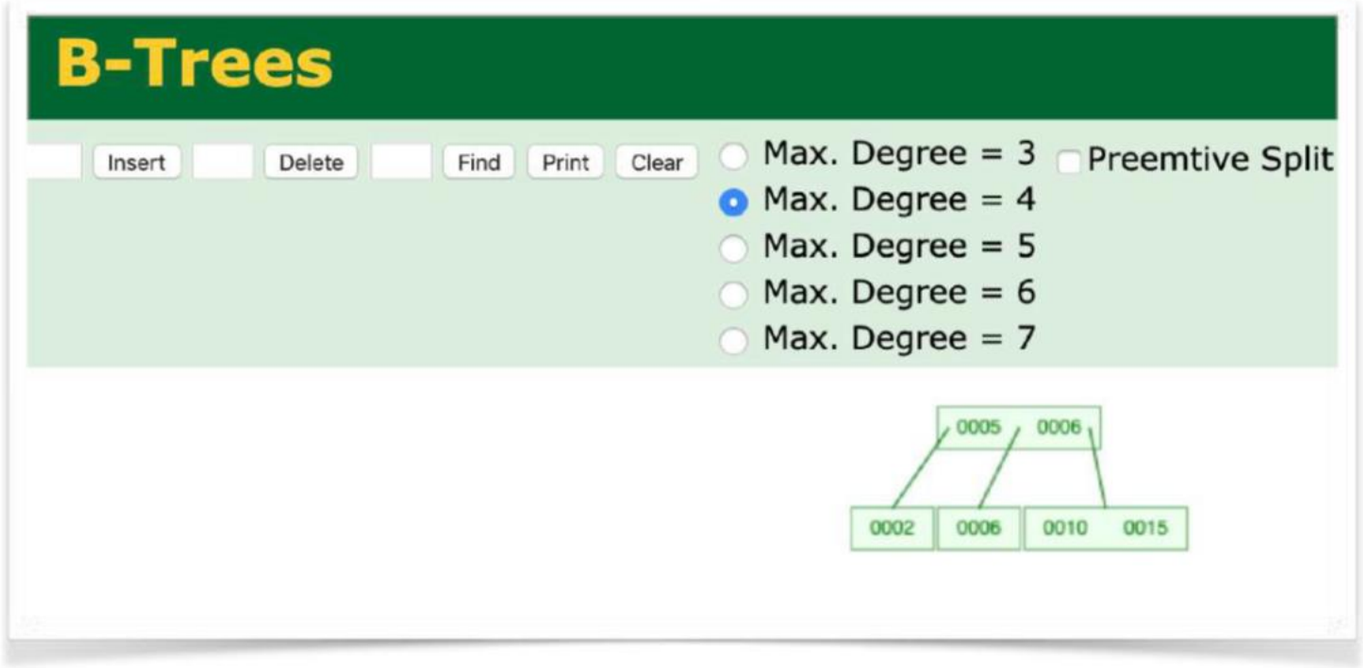
4-го порядка

B—дерево



пример

вставки 65



cs.usfca.edu/~galles/visualization/BTree.html