

ИНФОРМАТИКА



```
~/polytech $ date
```

```
осень 2019
```

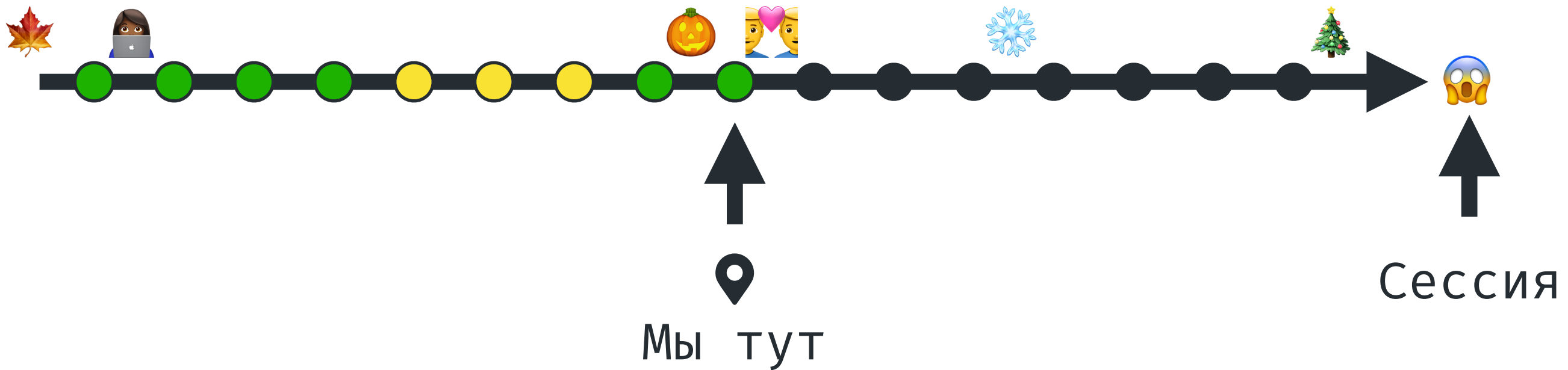
```
~/polytech $ name
```

```
константин кориков
```

```
~/polytech $ topic
```

```
атд: дерево
```

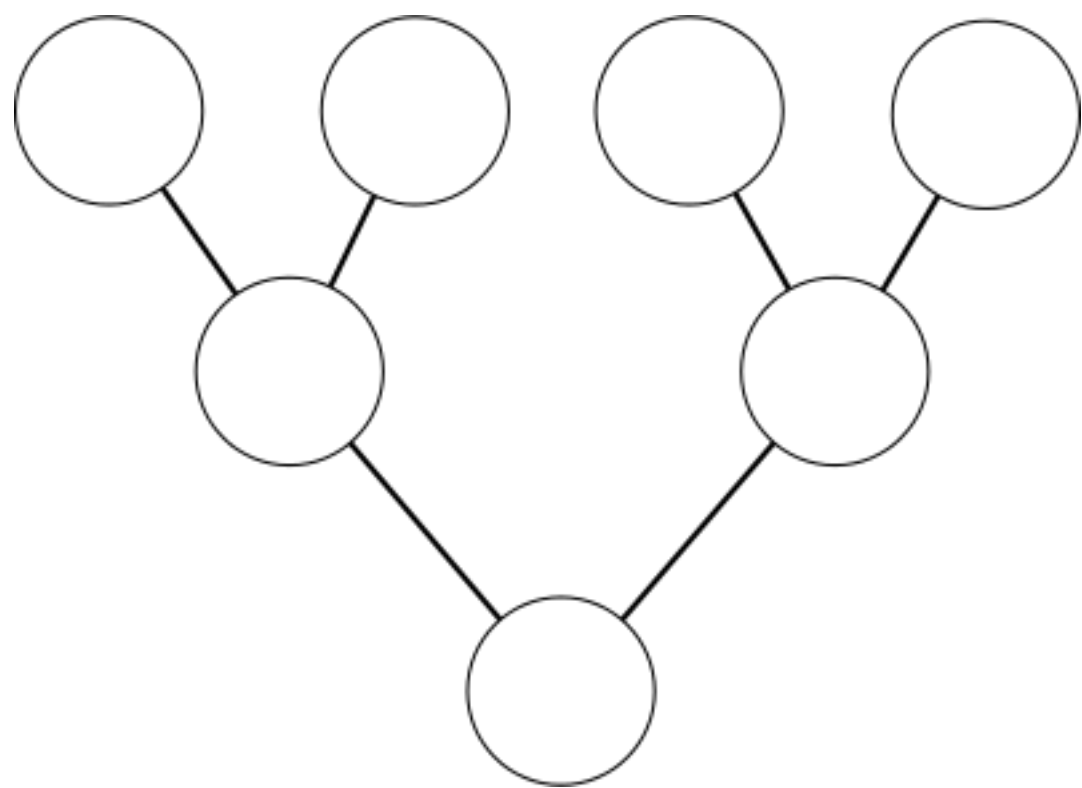
ГДЕ МЫ?

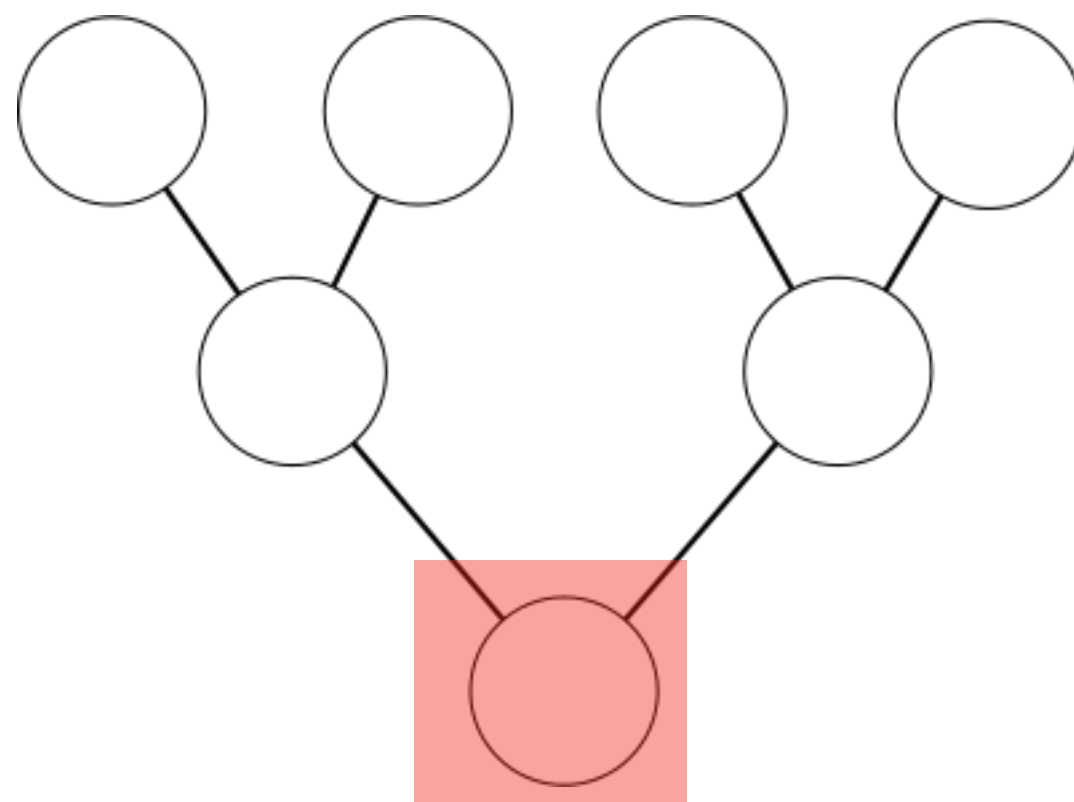


О ЧЁМ ПОГОВОРИМ?

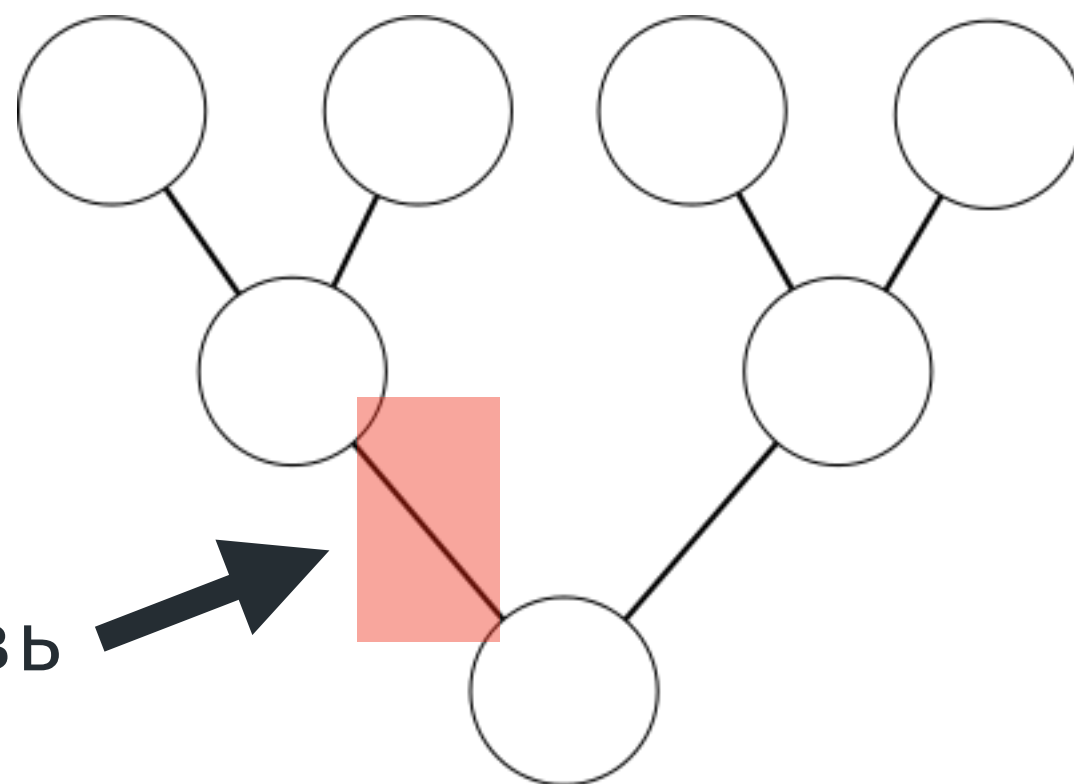
1. АД дерево
2. Реализация деревьев
3. Бинарное дерево
4. Бинарное дерево поиска
5. Сбалансированные деревья

АТД дерево



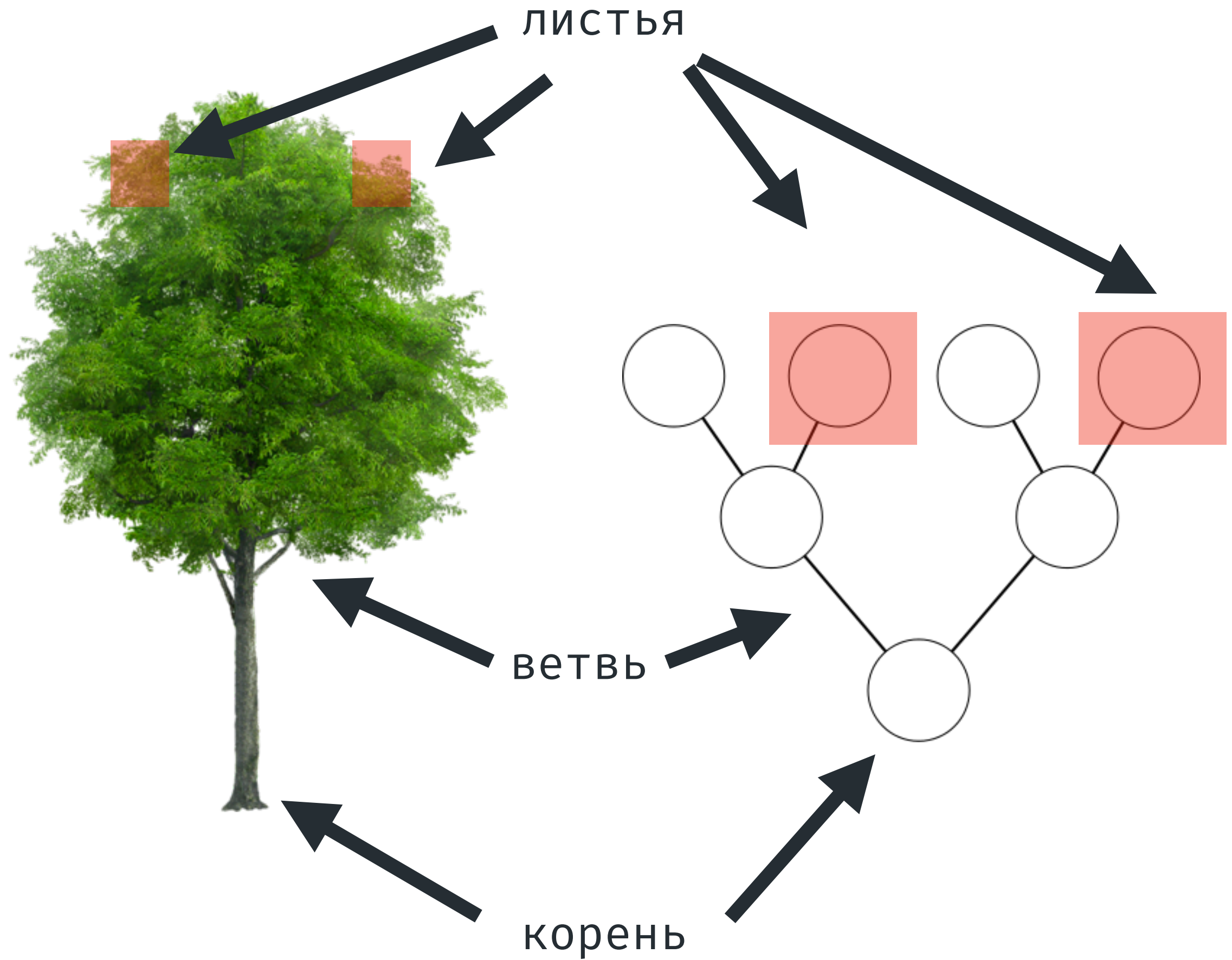


корень



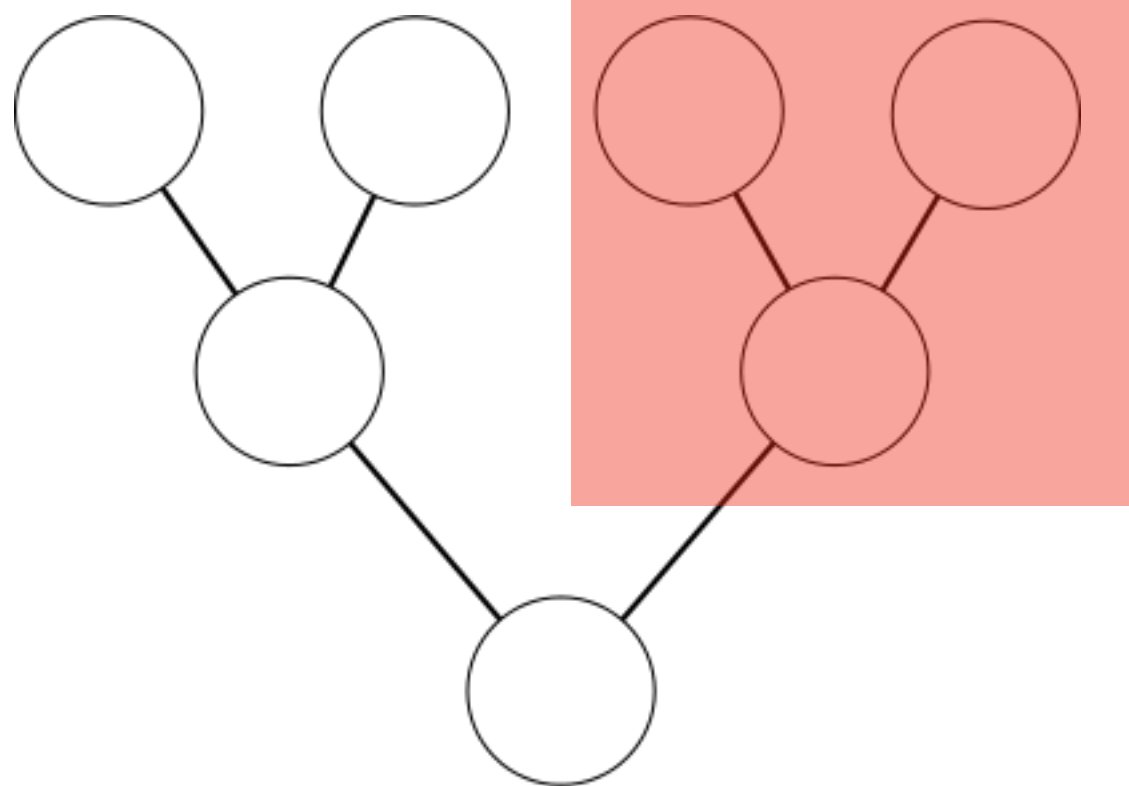
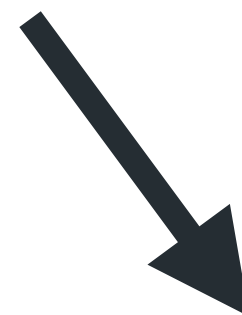
ветвь

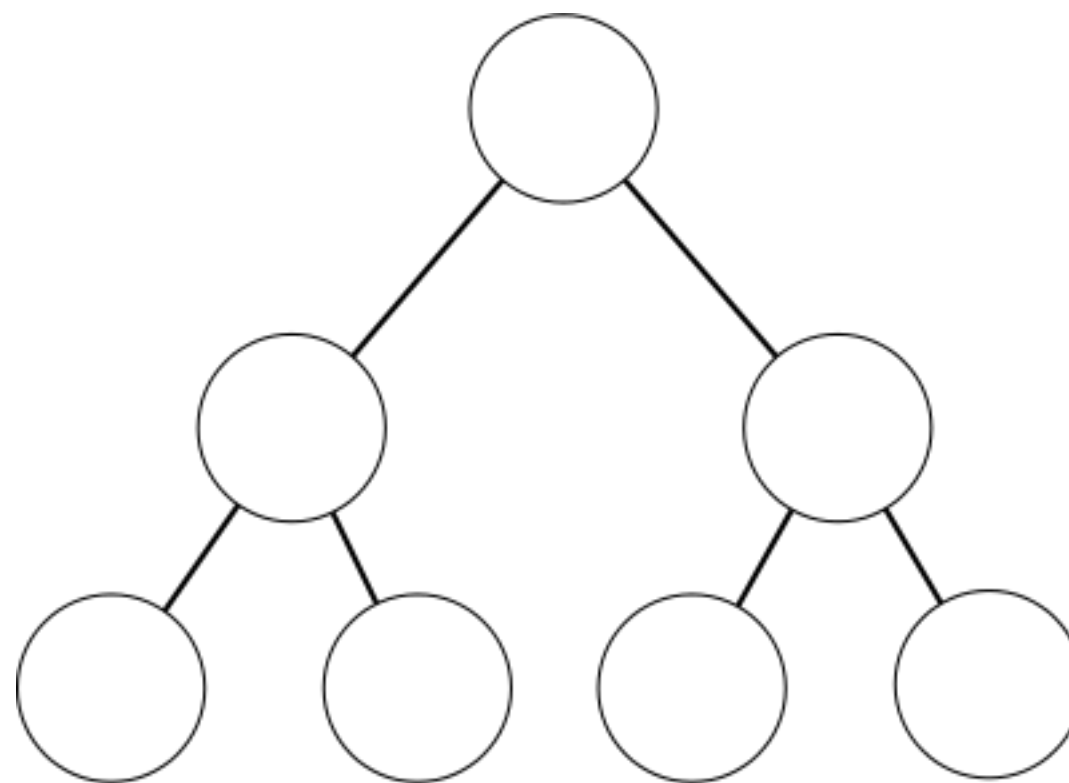
корень



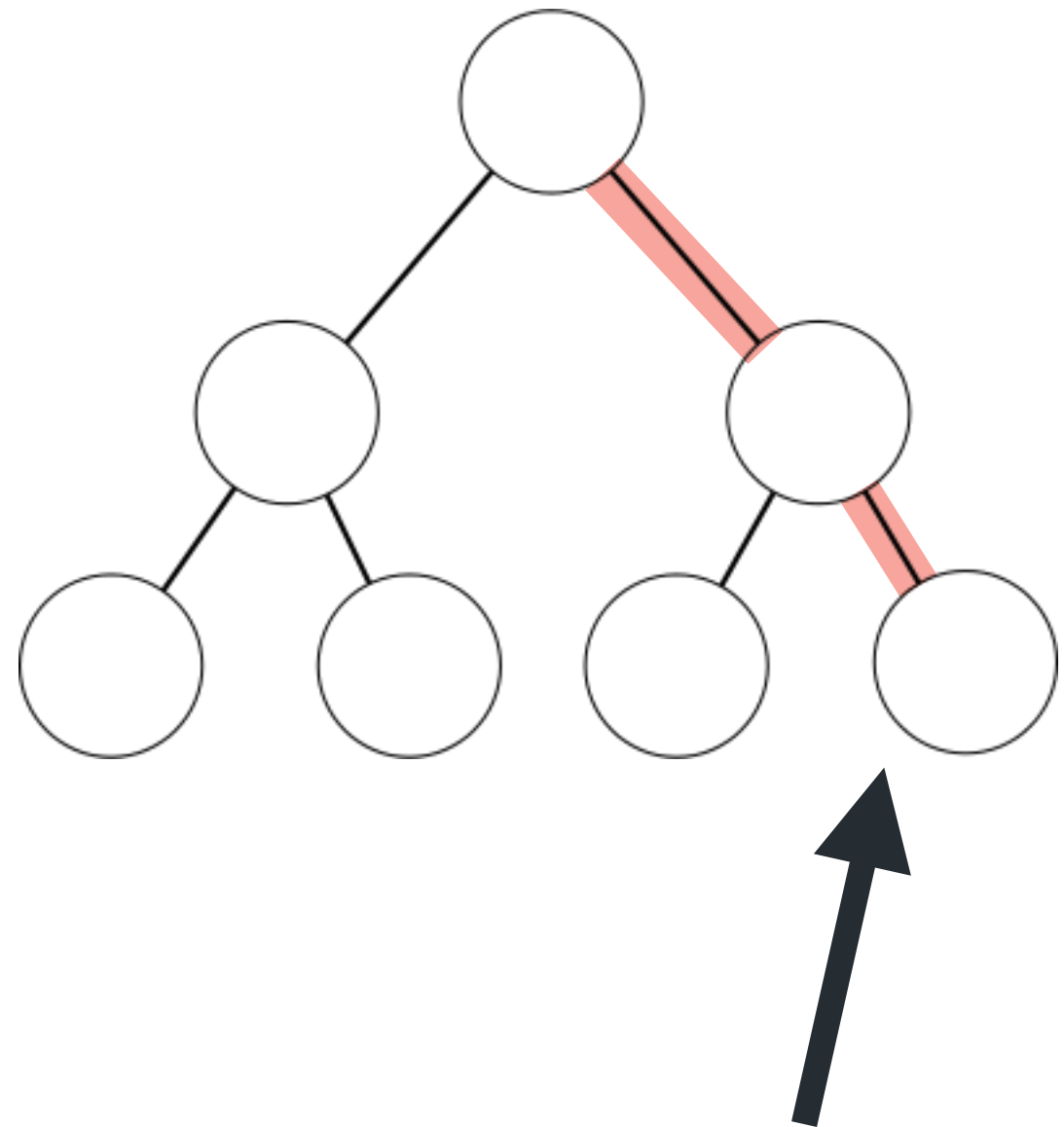


поддерево

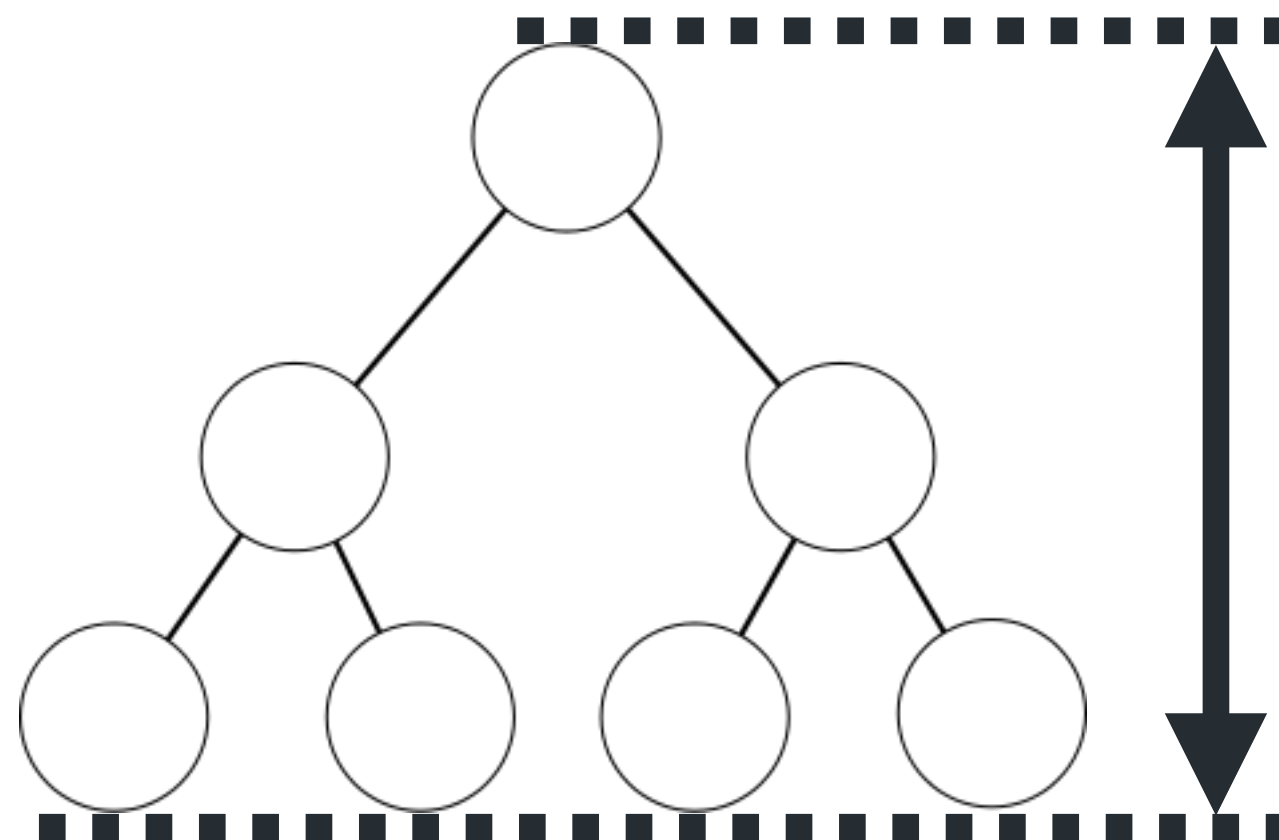




дерево (корневое дерево)



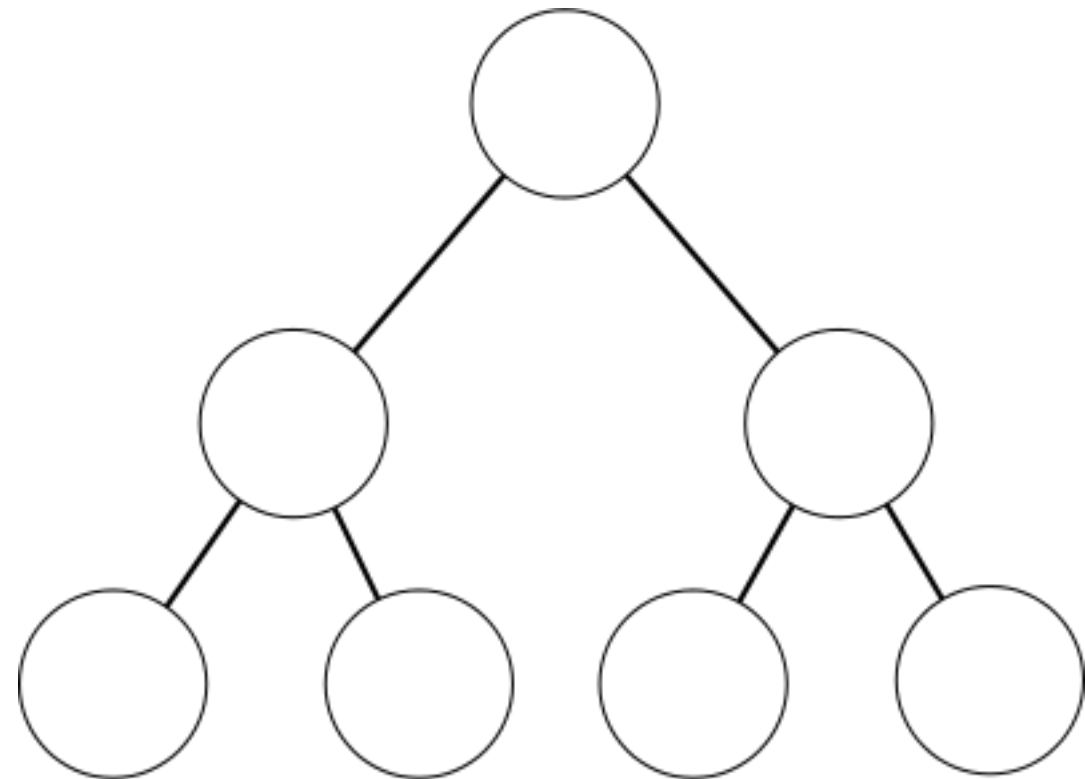
глубина вершины = длина пути от корня к вершине

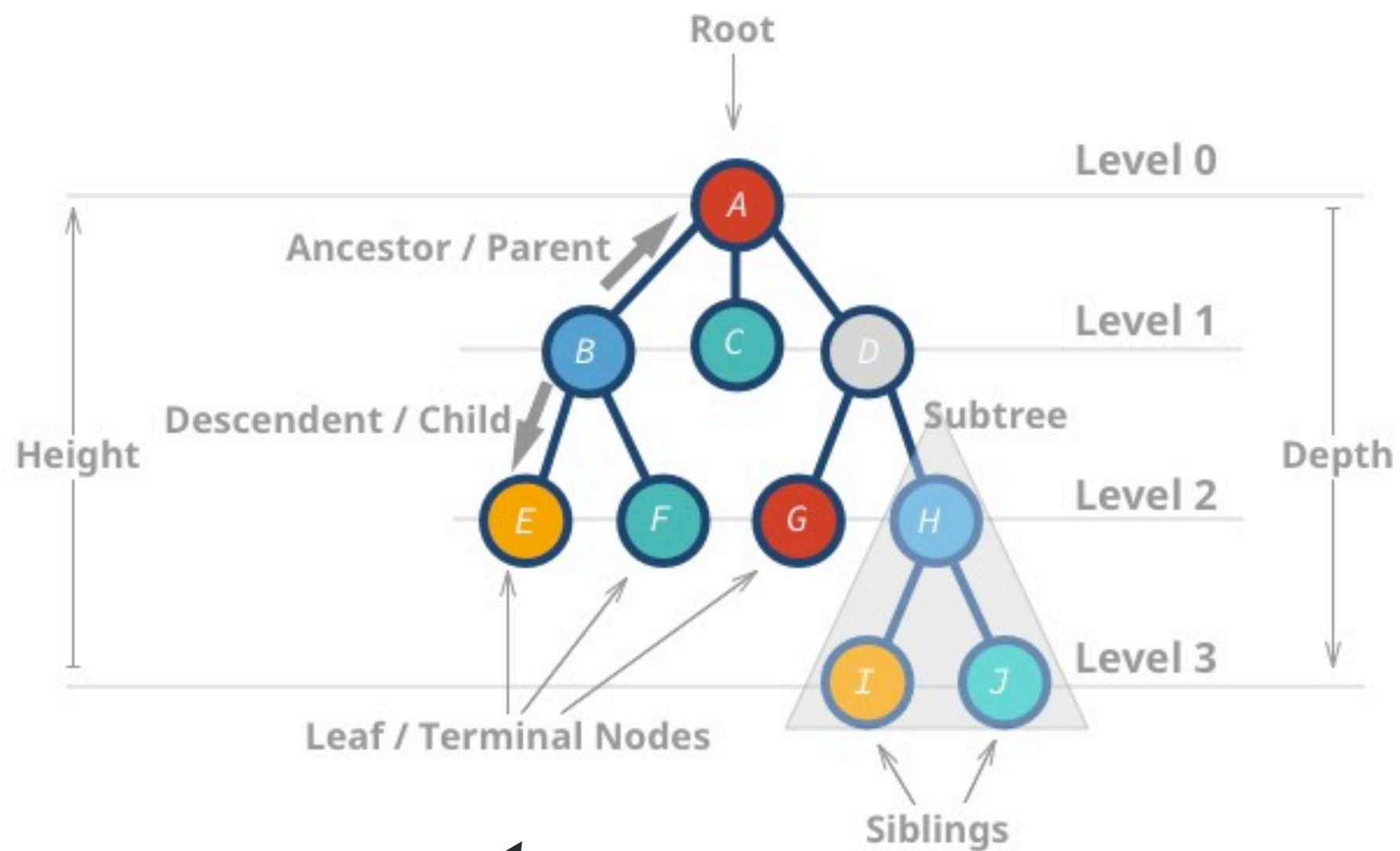


высота дерева = максимальная глубина

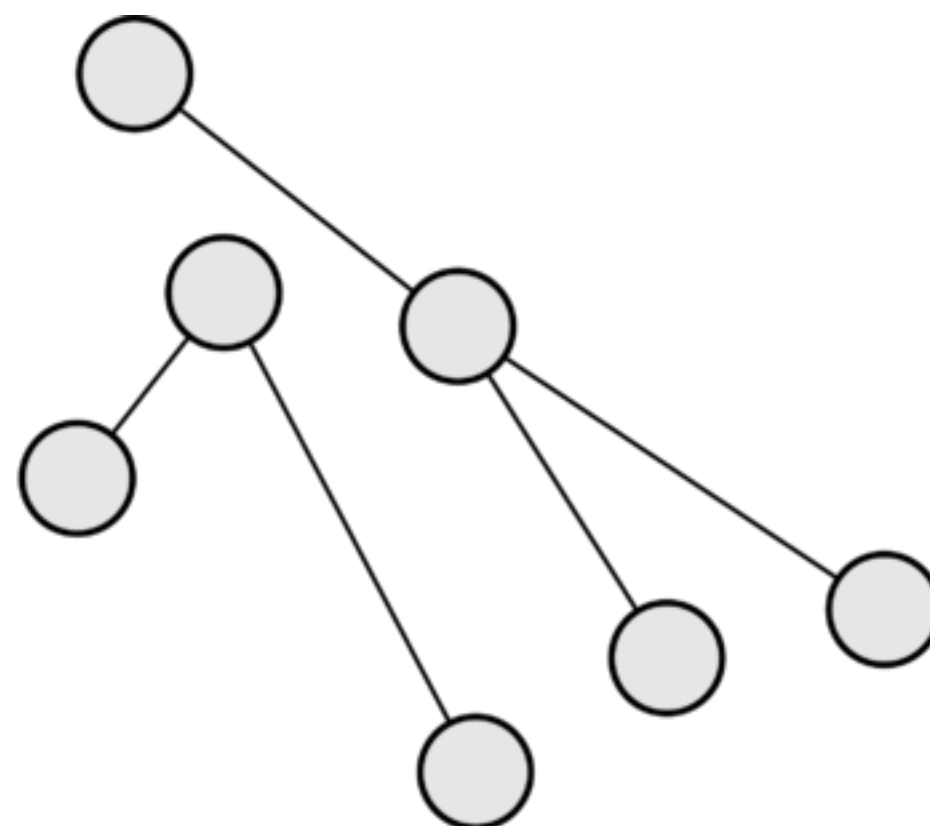
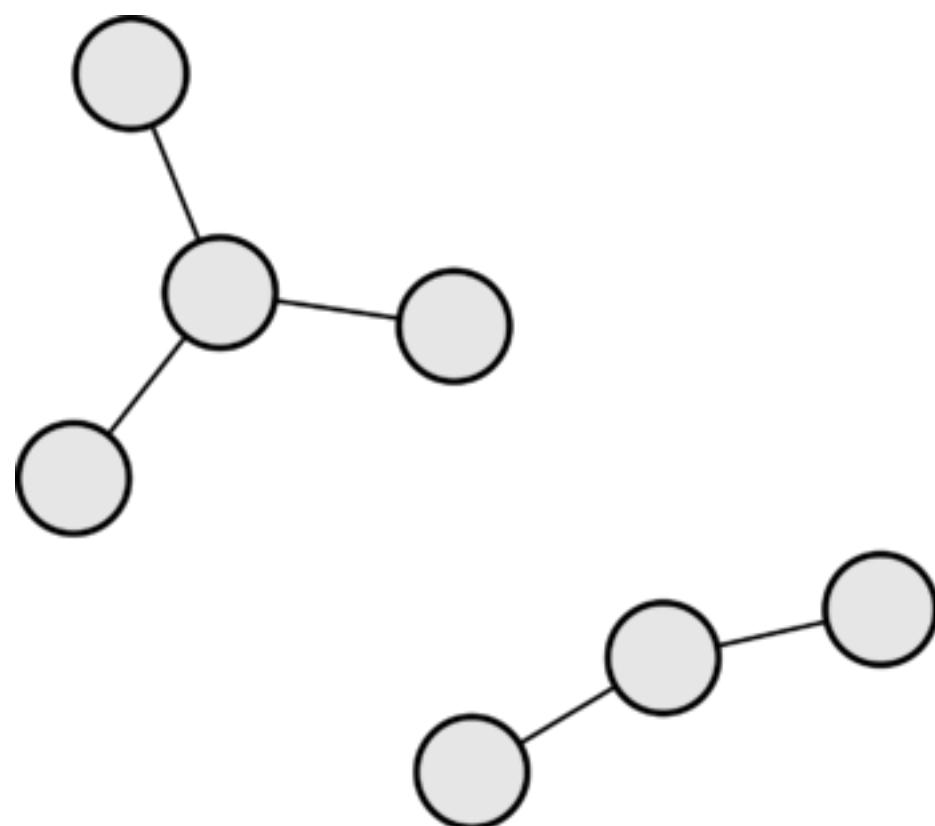


$$|\text{Ветвей}| = |\text{Вершин}| - 1$$

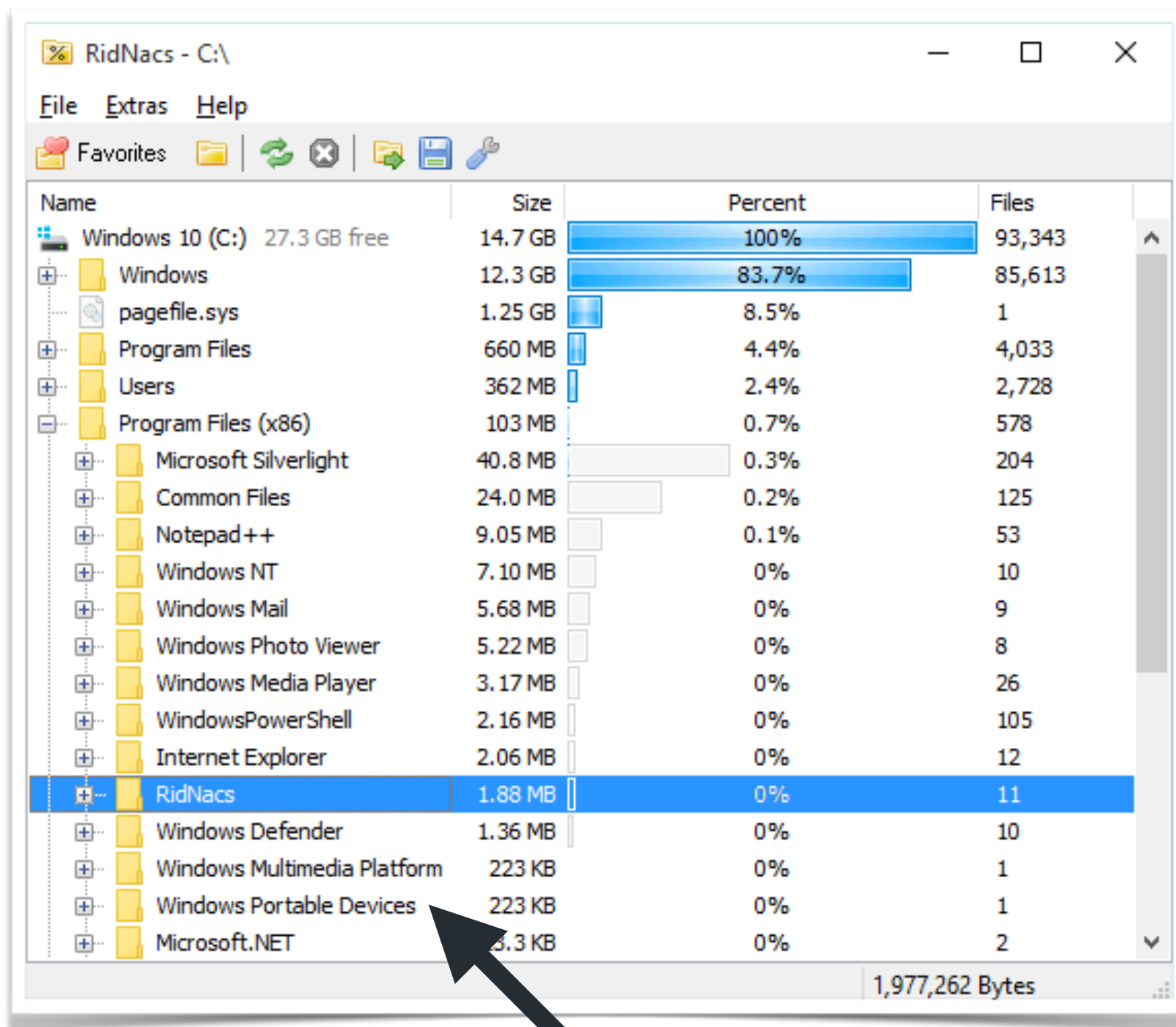




термины на английском



лес = набор деревьев 😊



деревья встречаются часто

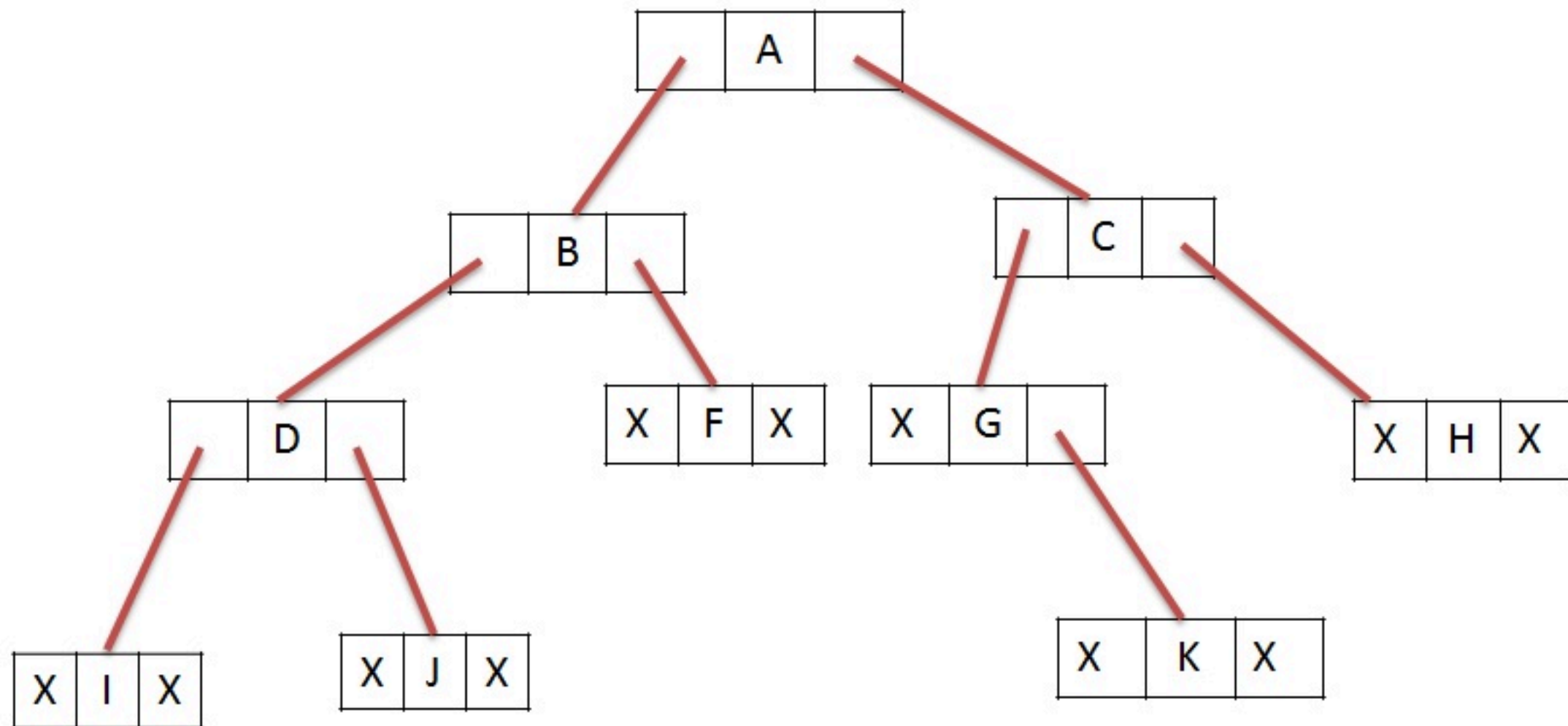
```
html body#{page.bodyid}.#{page.bodyclass} div#container.outer.jb_home div#content_enclosure div#cont
</dl>
</div>
</div>
<div id="content_enclosure">
  <div id="content">
    <ul id="products_pan">
      <li class="idea">
        <h4><a href="idea/"><span>IntelliJ IDEA</span><br/>
          <span class="desc">The Most Intelligent Java IDE</span>
        </a></h4>
      </li>
      <li class="mps">
        <h4><a href="mps/"><span>MPS</span><br/>
          <span class="desc">Meta Programming System</span>
        </a></h4>
      </li>
      <li class="webstorm">
      </li>
    </ul>
```

деревья встречаются часто

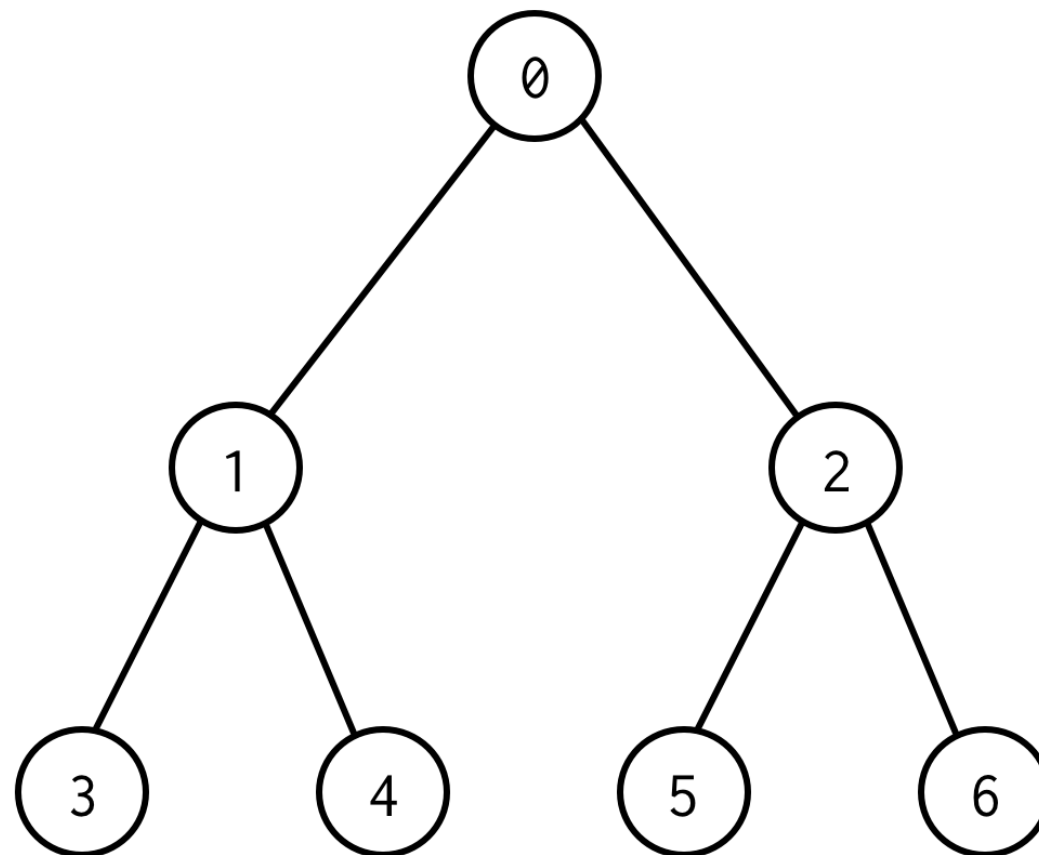


Реализация деревьев

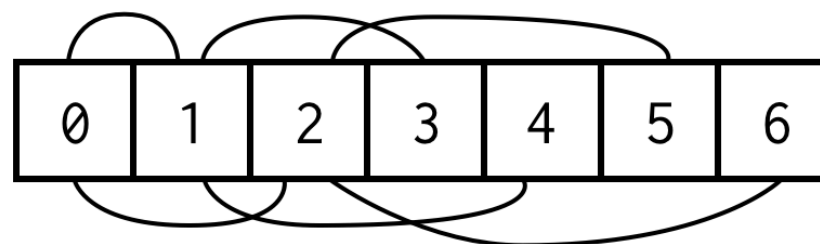
На связном списке



На массиве



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

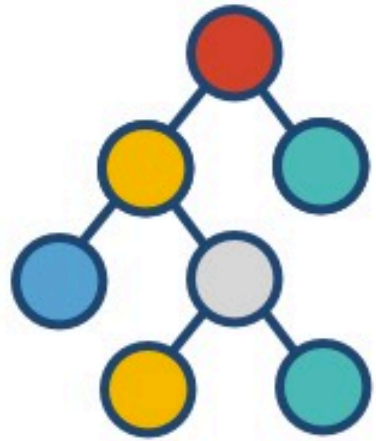


На связном списке в Python

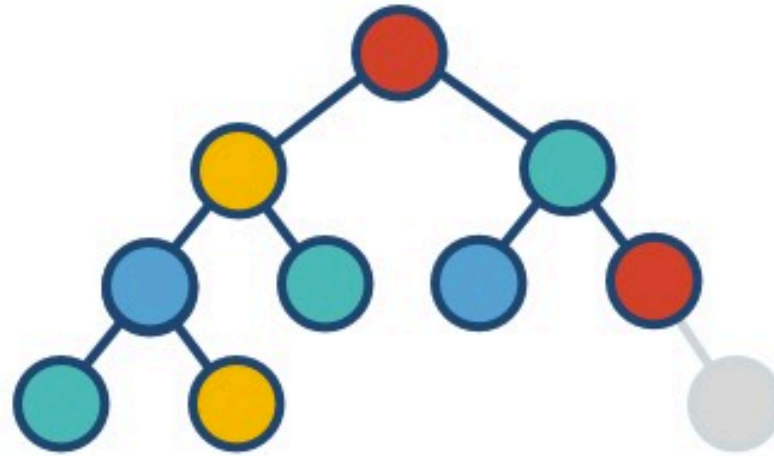


```
1 class Node:
2     def __init__(self, value):
3         self.value = value
4         self.left_child = None
5         self.right_child = None
6
7
8 a, b, c = Node(1), Node(2), Node(3)
9 a.left_child = b
10 a.right_child = c
```

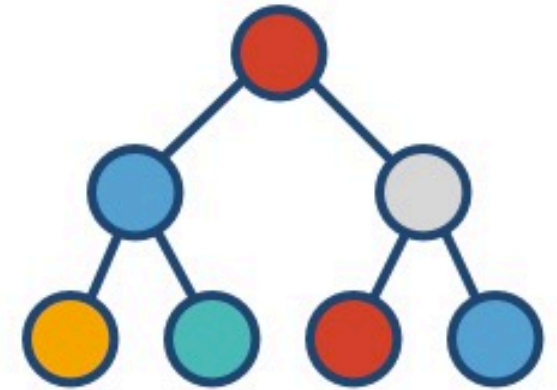
Бинарное дерево



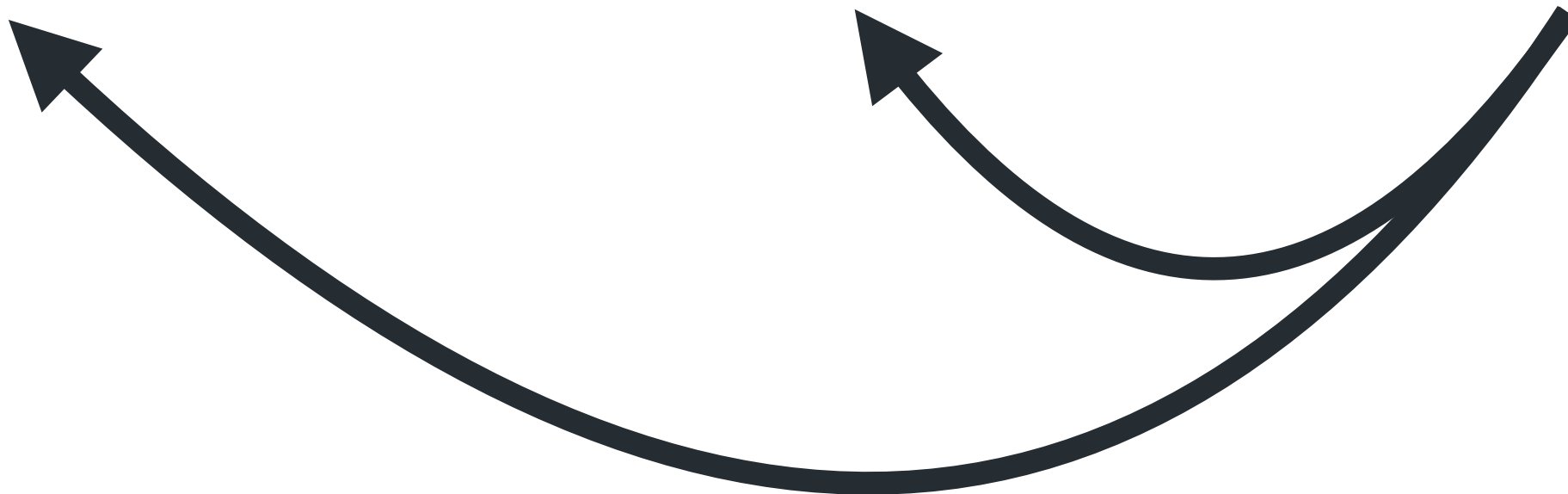
Full Binary Tree

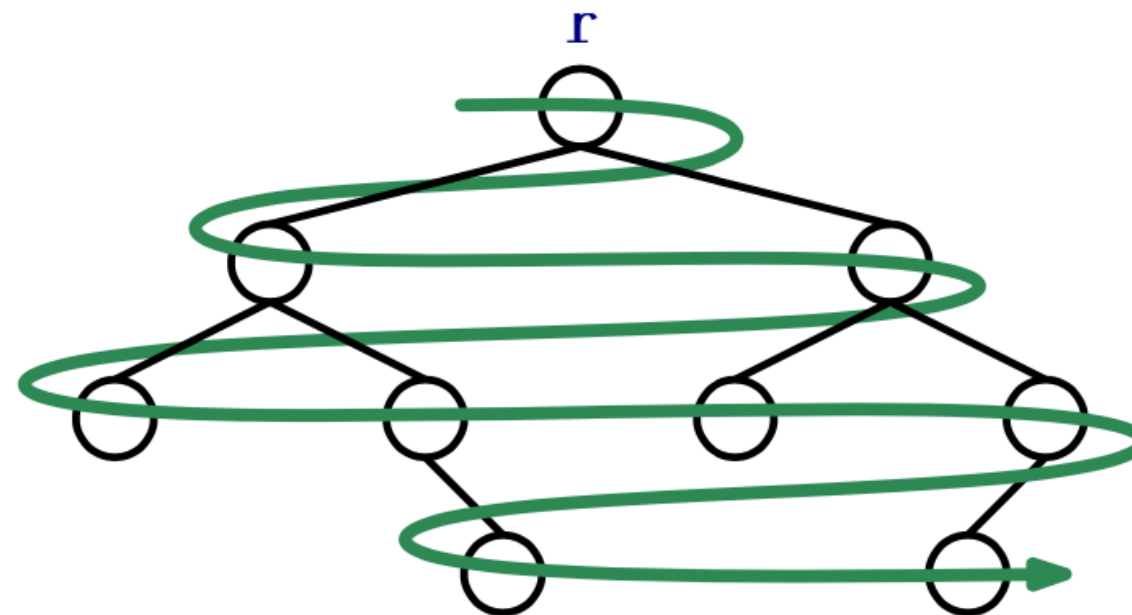


Complete Binary Tree

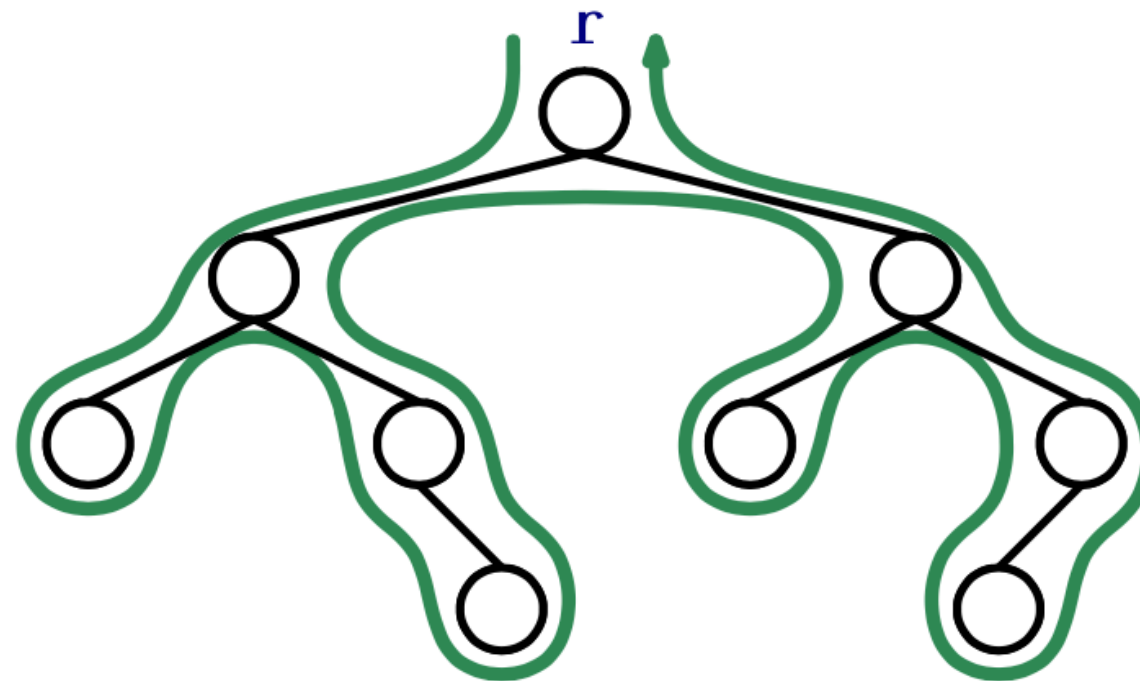


Perfect Binary Tree



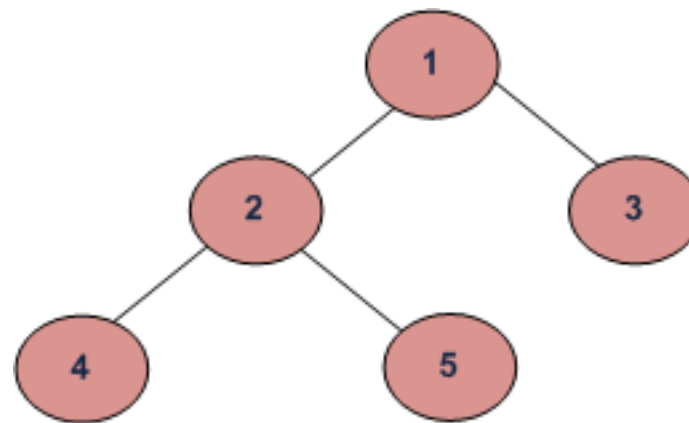


Обход в ширину
Breadth-First Traversal: BFT



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

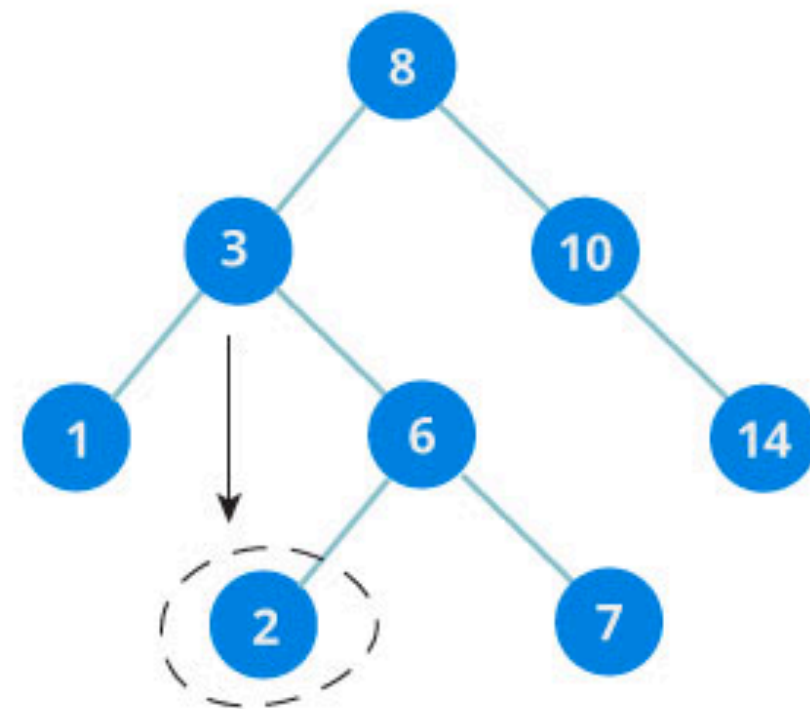
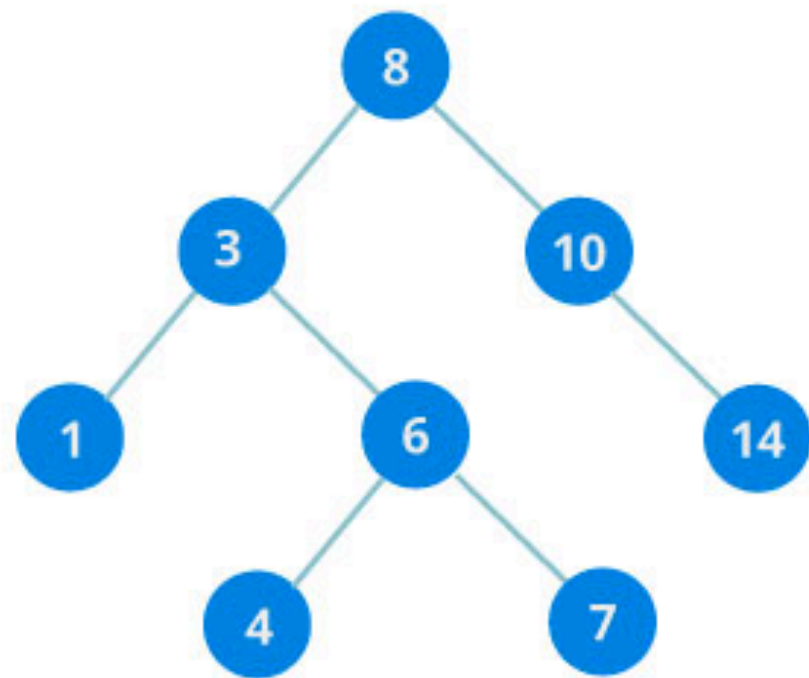


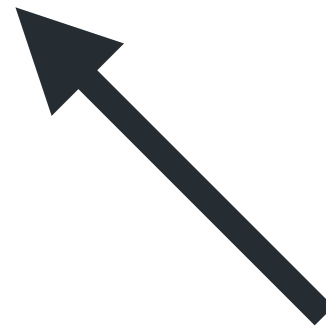
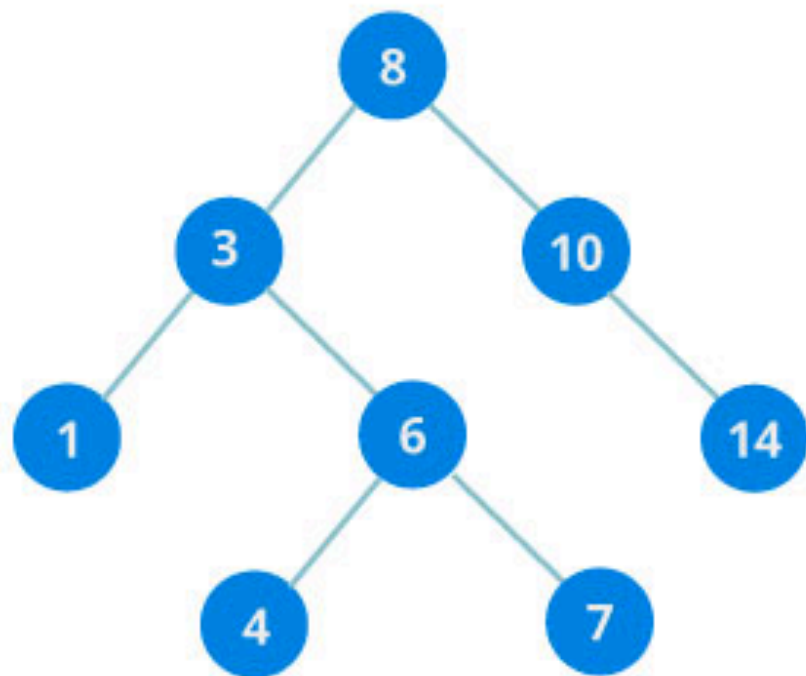
```
1 def walk_preorder(node):
2     if node:
3         print(node.value)
4         walk_preorder(node.left_child)
5         walk_preorder(node.right_child)
6
7 def walk_inorder(node):
8     if node:
9         walk_inorder(node.left_child)
10        print(node.value)
11        walk_inorder(node.right_child)
12
13 def walk_postorder(node):
14     if node:
15         walk_postorder(node.left_child)
16         walk_postorder(node.right_child)
17         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 $key[y] \leq key[x]$. Если y находится в правом поддереве x , то $key[y] \geq key[x]$.





inorder-обход = неубывающий порядок
1 3 4 6 7 8 10 14

Поиск 🕵️

```
1 def search(root, x):  
2     if root is None or root.value == x:  
3         return root  
4     if root.value < x:  
5         return search(root.right_child, x)  
6     return search(root.left_child, x)
```

высота дерева

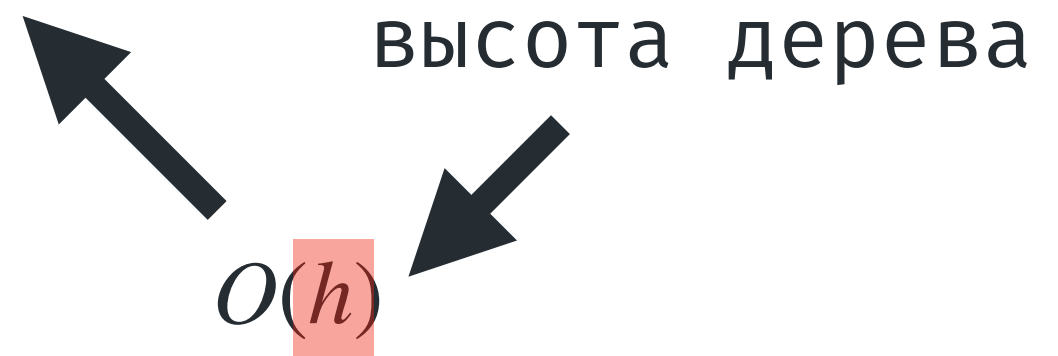
$O(h)$

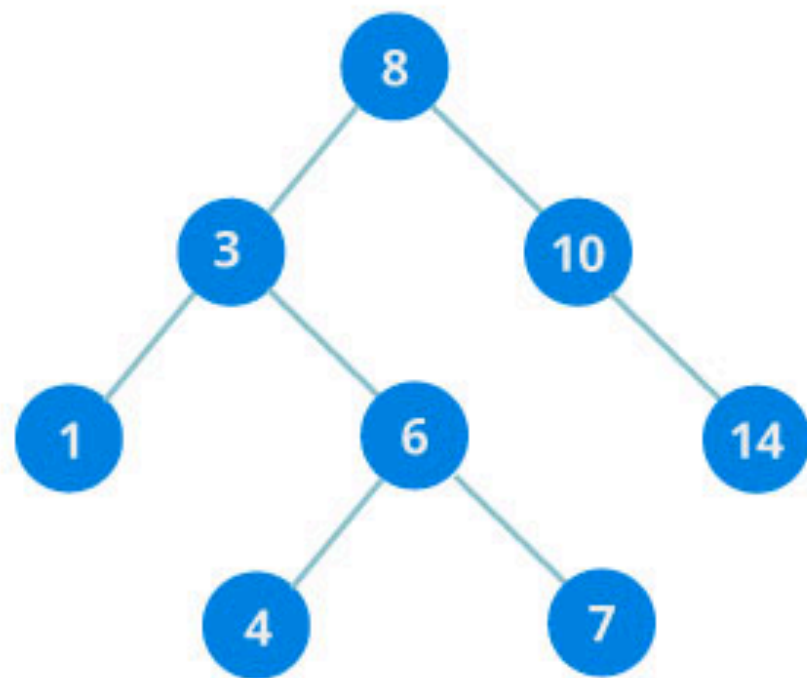
Поиск 🕵️

```
1 def search(root, x):  
2     while root is not None and root.value != x:  
3         root = root.right_child if root.value < x else root.left_child  
4     return root
```

высота дерева

$O(h)$





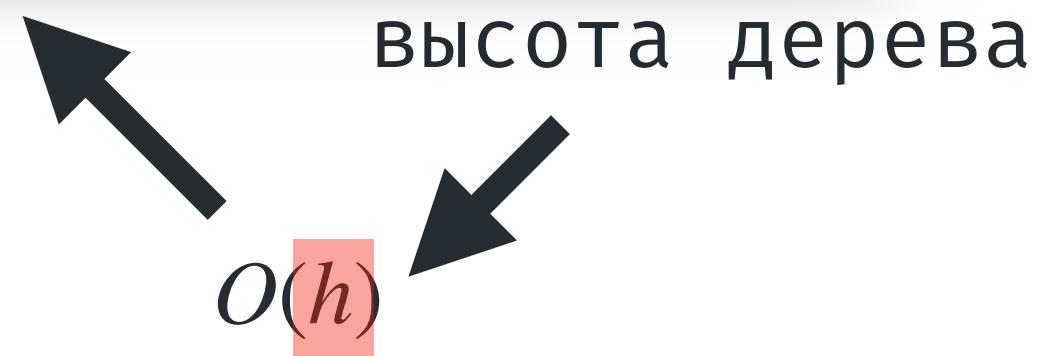
Как найти min и max?

Вставка

```
1 def insert(root, x):
2     if root is None:
3         return Node(x)
4
5     if root.value ≤ x:
6         root.right_child = insert(root.right_child, x)
7     else:
8         root.left_child = insert(root.left_child, x)
9     return root
```

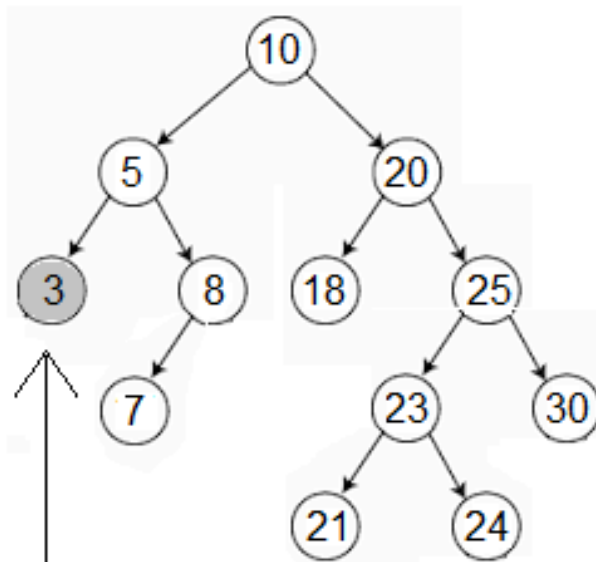
высота дерева

$O(h)$



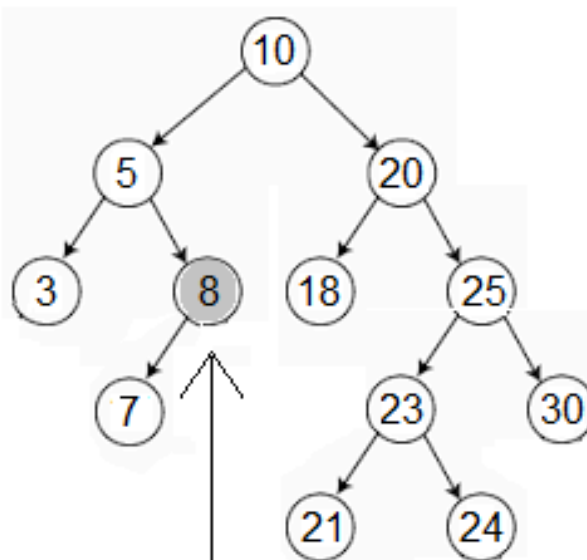
Удаление

Case 1



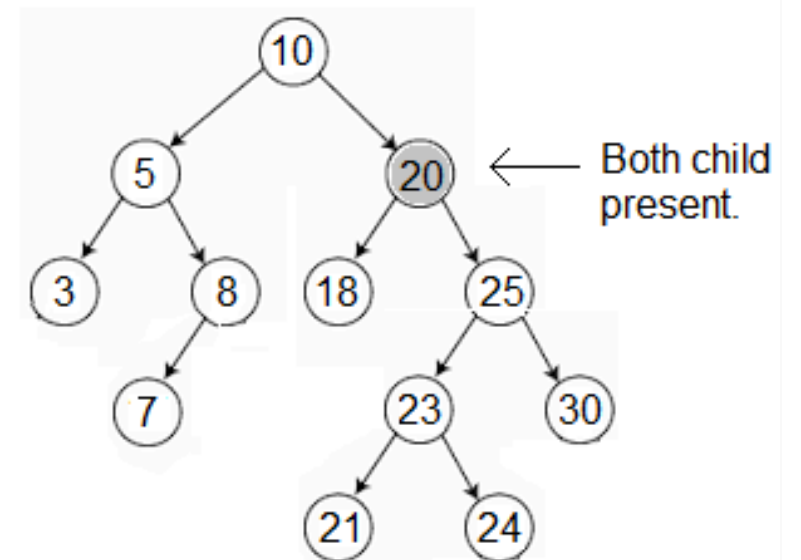
No Child Present

Case 2



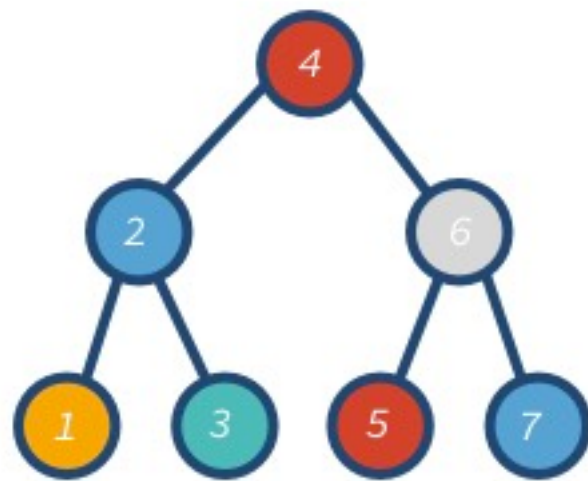
One child present,
Left child present in our case.

Case 3

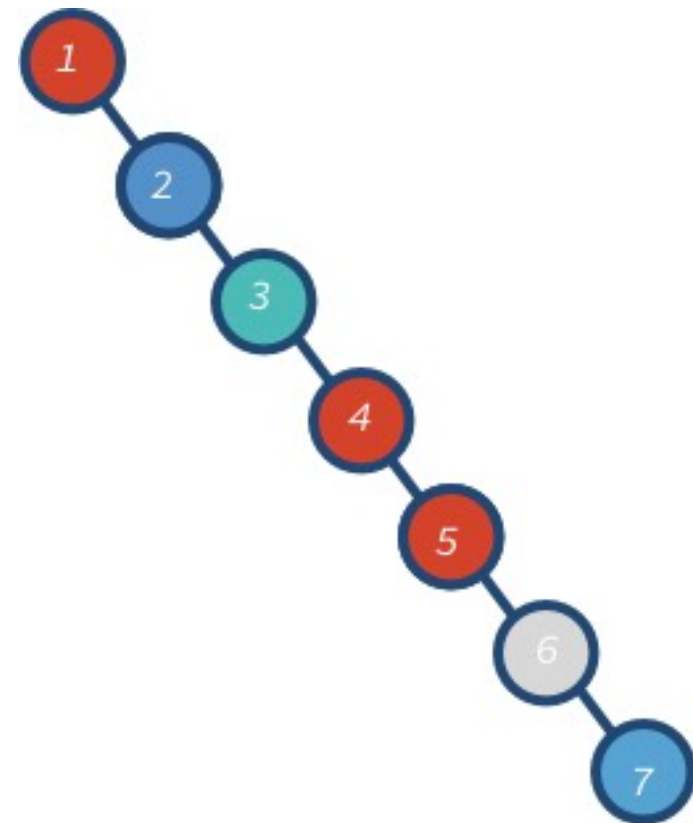


```
1 def find_min_node(root):
2     while root.left_child is not None:
3         root = root.left_child
4     return root
5
6 def delete(root, x):
7     if root is None: # Случай 0. Пустое дерево
8         return root
9     if x < root.value: # Рекурсивно опускаемся по дереву влево
10        root.left_child = delete(root.left_child, x)
11    elif x > root.value: # Рекурсивно опускаемся по дереву вправо
12        root.right_child = delete(root.right_child, x)
13    else: # Если нашли вершину ⇒ удаляем
14        # Случай 1 и 2. Нет детей или 1 ребенок
15        if root.left_child is None:
16            return root.right_child
17        if root.right_child is None:
18            return root.left_child
19        # Случай 3. 2 ребенка
20        min_node = find_min_node(root.right_child)
21        root.value = min_node.value
22        root.right_child = delete(root.right_child , min_node.value)
23    return root
```

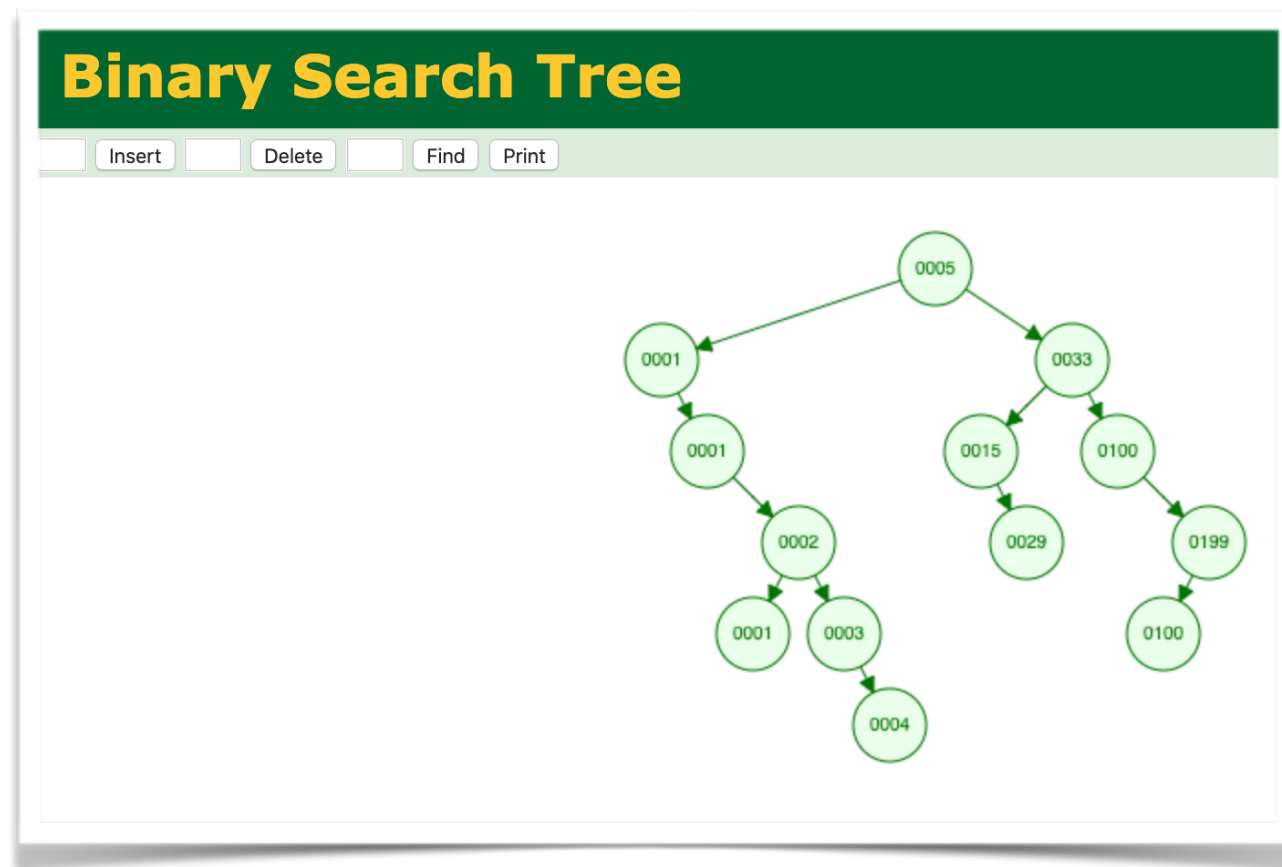
$$\lfloor \log_2 n \rfloor \leq h \leq n - 1$$



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	
<u>Array</u>	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$
<u>Stack</u>	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$
<u>Queue</u>	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$
<u>Singly-Linked List</u>	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$
<u>Doubly-Linked List</u>	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$
<u>Skip List</u>	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n \log(n))$
<u>Hash Table</u>	N/A	$\theta(1)$	$\theta(1)$	$\theta(1)$	N/A	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$
<u>Binary Search Tree</u>	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$
<u>Cartesian Tree</u>	N/A	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	N/A	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$
<u>B-Tree</u>	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$
<u>Red-Black Tree</u>	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$
<u>Splay Tree</u>	N/A	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	N/A	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$
<u>AVL Tree</u>	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$
<u>KD Tree</u>	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$

Сбалансированные деревья

Что делать с
несбалансированным
деревом?



Сбалансировать
(повороты дерева)

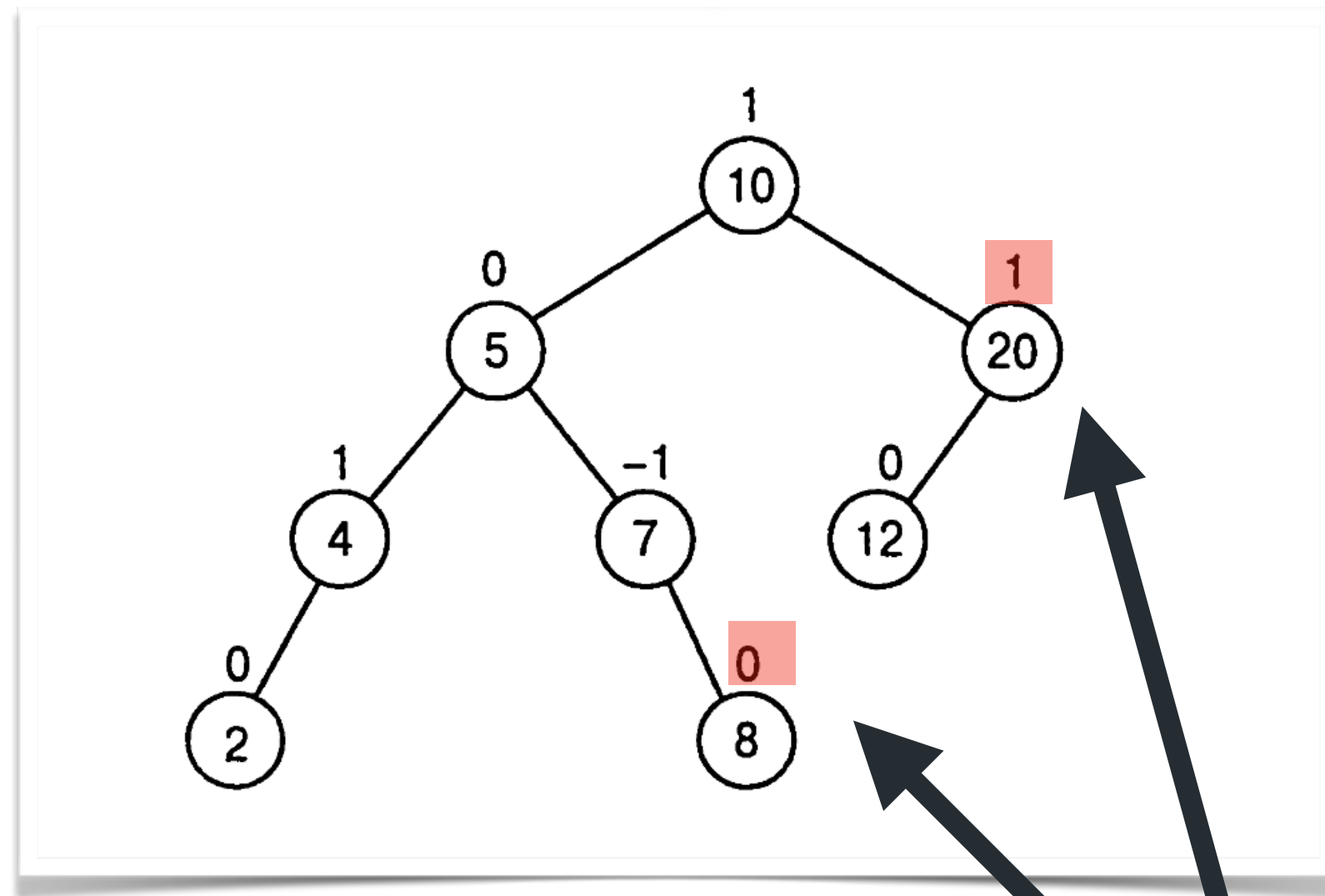


Модифицировать
дерево

Common Data Structure Operations

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

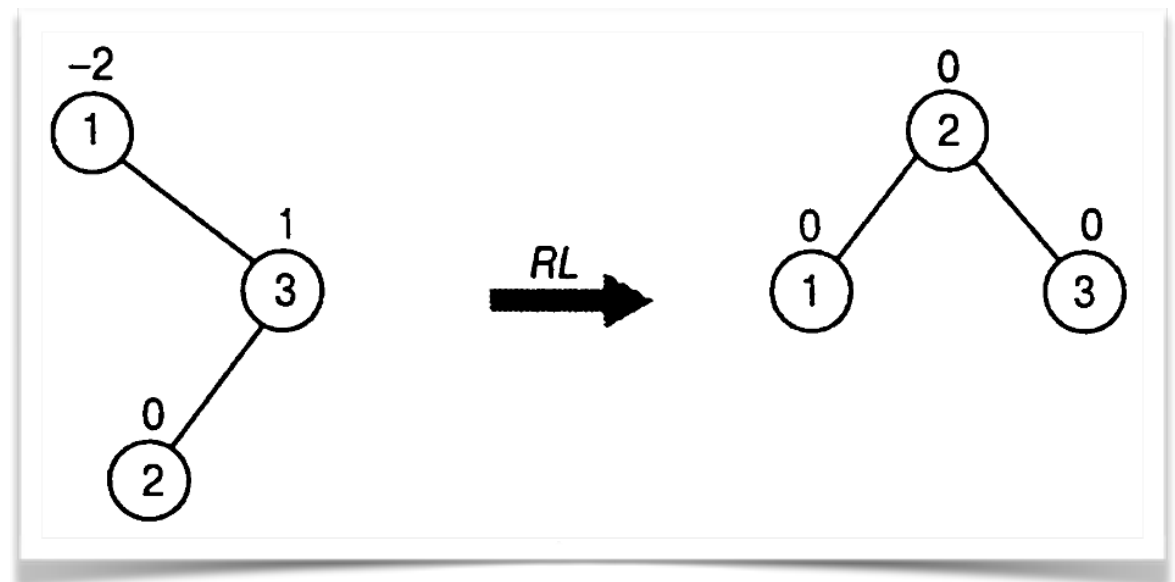
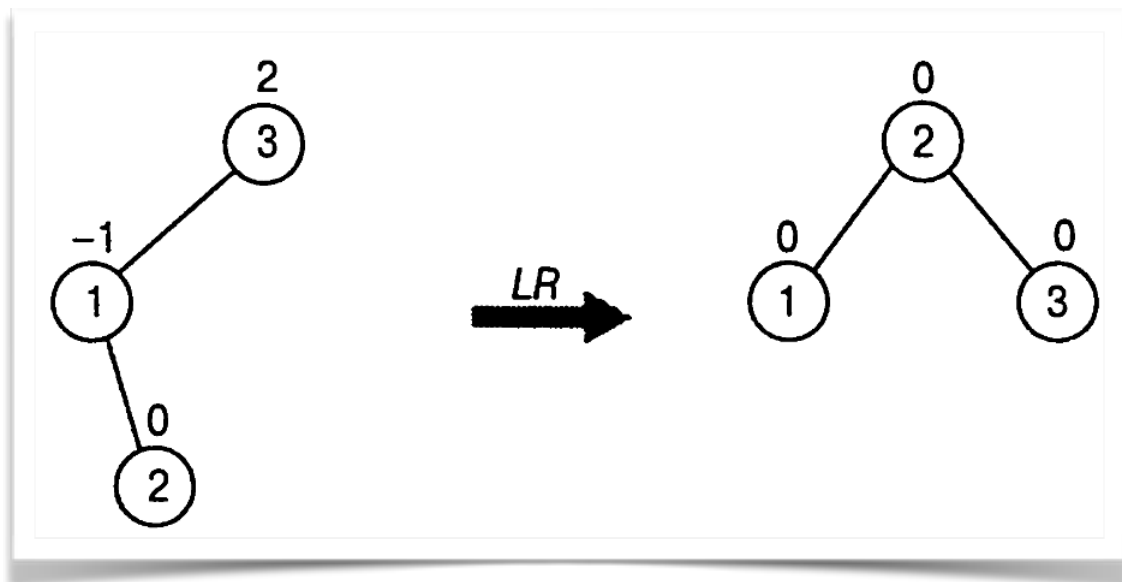
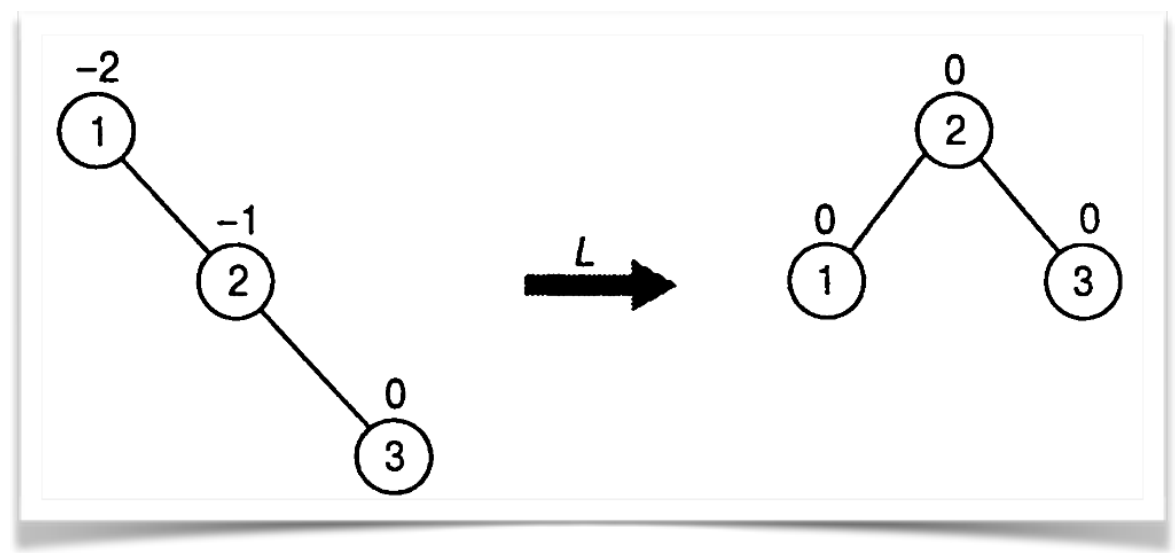
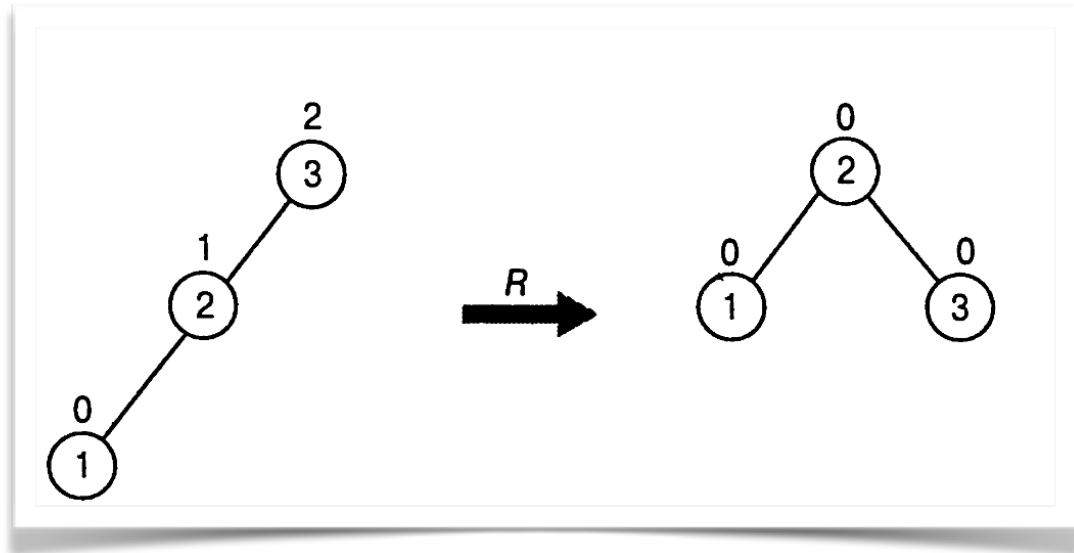
AVL-дерево

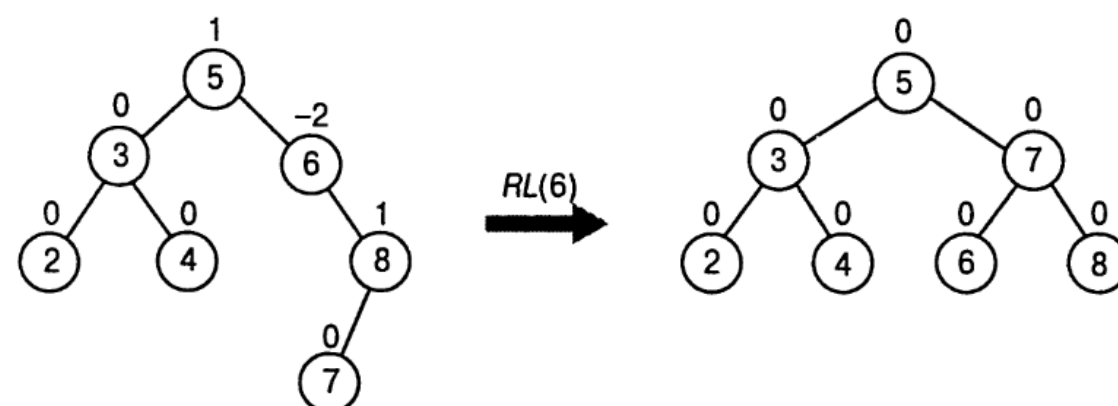
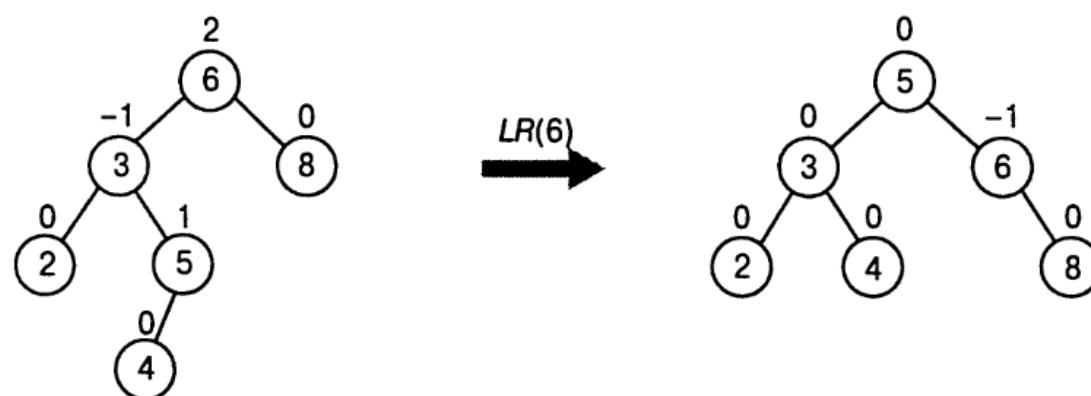
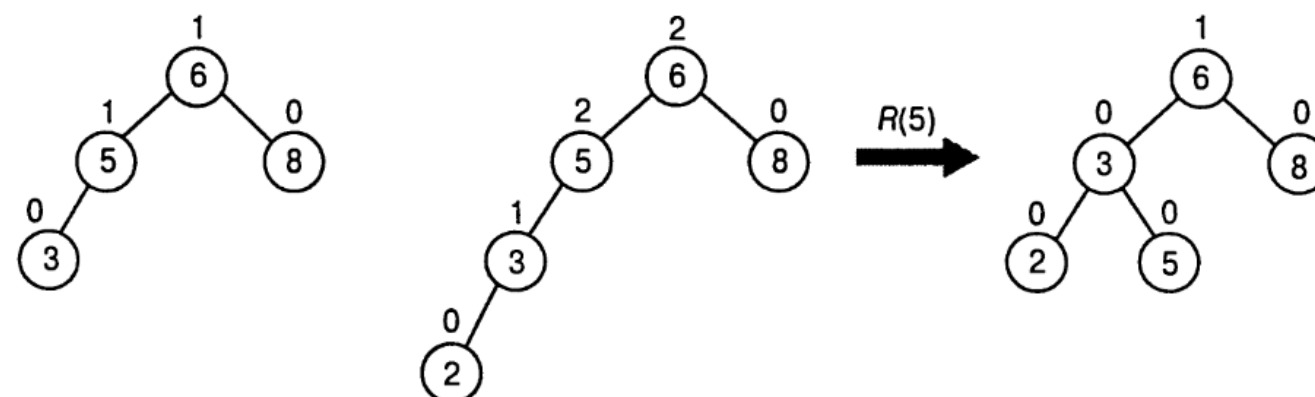
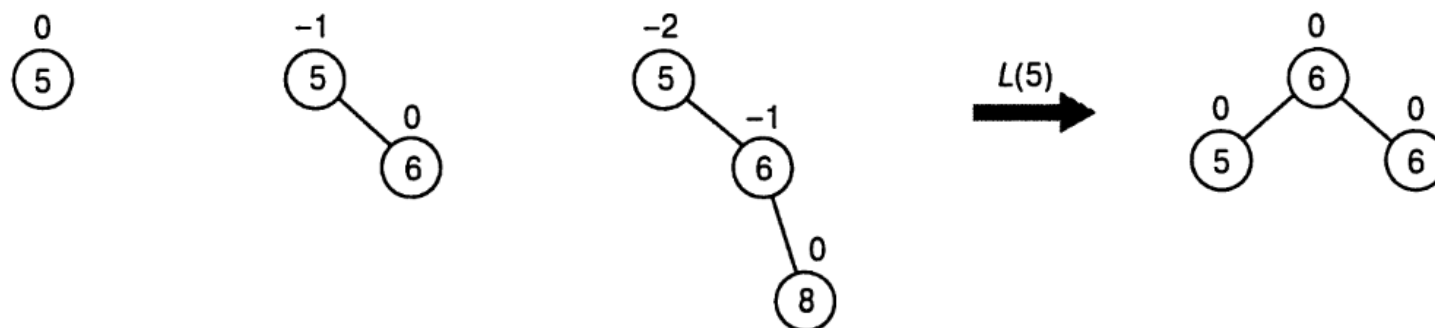


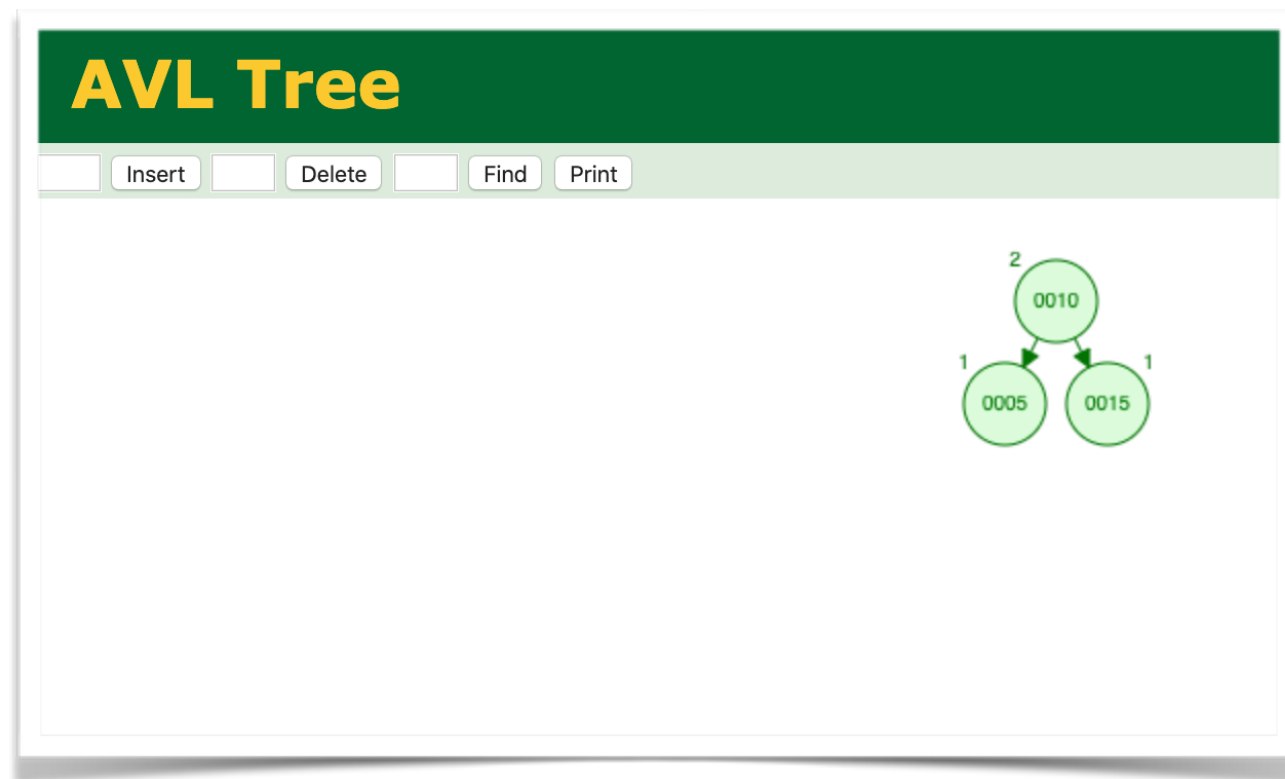
0 или ± 1

balance factor

AVL-дерево (rotations)





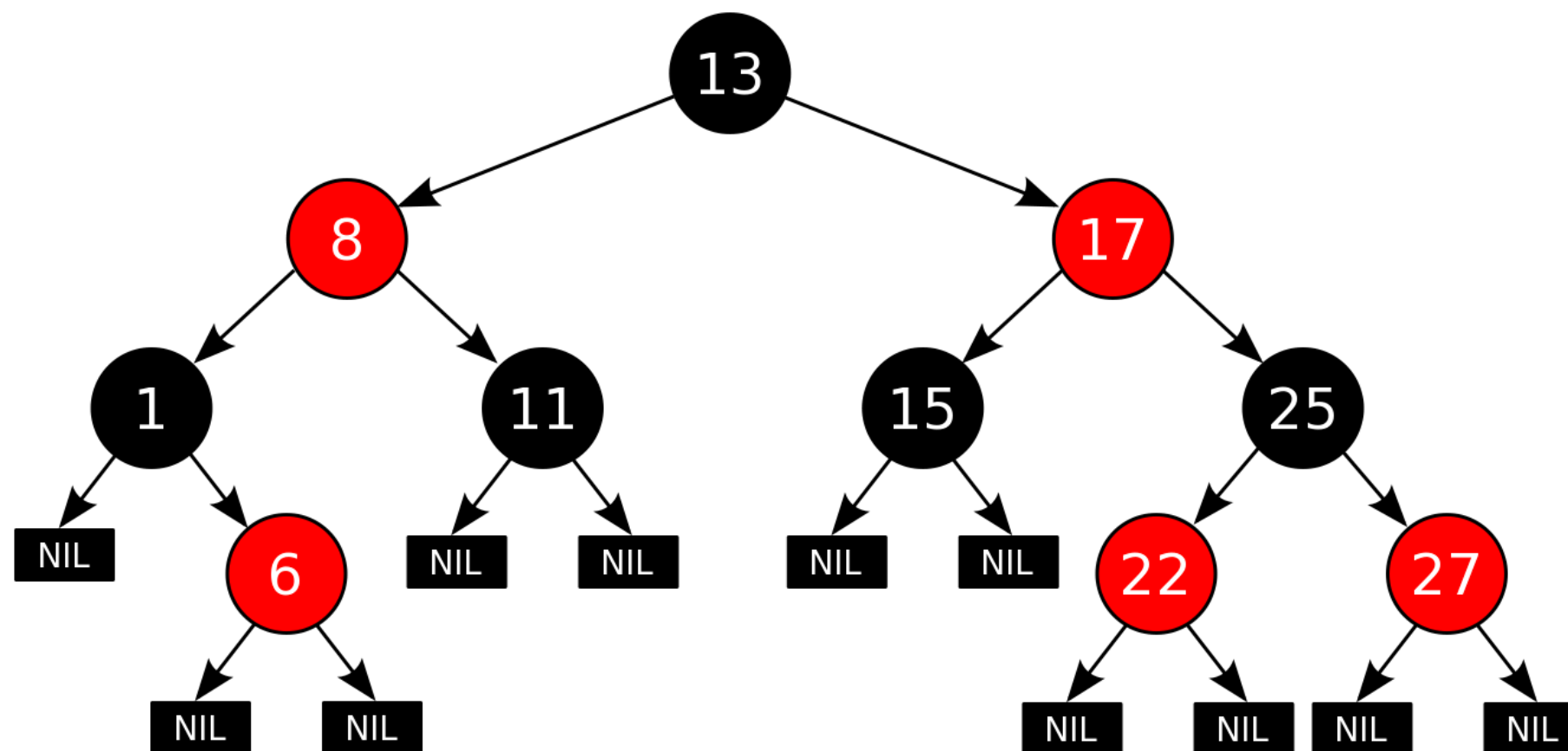


cs.usfca.edu/~galles/visualization/AVLtree.html

Common Data Structure Operations

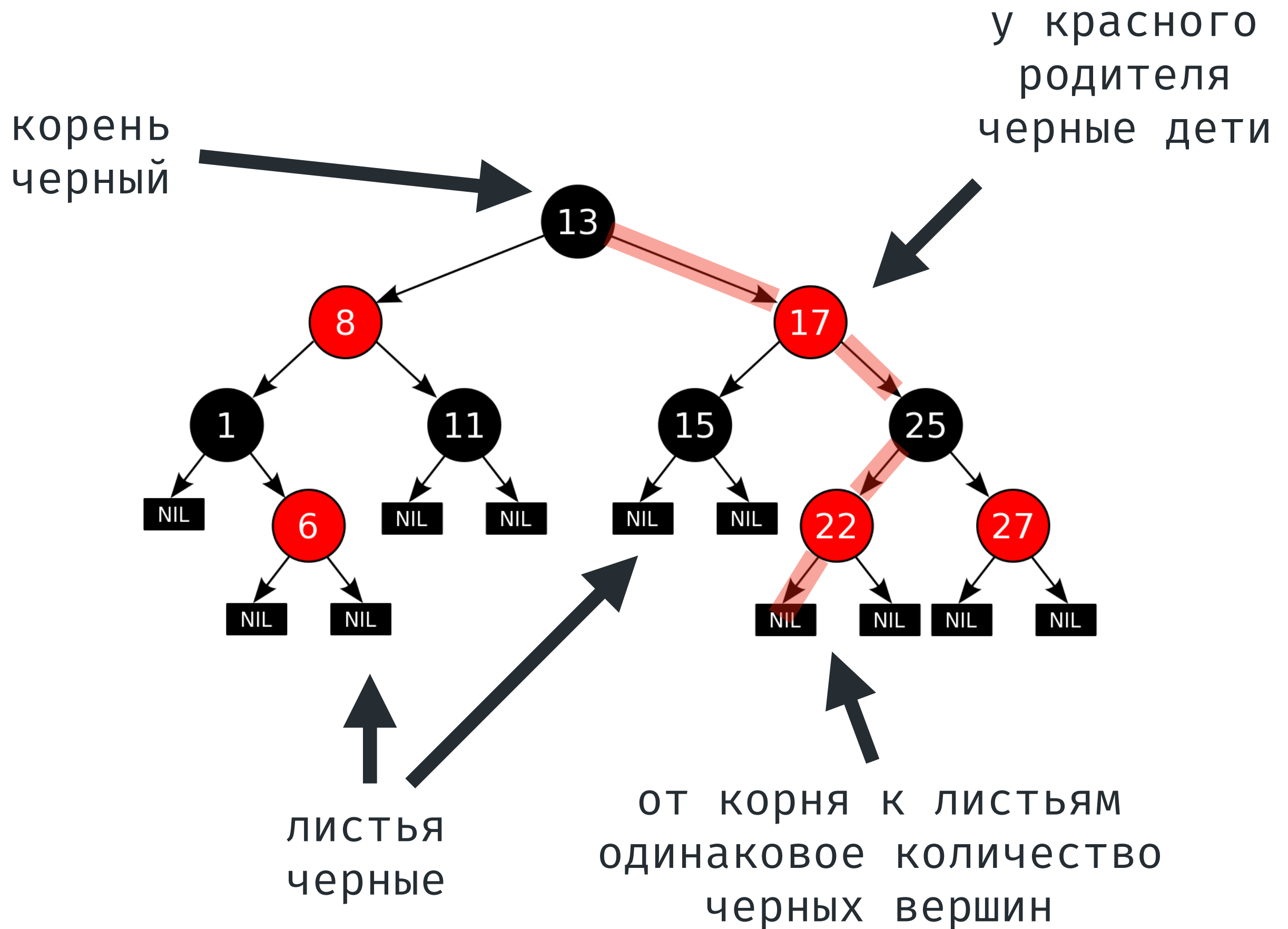
Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
<u>Array</u>	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$
<u>Stack</u>	$\theta(n)$	$\theta(n)$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

Красно-черное дерево

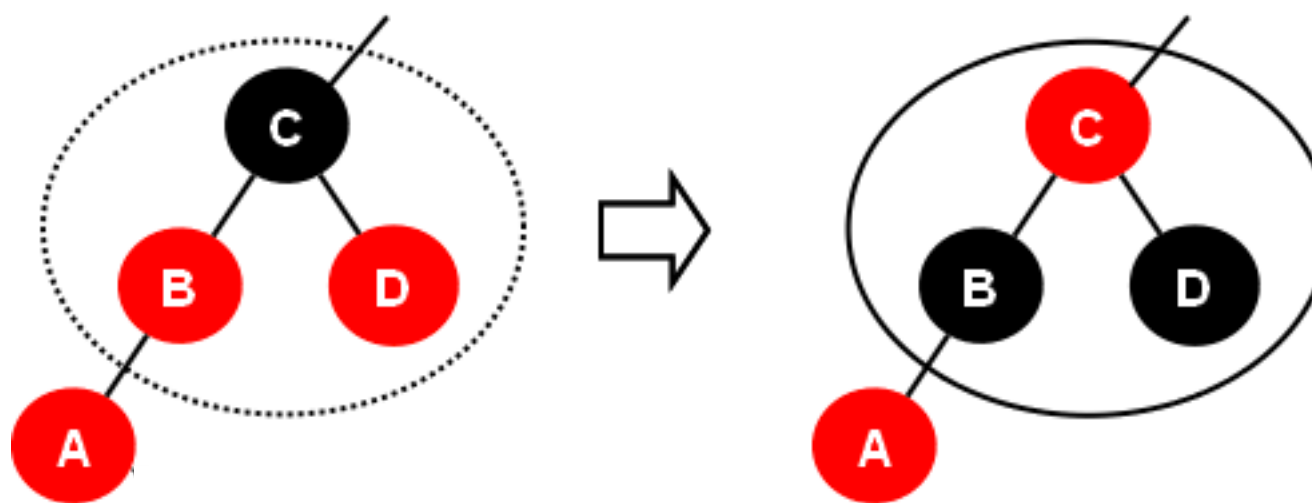
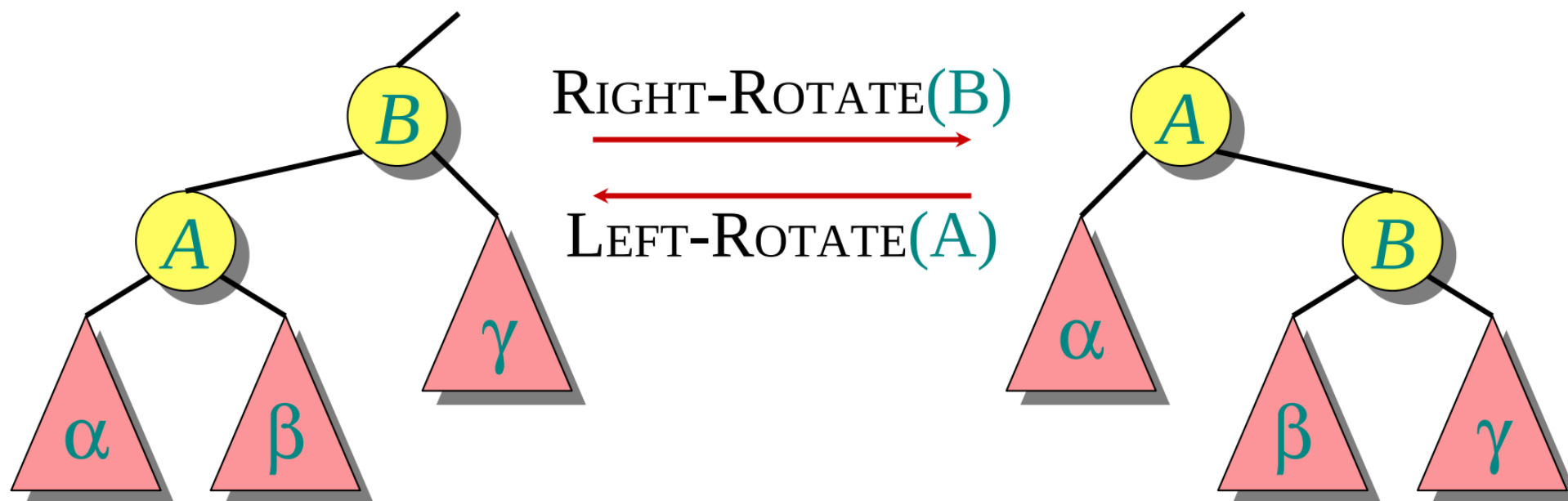


$$h \leq 2 \log(n + 1)$$

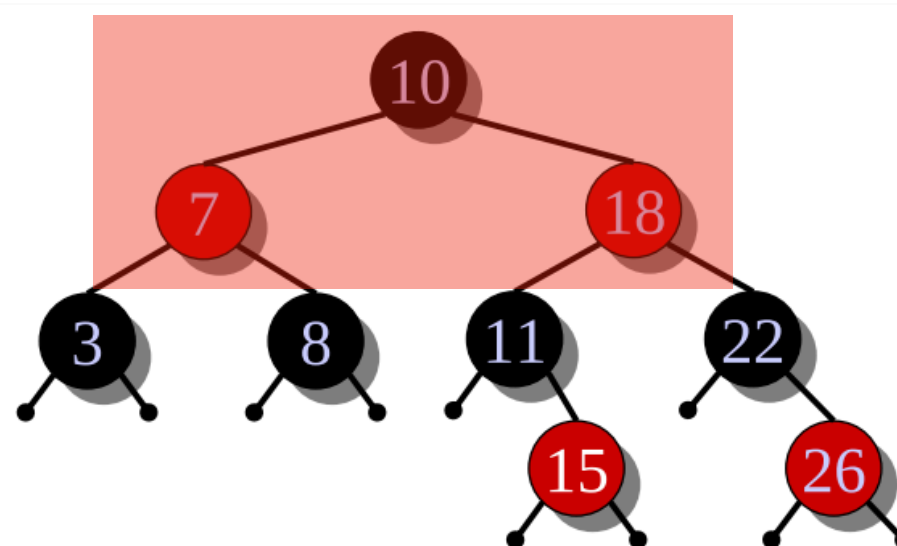
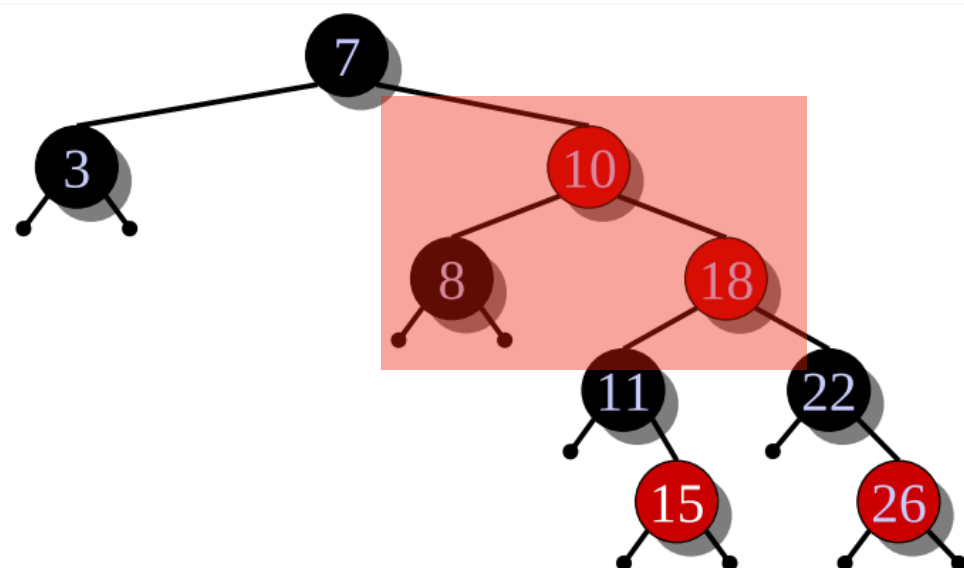
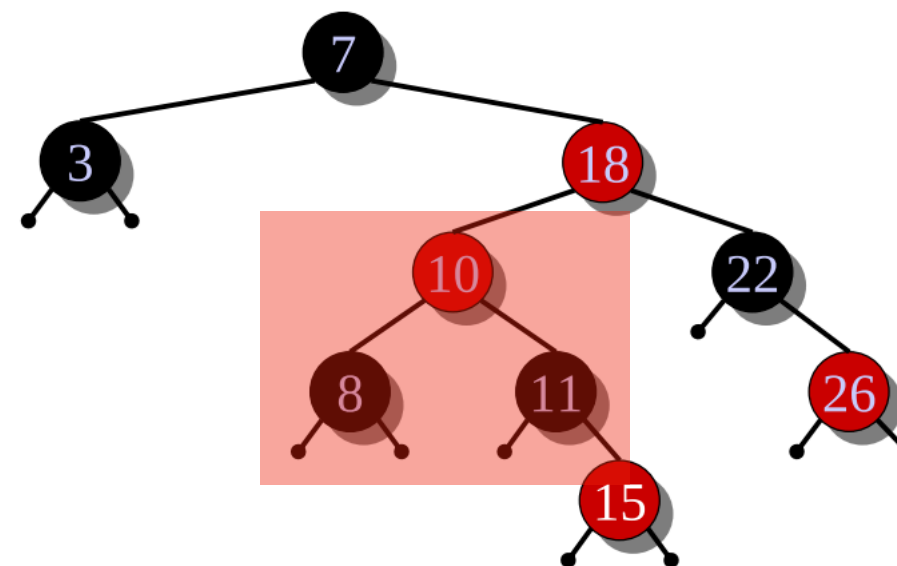
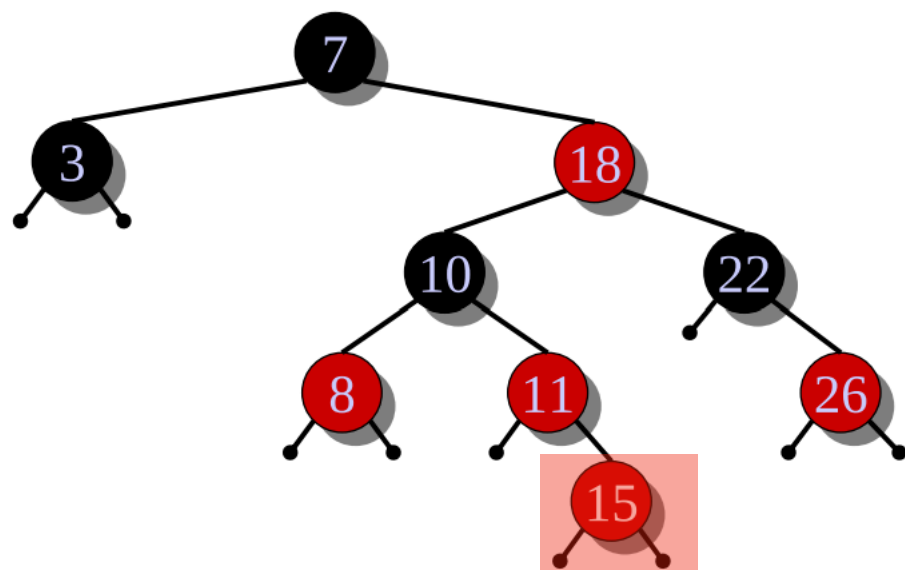
Красно-черное дерево

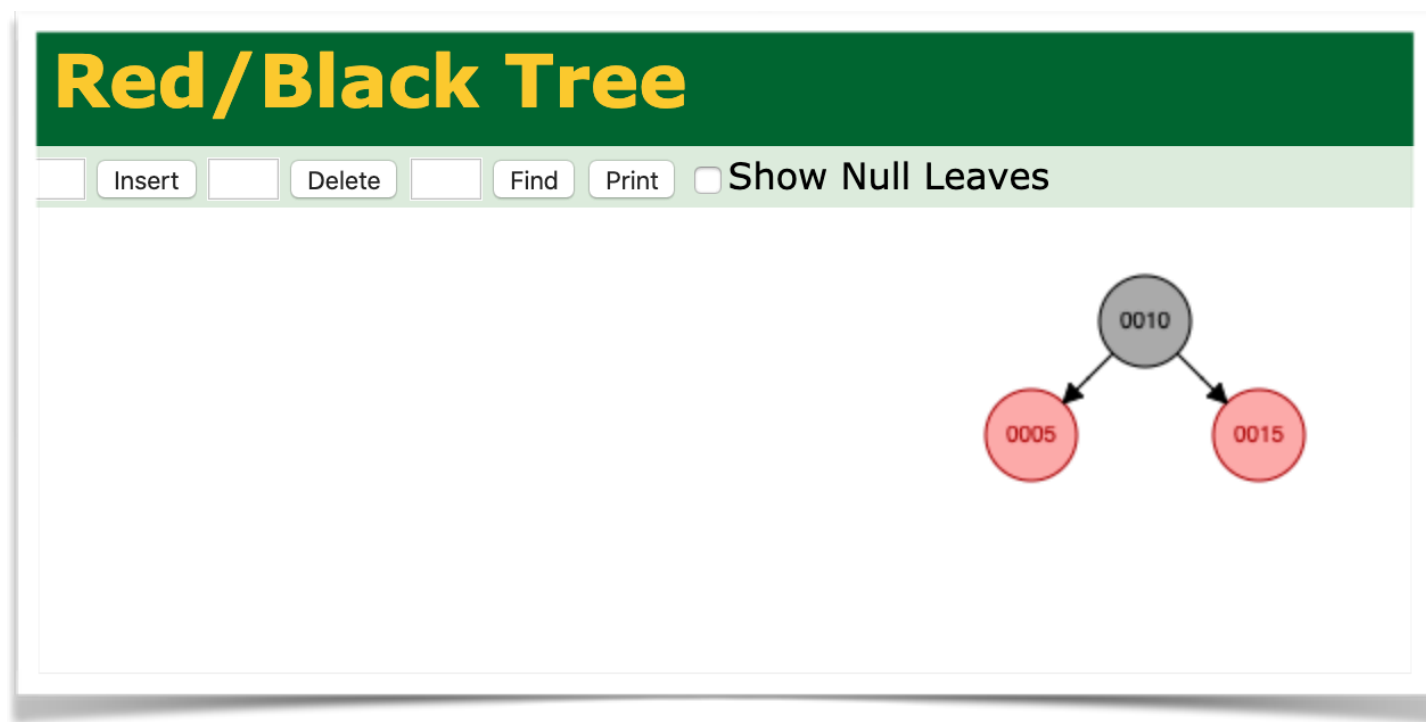


Красно-черное дерево



Красно-черное дерево



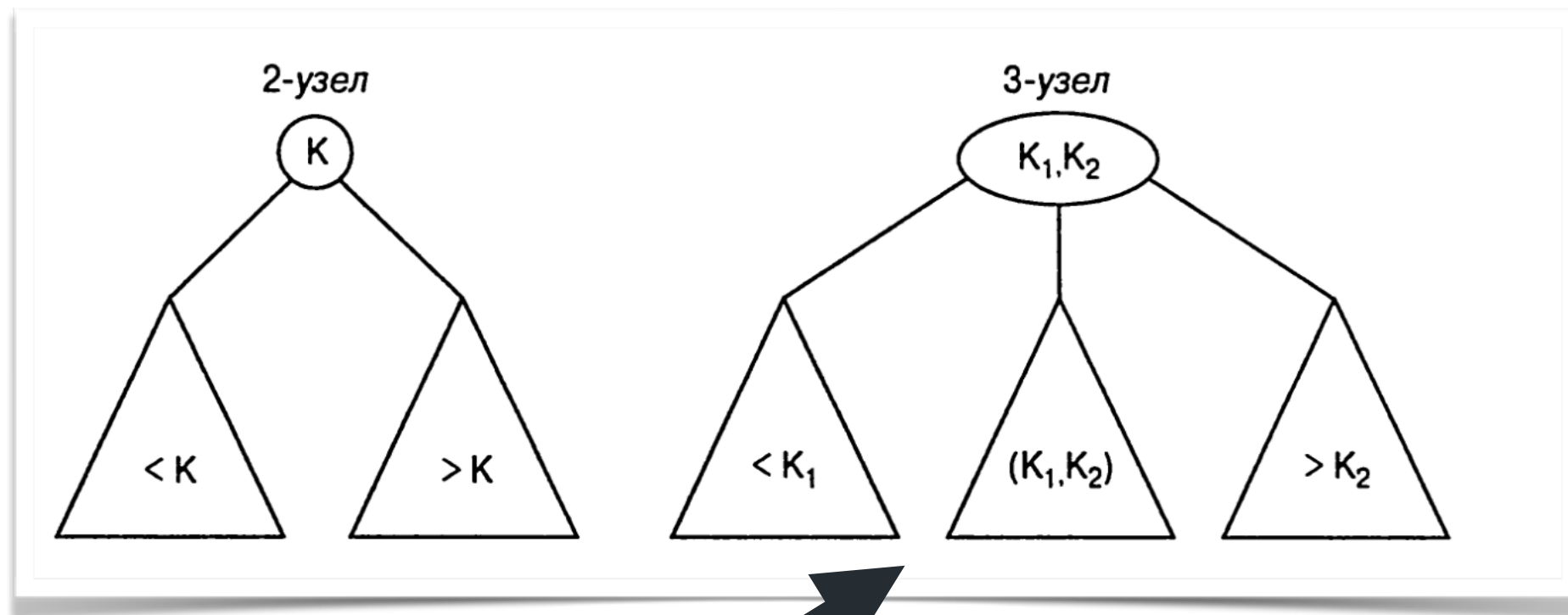


cs.usfca.edu/~galles/visualization/RedBlack.html

Common Data Structure Operations

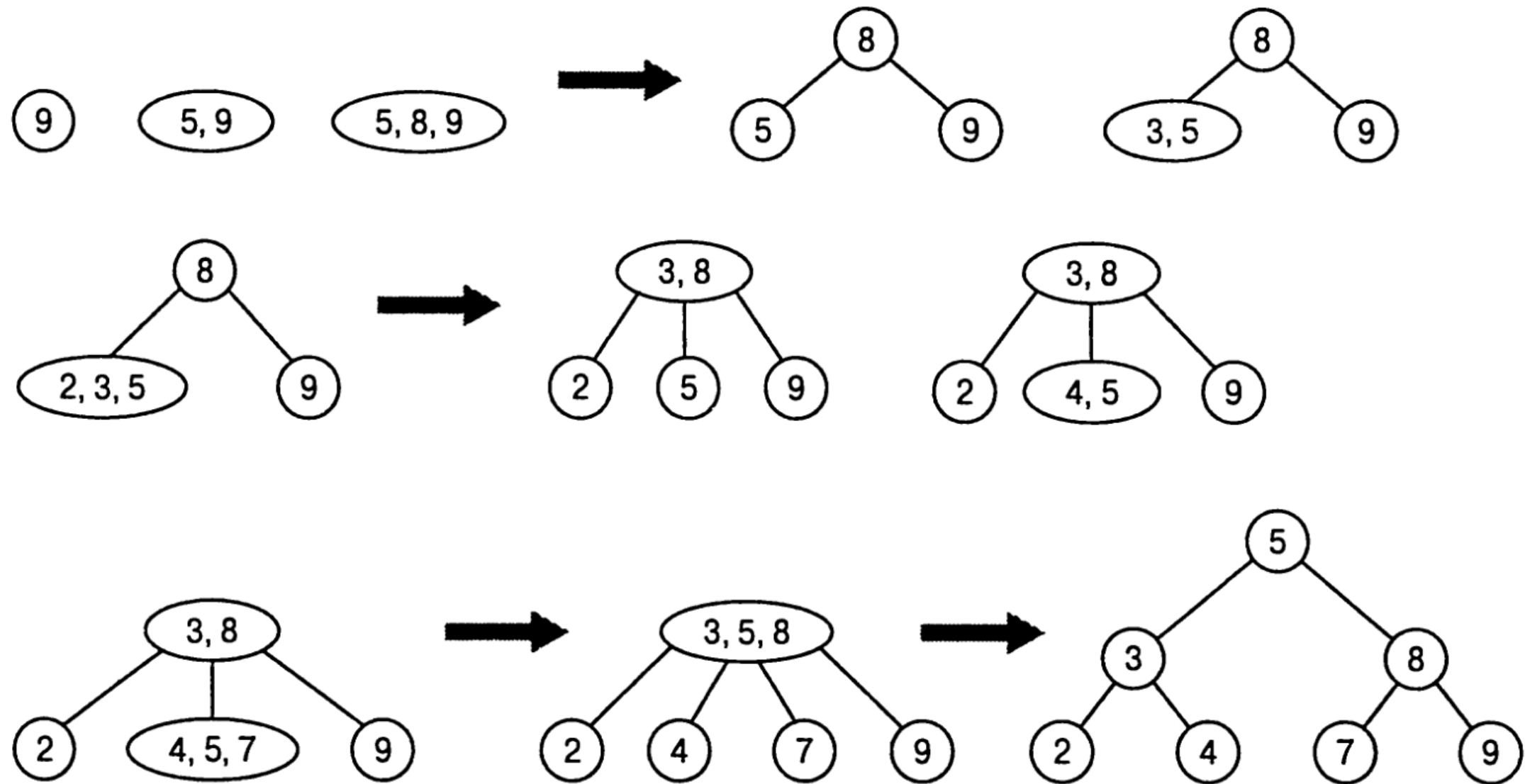
Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
<u>Array</u>	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$
<u>Stack</u>	$\theta(n)$	$\theta(n)$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

2-3-дерево



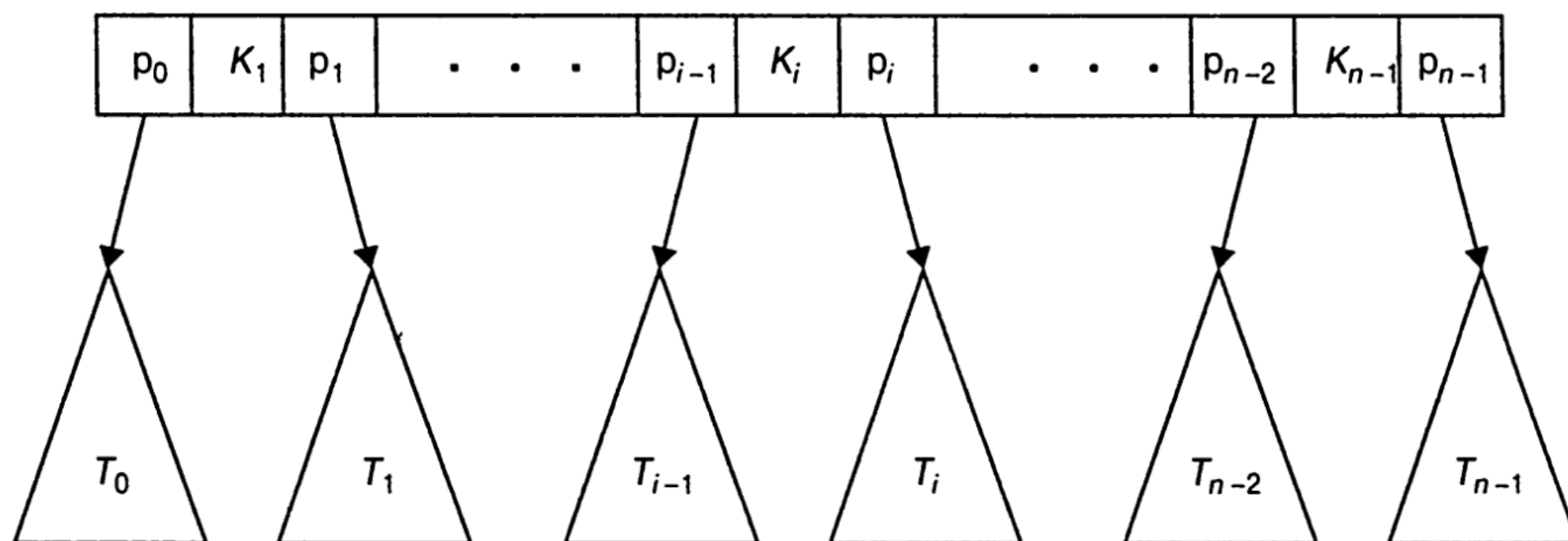
поддерживая
одной высоты

2-3-дерево

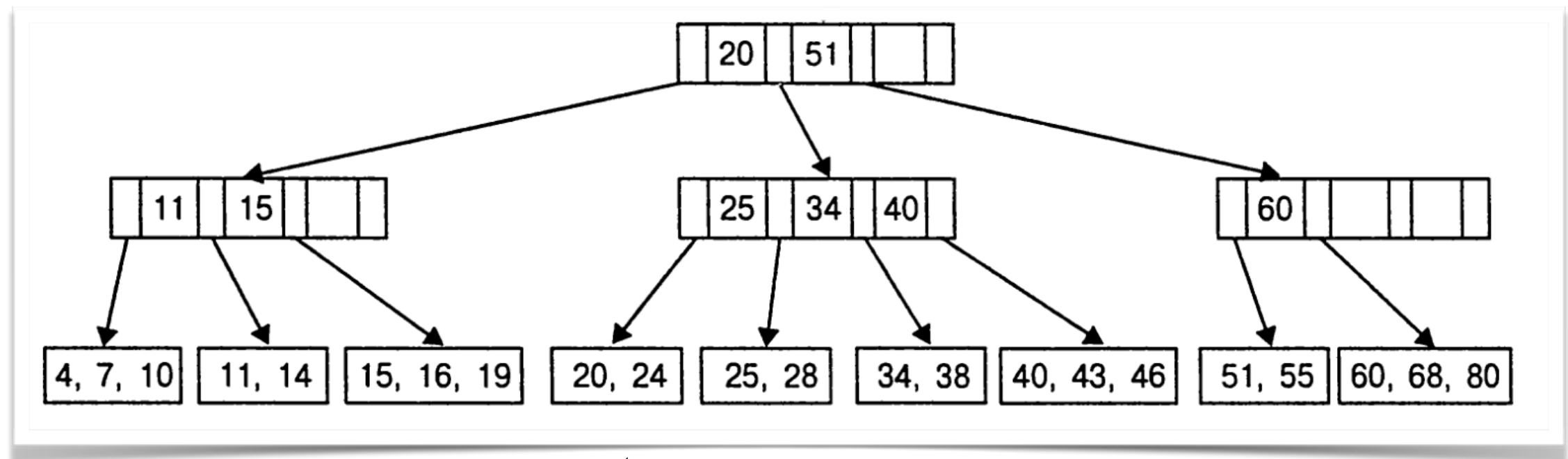


обобщение
2-3-деревя

В-дерево

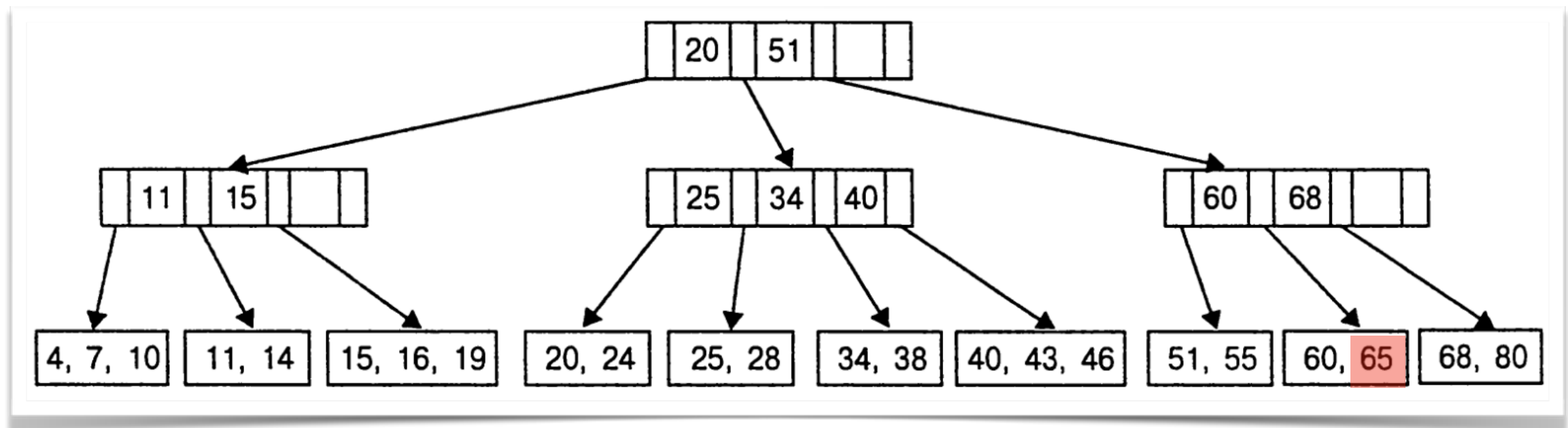


В-дерево



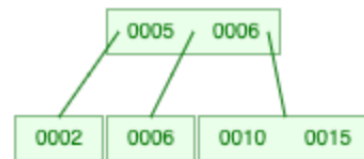
4-го порядка

В-дерево



пример
вставки 65

B-Trees

 Insert Delete Find Print Clear ☐ Max. Degree = 3 ☐ Preemptive Split☒ Max. Degree = 4☐ Max. Degree = 5☐ Max. Degree = 6☐ Max. Degree = 7

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

