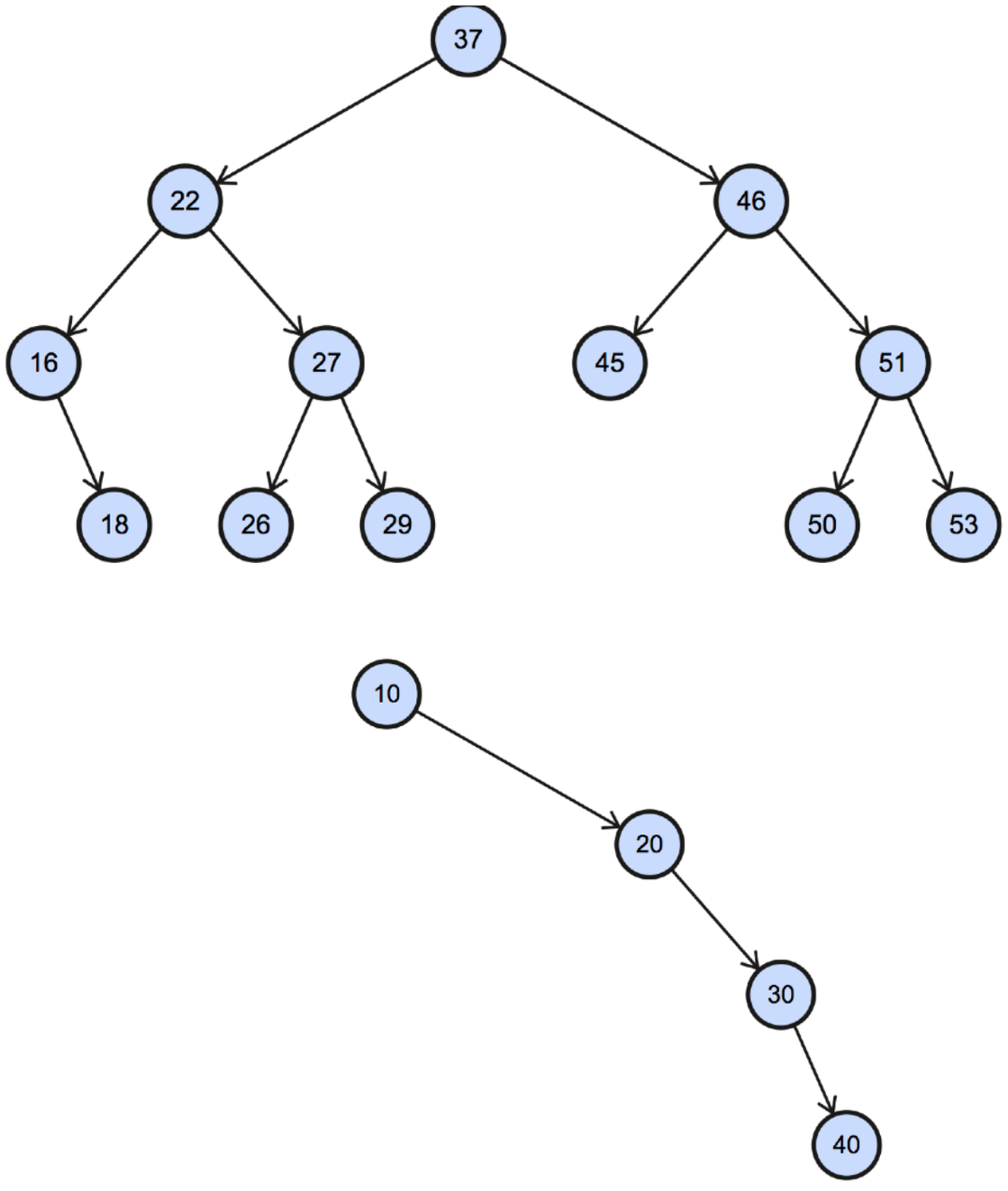
Бинарные деревья поиска. Продолжение.

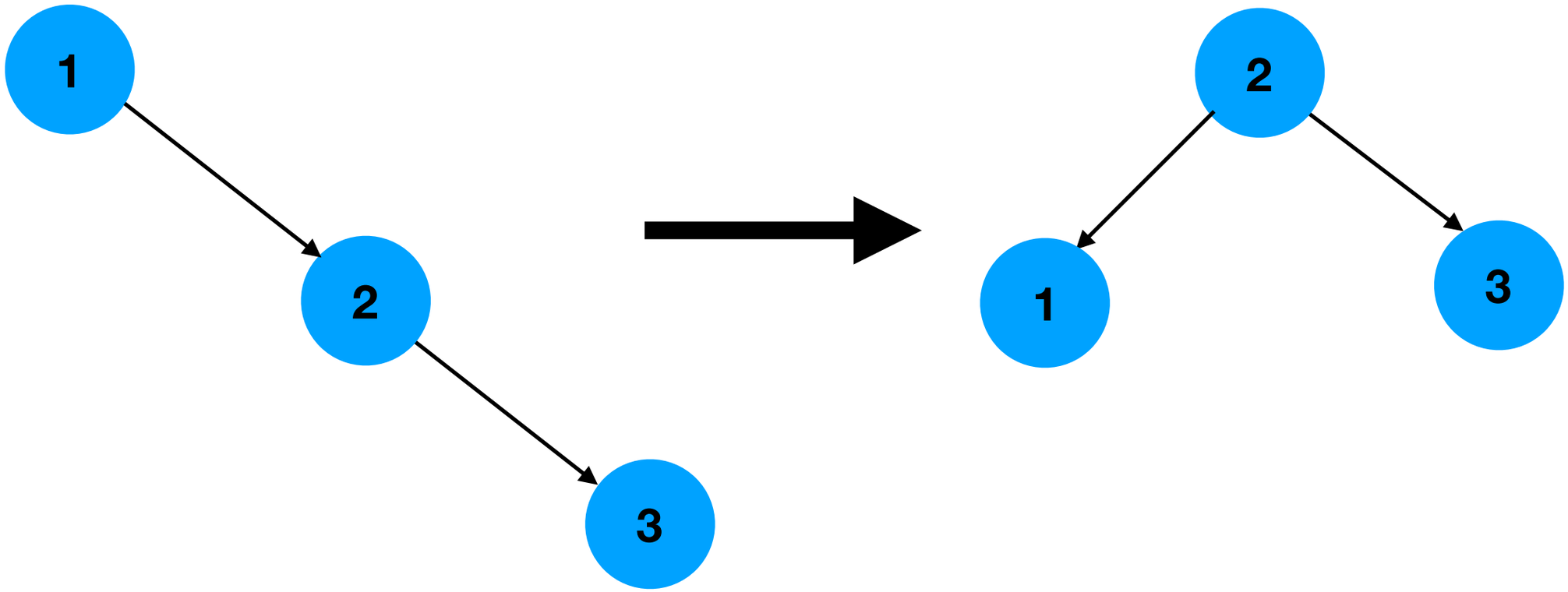
Время работы



Операция	Время работы в среднем случае	Время работы в худшем случае
Вставка	O(logN)	O(N)
Поиск	O(logN)	O(N)
Удаление	O(logN)	O(N)

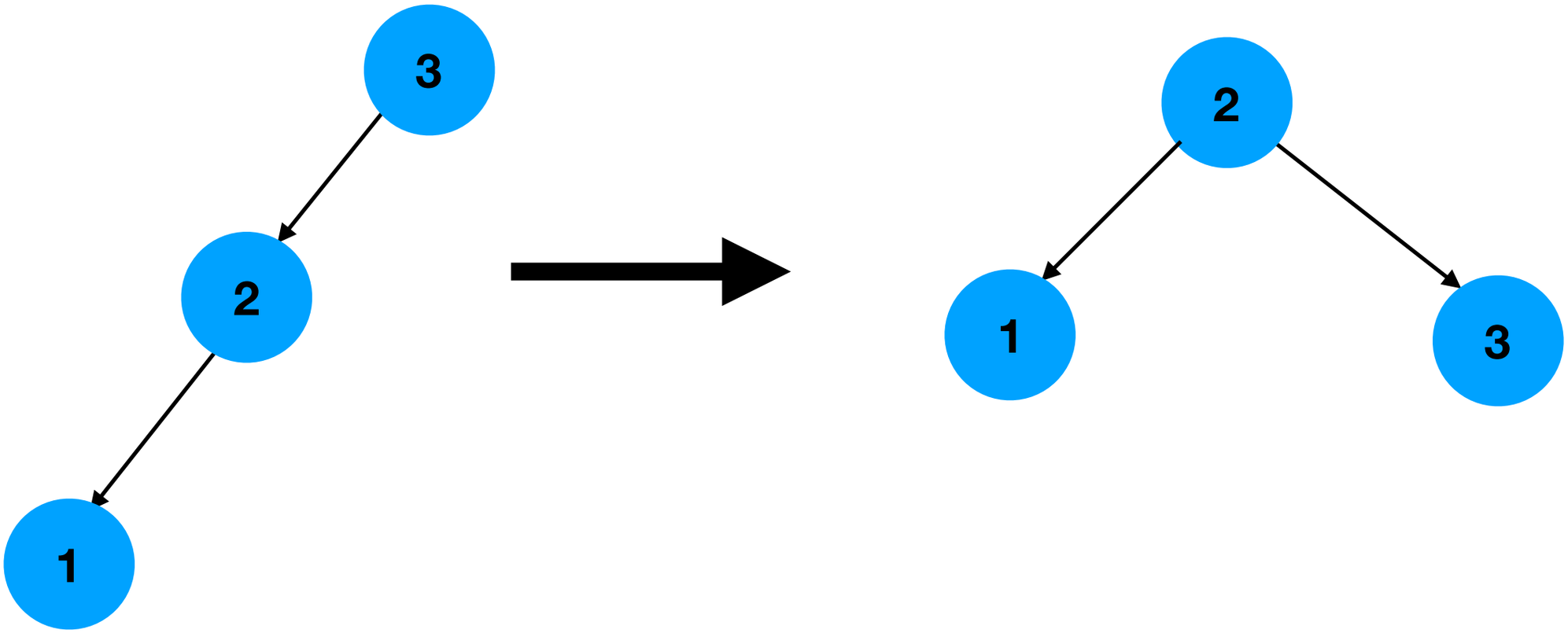
Сбалансированные бинарные деревья поиска

Идея - будем детектировать плохие ветви дерева и как-то их править

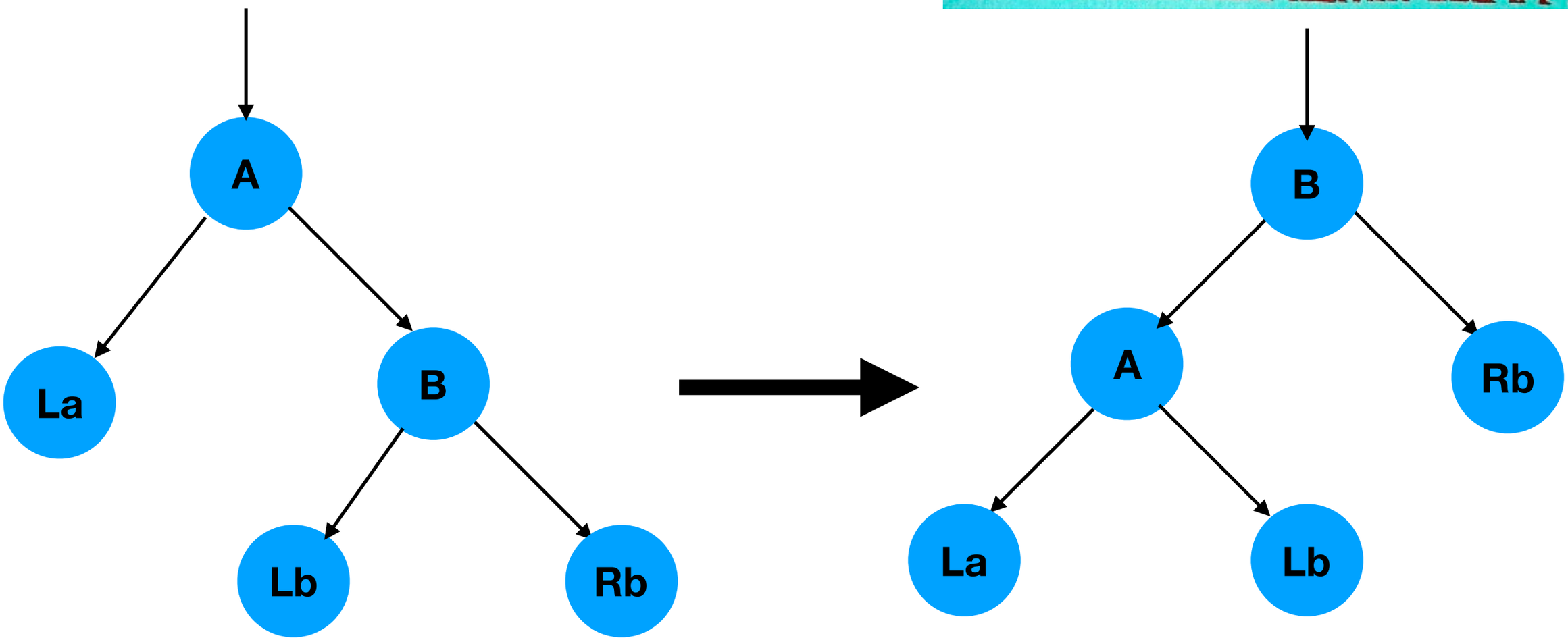


Сбалансированные бинарные деревья поиска

Идея - будем детектировать плохие ветви дерева и как-то их править

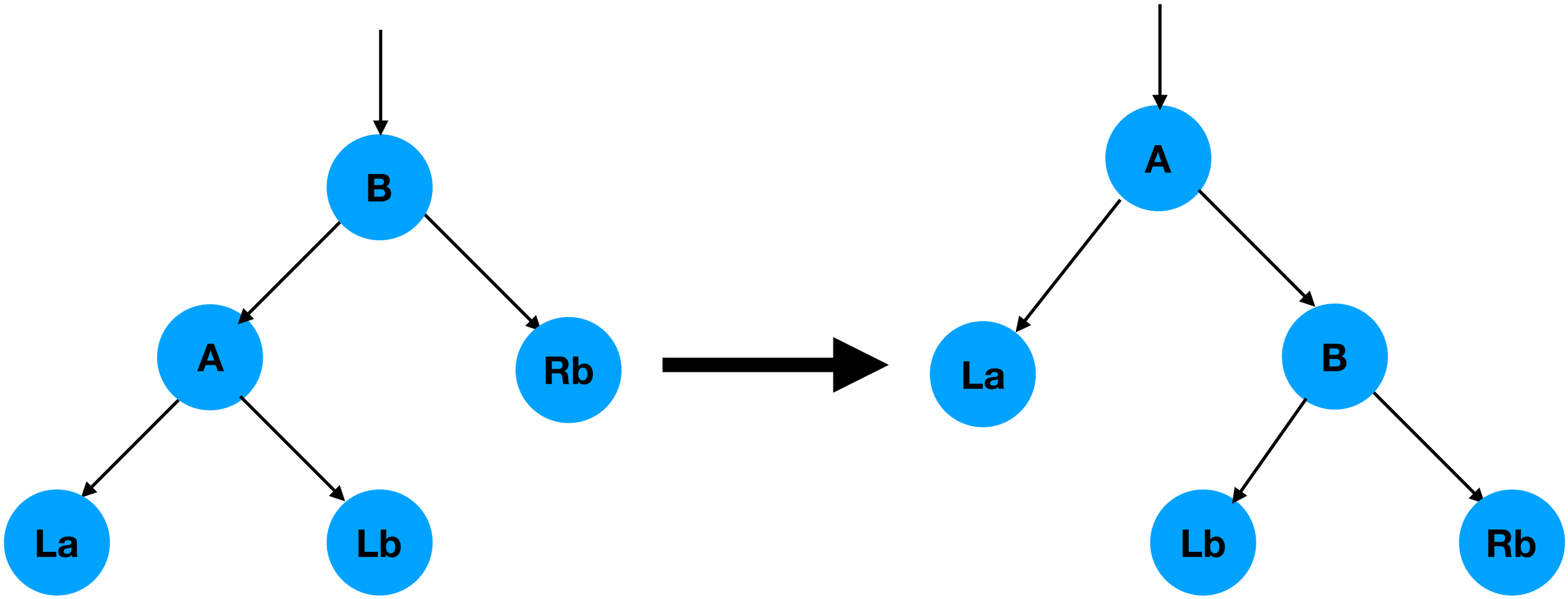


Левый поворот



Увеличиваем глубину слева

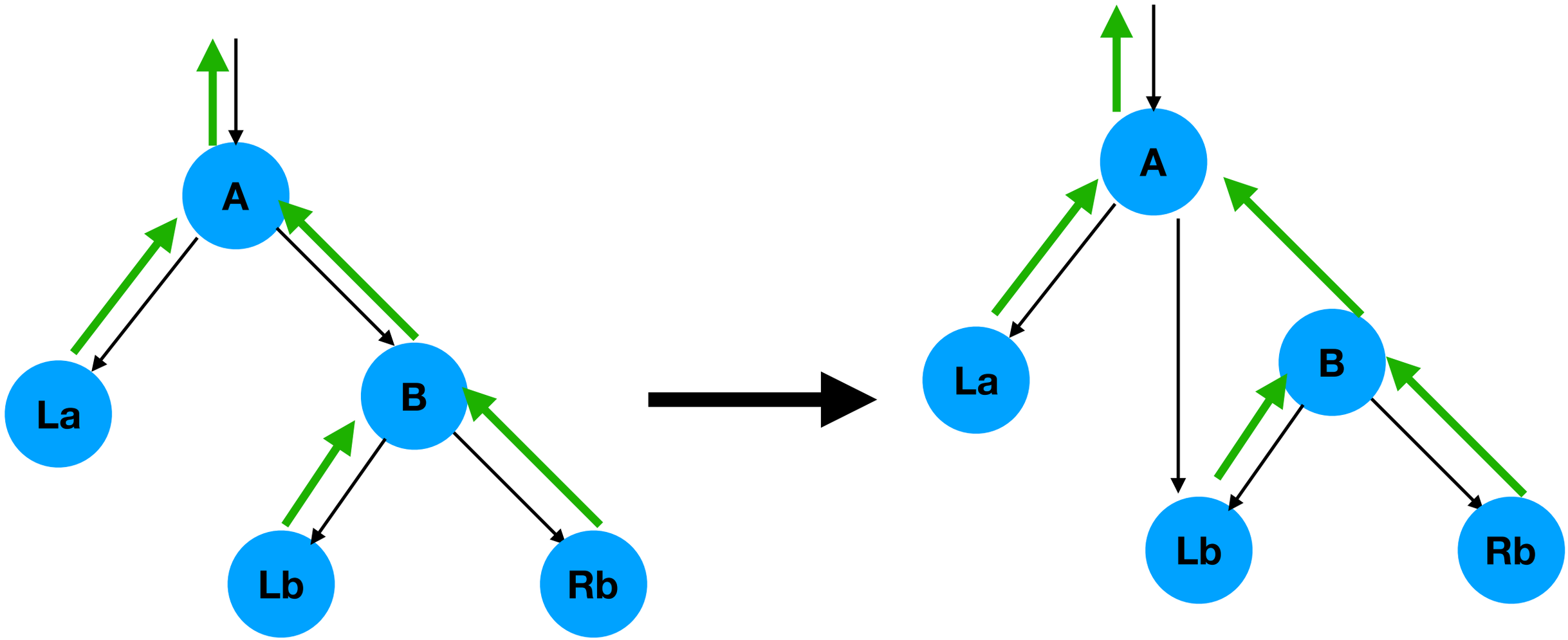
Правый поворот



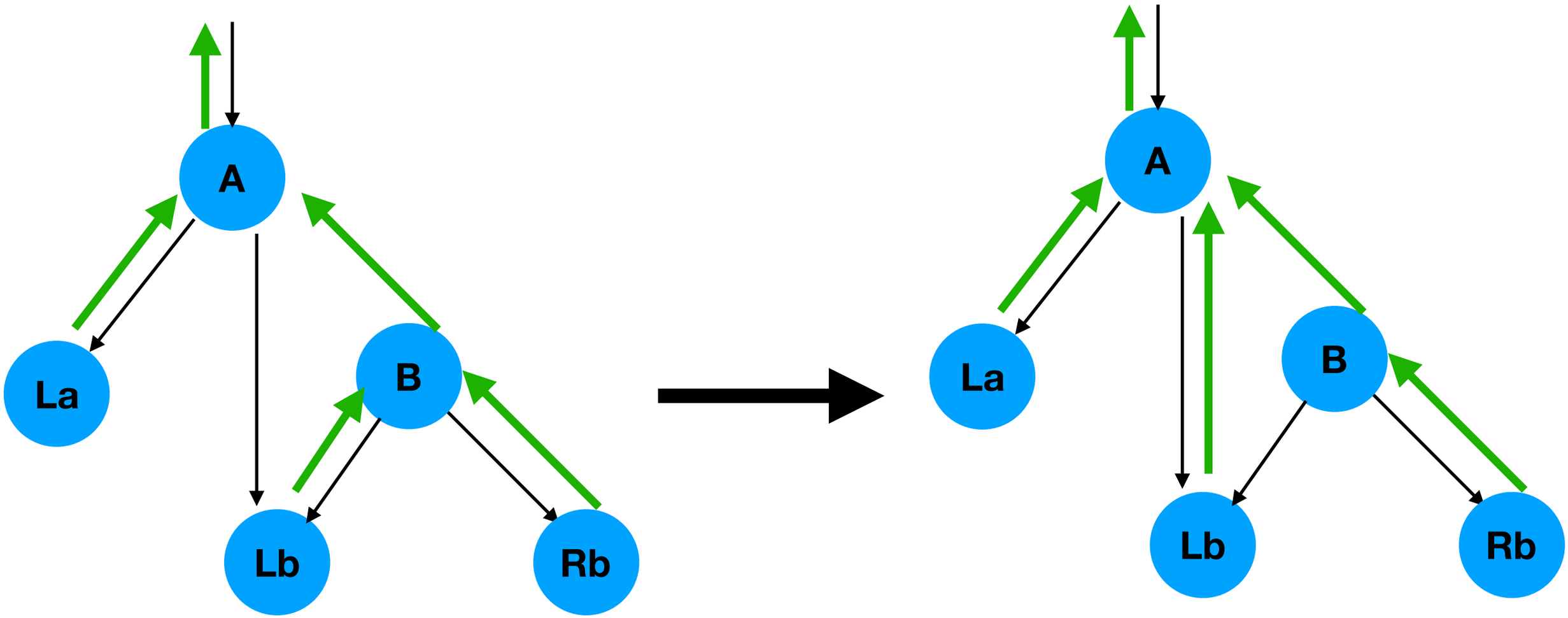
Увеличиваем глубину справа

Пишем псевдокод одного из поворотов

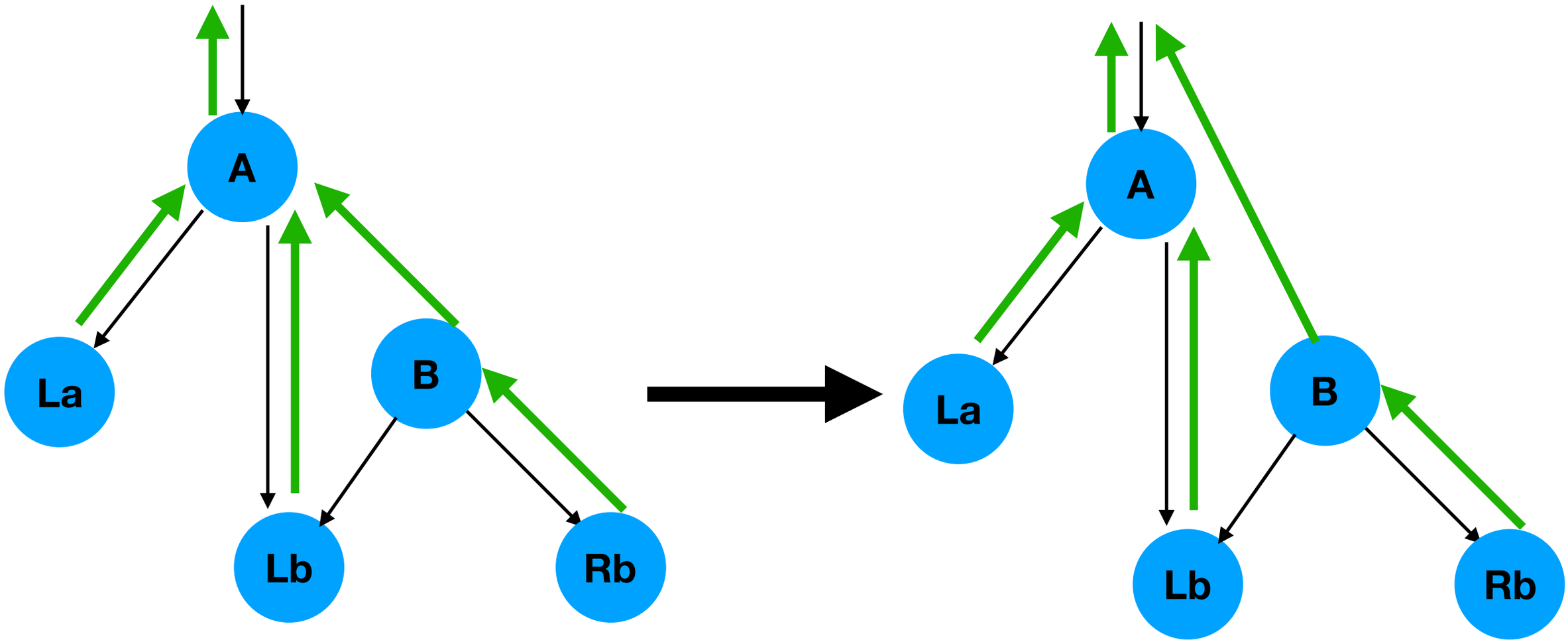
Вы спросите: какой уклон хуже? Нельзя так ставить вопрос. Оба они хуже, и первый и второй уклоны. Иосиф Сталин



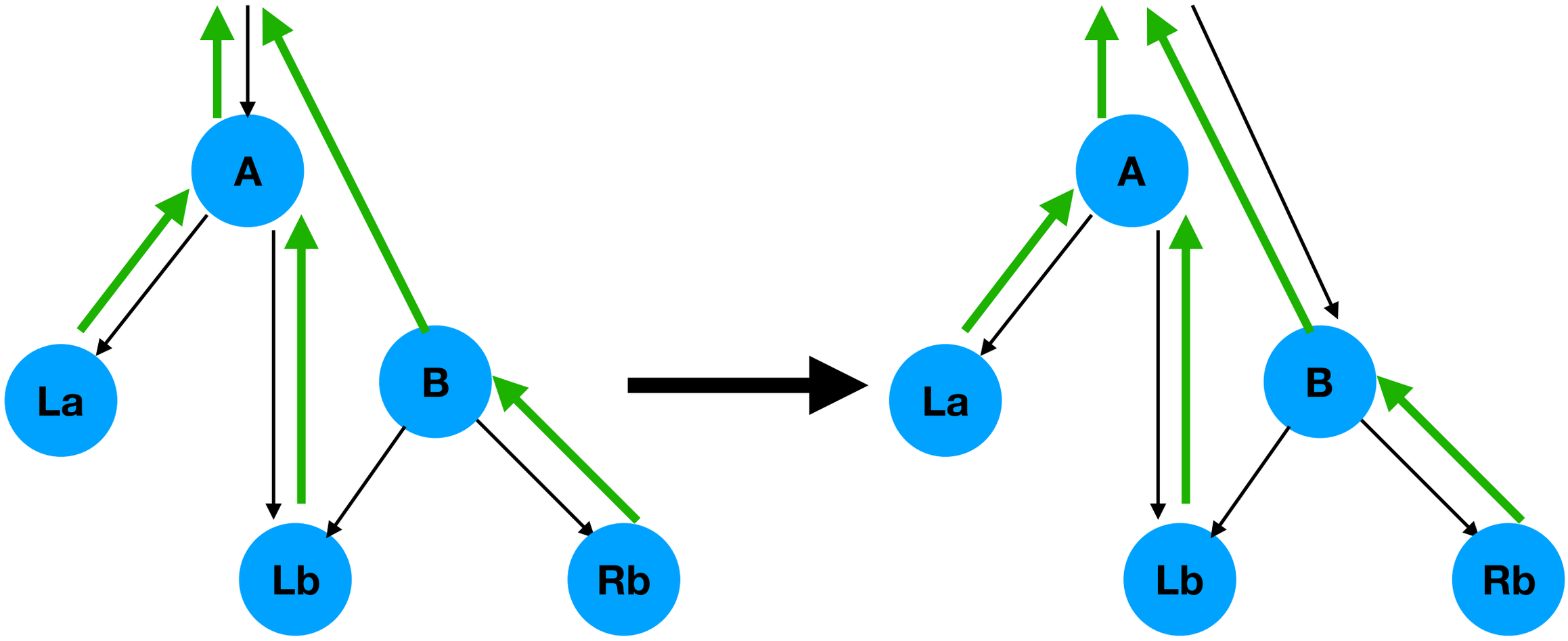
Пишем псевдокод одного из поворотов



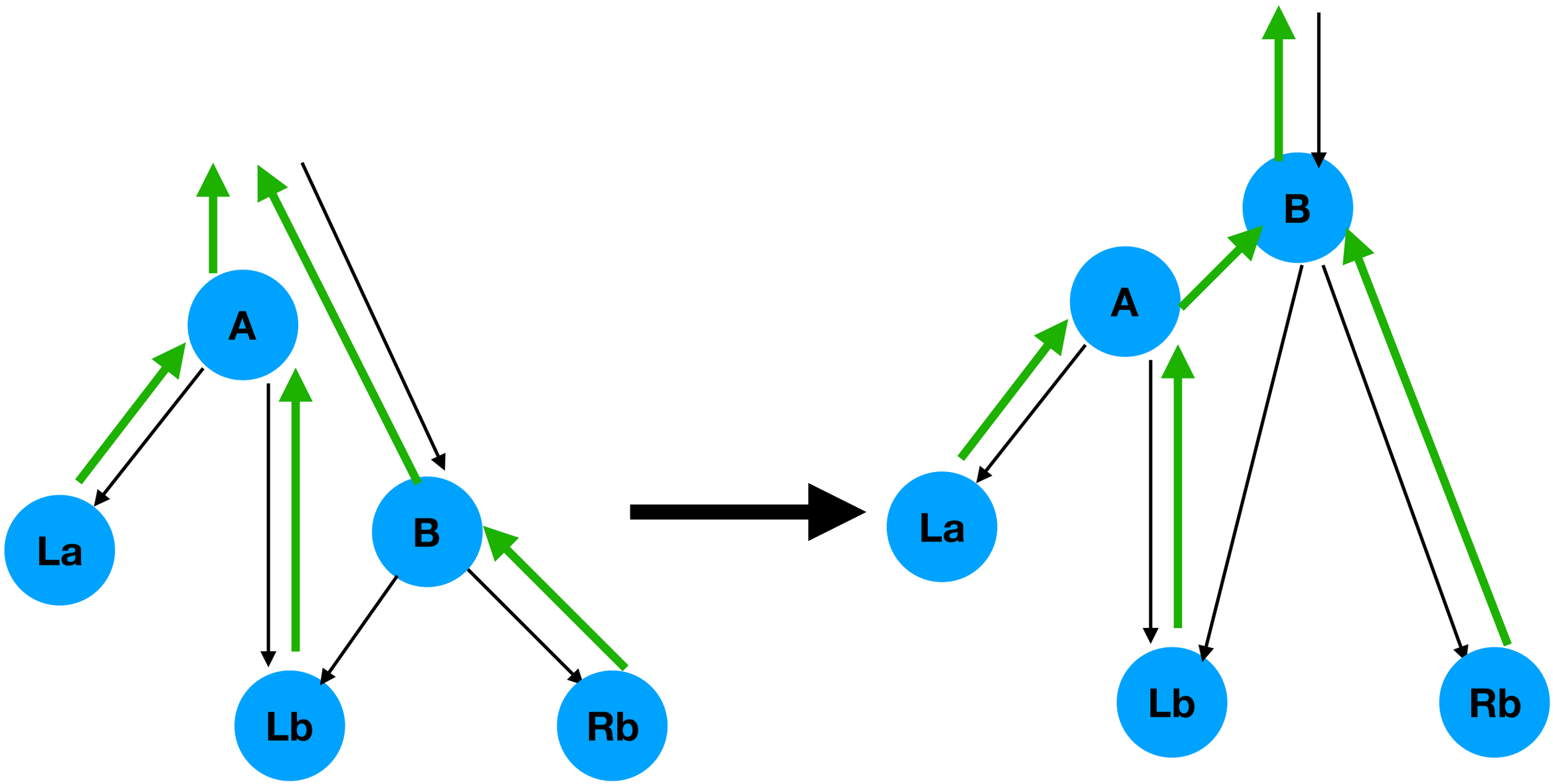
Пишем псевдокод одного из поворотов



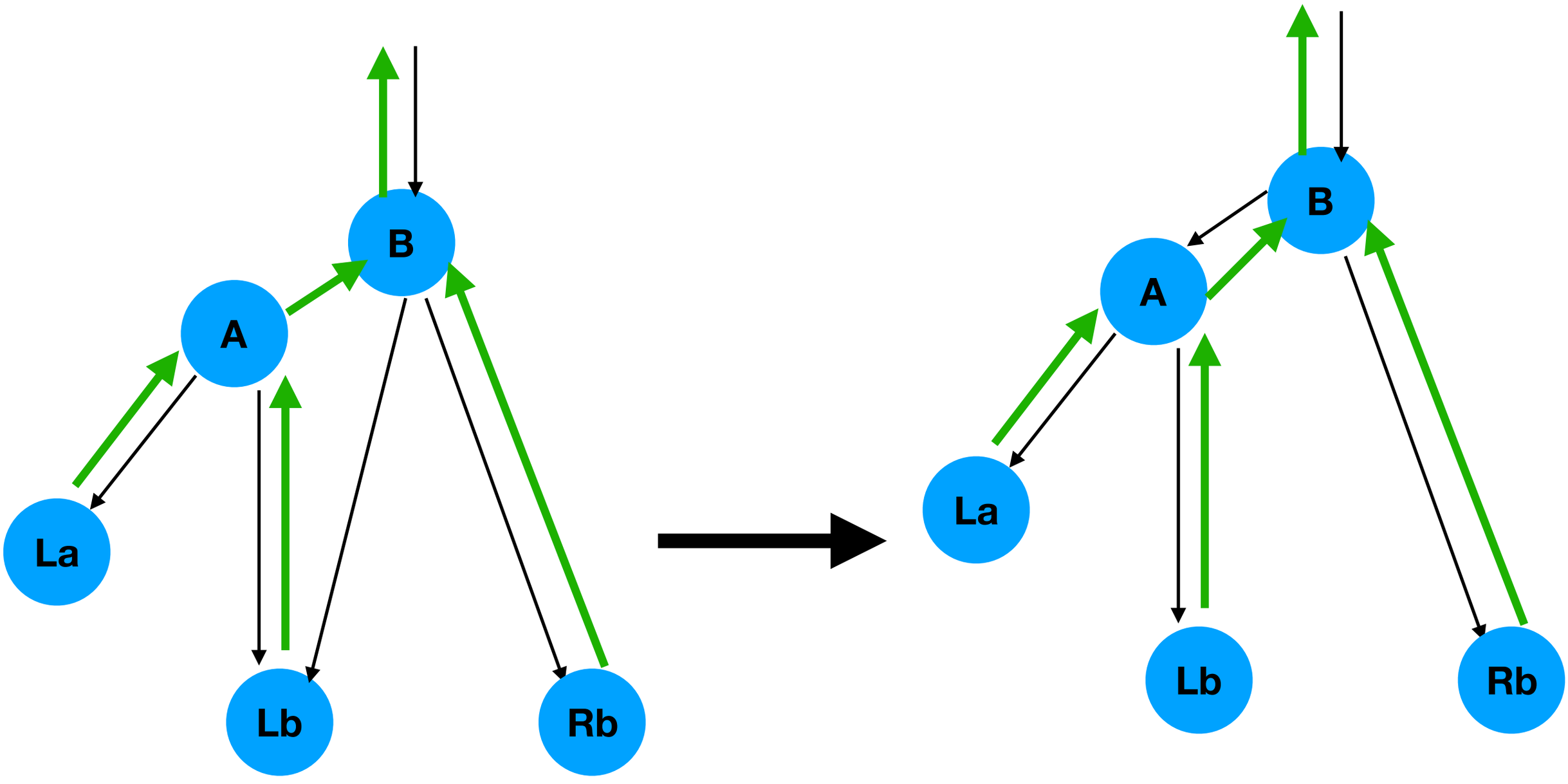
Пишем псевдокод одного из поворотов



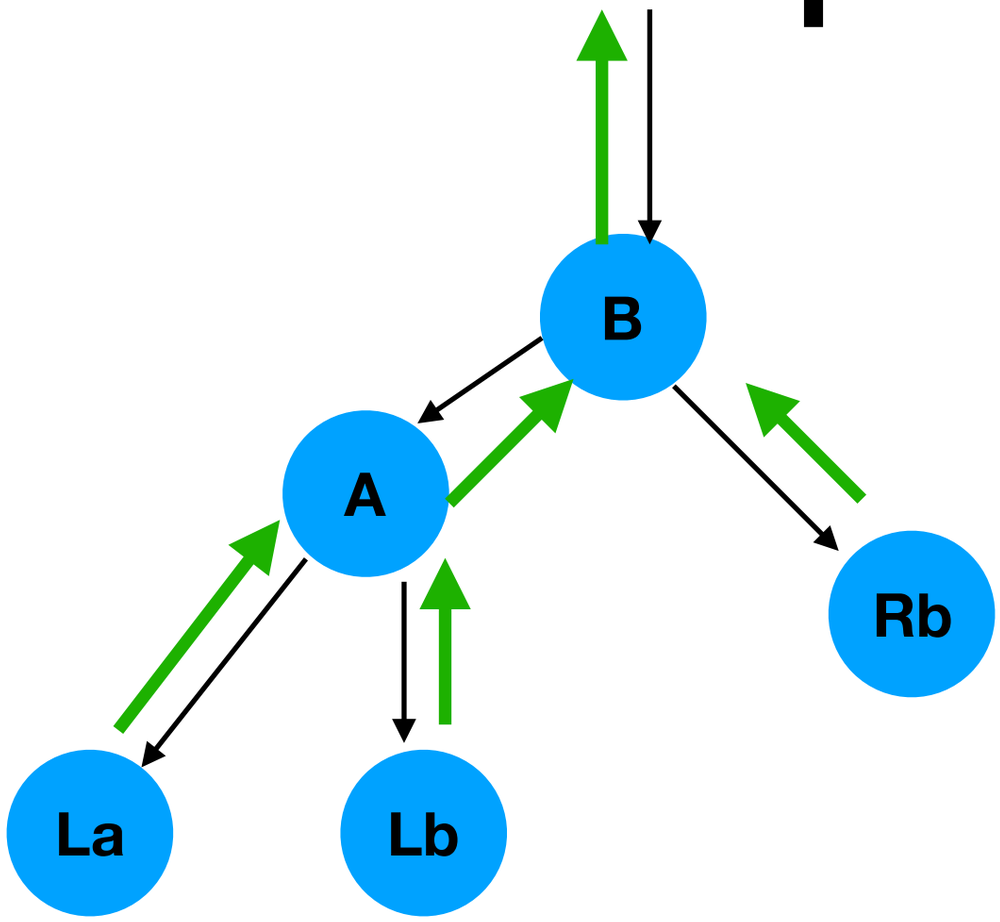
Пишем псевдокод одного из поворотов



Пишем псевдокод одного из поворотов



Пишем псевдокод одного из поворотов



Пишем псевдокод одного из поворотов

```
fn leftRotate(tree: Tree, node: Node) -> None {
    new_subroot = node.right;
    node.right = new_subroot.left;
    if (new_subroot.left != None){
        new_subroot.left.parent = node;
    }

    new_subroot.parent = node.parent;
    if (node.parent == None){
        root = new_subroot;
    }else{
        if node.parent.left == node{
            node.parent.left == new_subroot;
        }else{
            node.parent.right = new_subroot;
        }
    }

    node.parent = new_subroot;
    new_subroot.left = node;
}
```

Сбалансированные бинарные деревья поиска

- АВЛ-деревья - для каждой вершины АВЛ-дерева высота её двух поддеревьев различается не более чем на 1. Самый первый метод. Высота очень близка к высоте идеального бинарного дерева поиска

- **Красно-черные деревья** - разберем сейчас. Используются в ядре линукс.

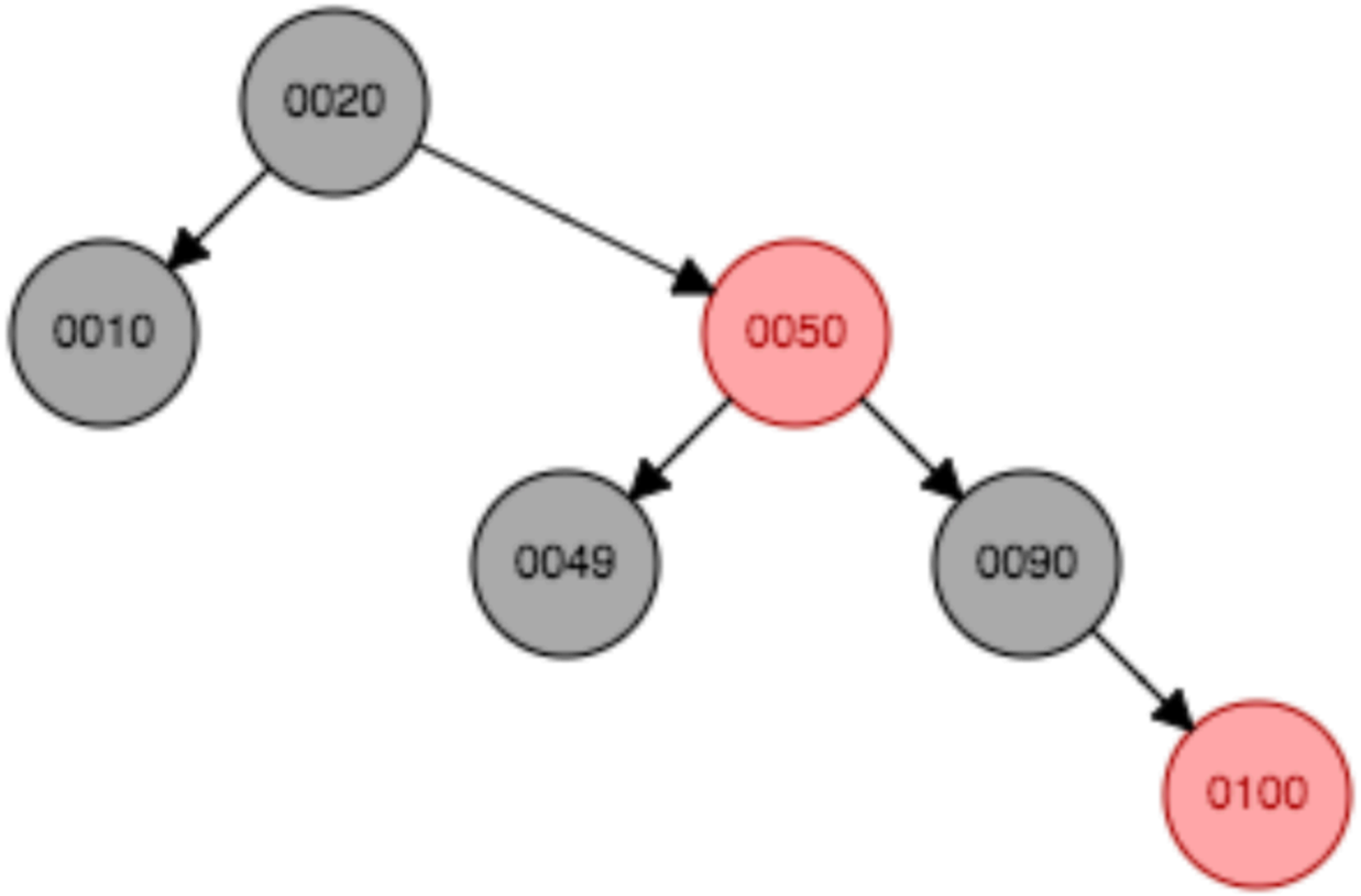
- Spacegoat деревья - простая реализация, не требующая дополнительной памяти. Можно ускорять одни операции в ущерб скорости других

- Splay-дерево - быстрее ищем недавно использованные элементы

Красно-черное дерево

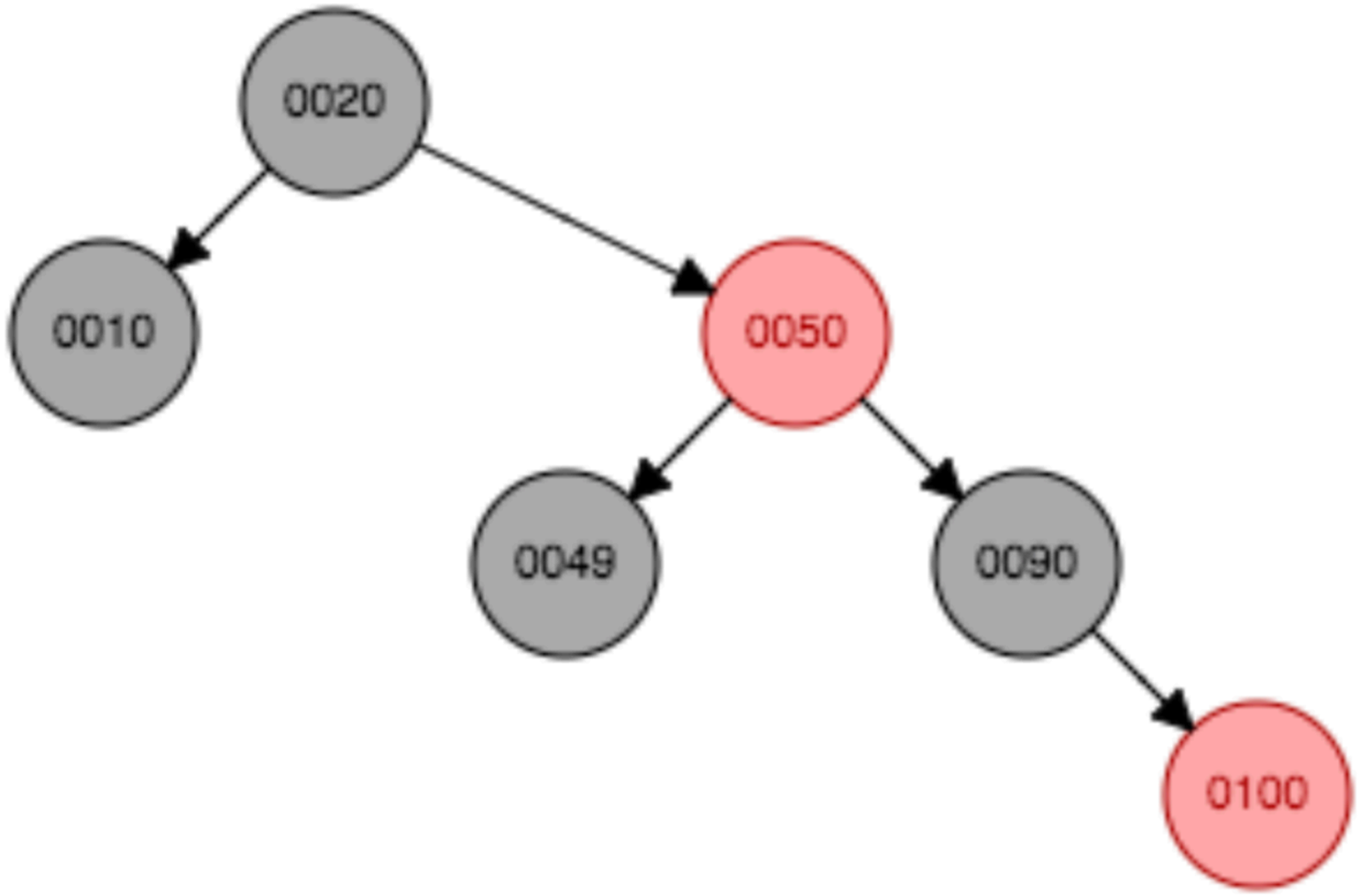
1. Каждая вершина может быть либо красной, либо черной. Бесцветных вершин или вершин другого цвета быть не может.
2. Если вершина красная, то оба ее потомка - черные
3. Все пути от корня к листьям содержат одинаковое число черных вершин
4. Корень имеет черный цвет
5. *Каждый лист имеет черный цвет - разберем попозже, пока игнорируем

Красно-черное дерево



Высота может отличаться от высоты идеально сбалансированного в два раза

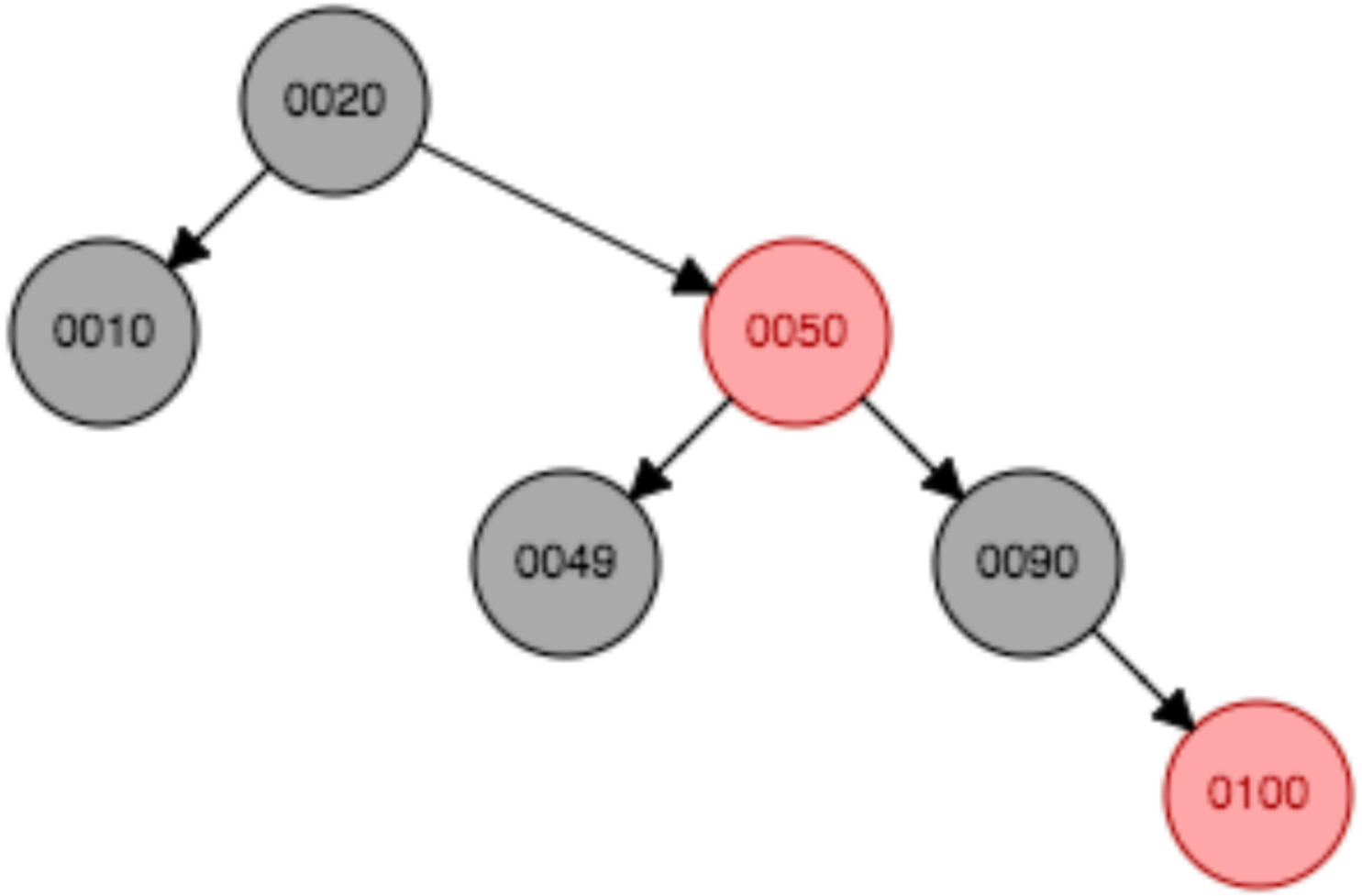
Красно-черное дерево



Высота может отличаться от высоты идеально сбалансированного в два раза

Какого цвета любое вставляемое значение в начале вставки?

Красно-черное дерево

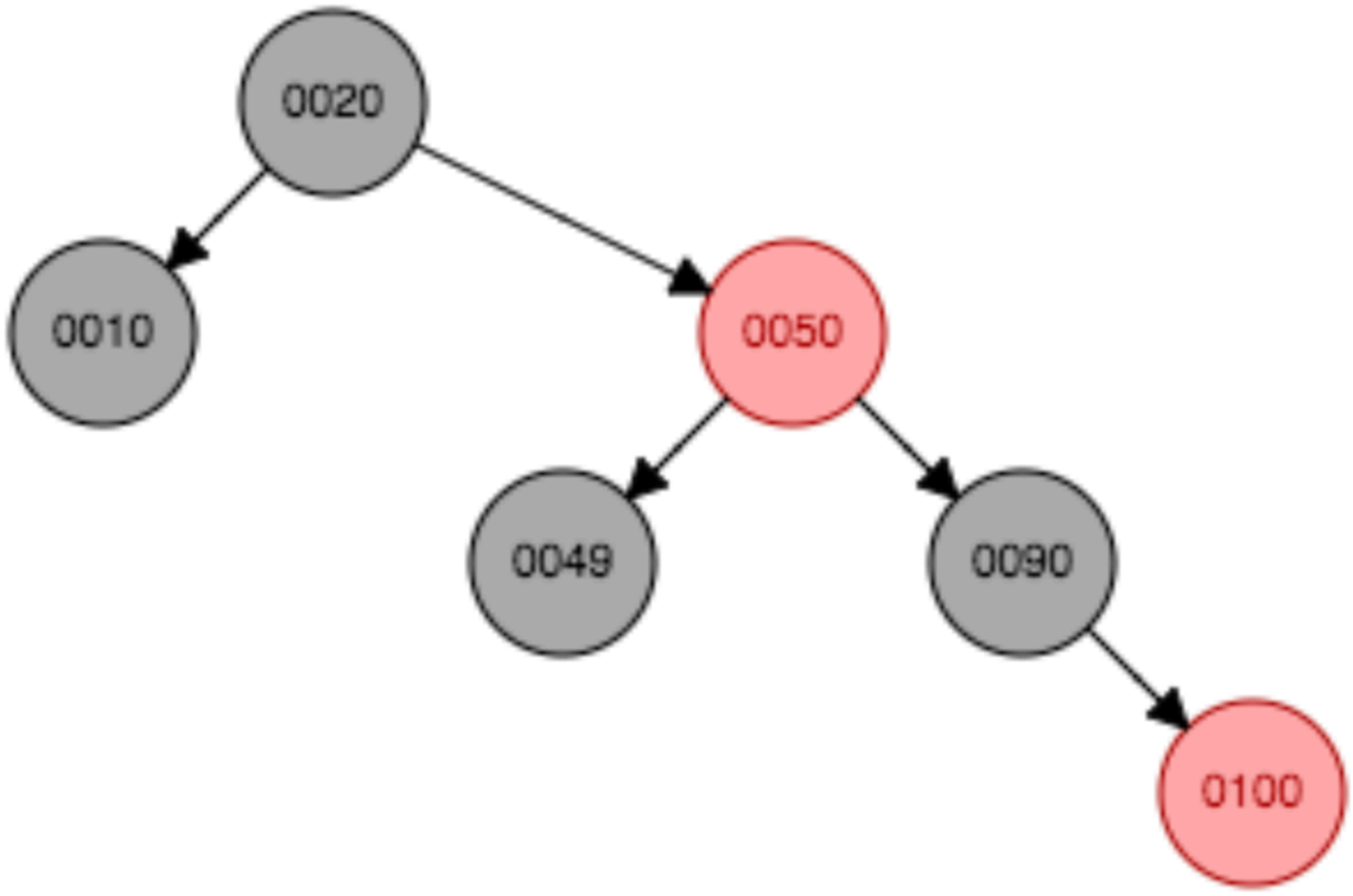


Высота может отличаться от высоты идеально сбалансированного в два раза

Какого цвета любое вставляемое значение в начале вставки?

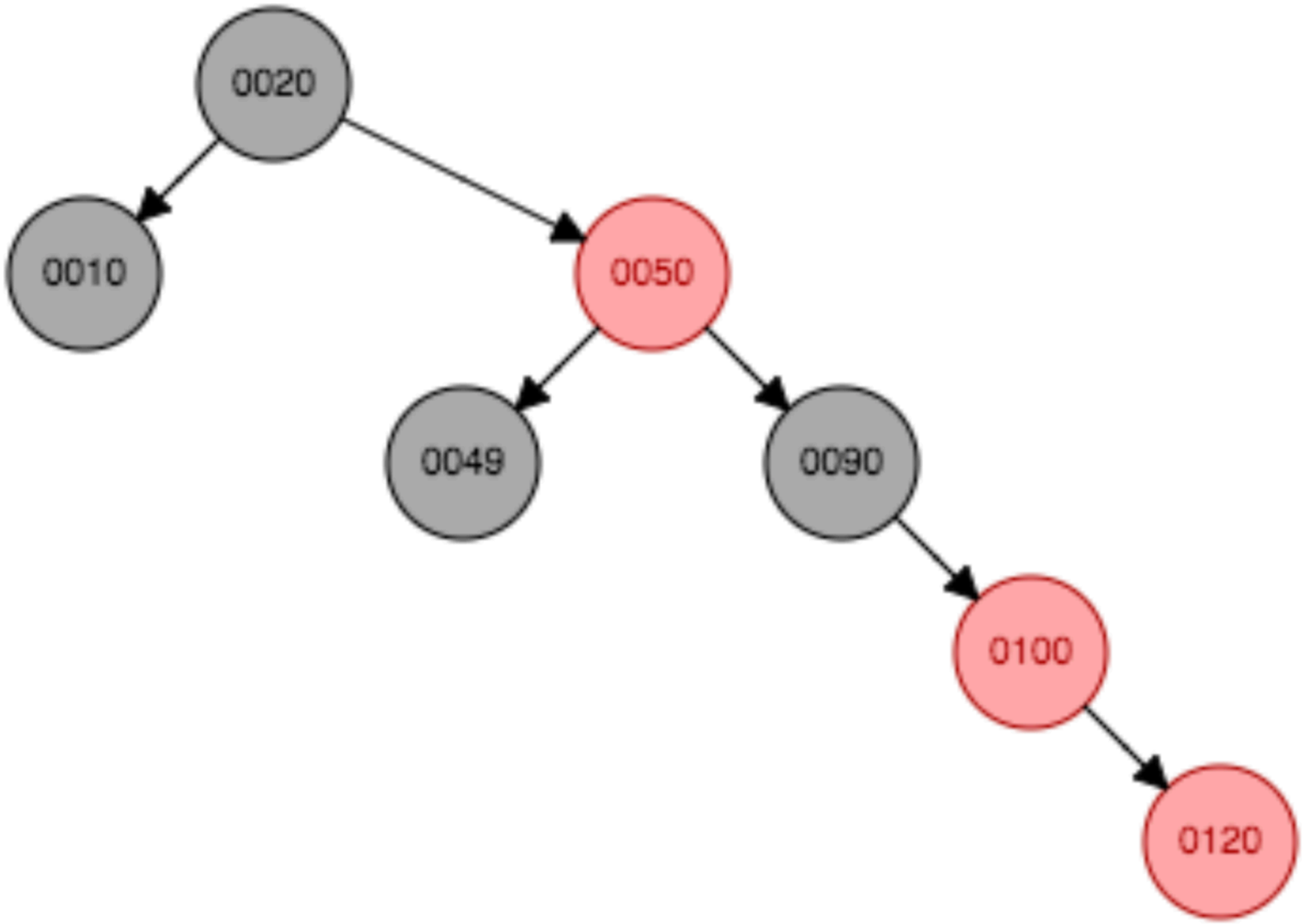
Красный

Красно-черное дерево

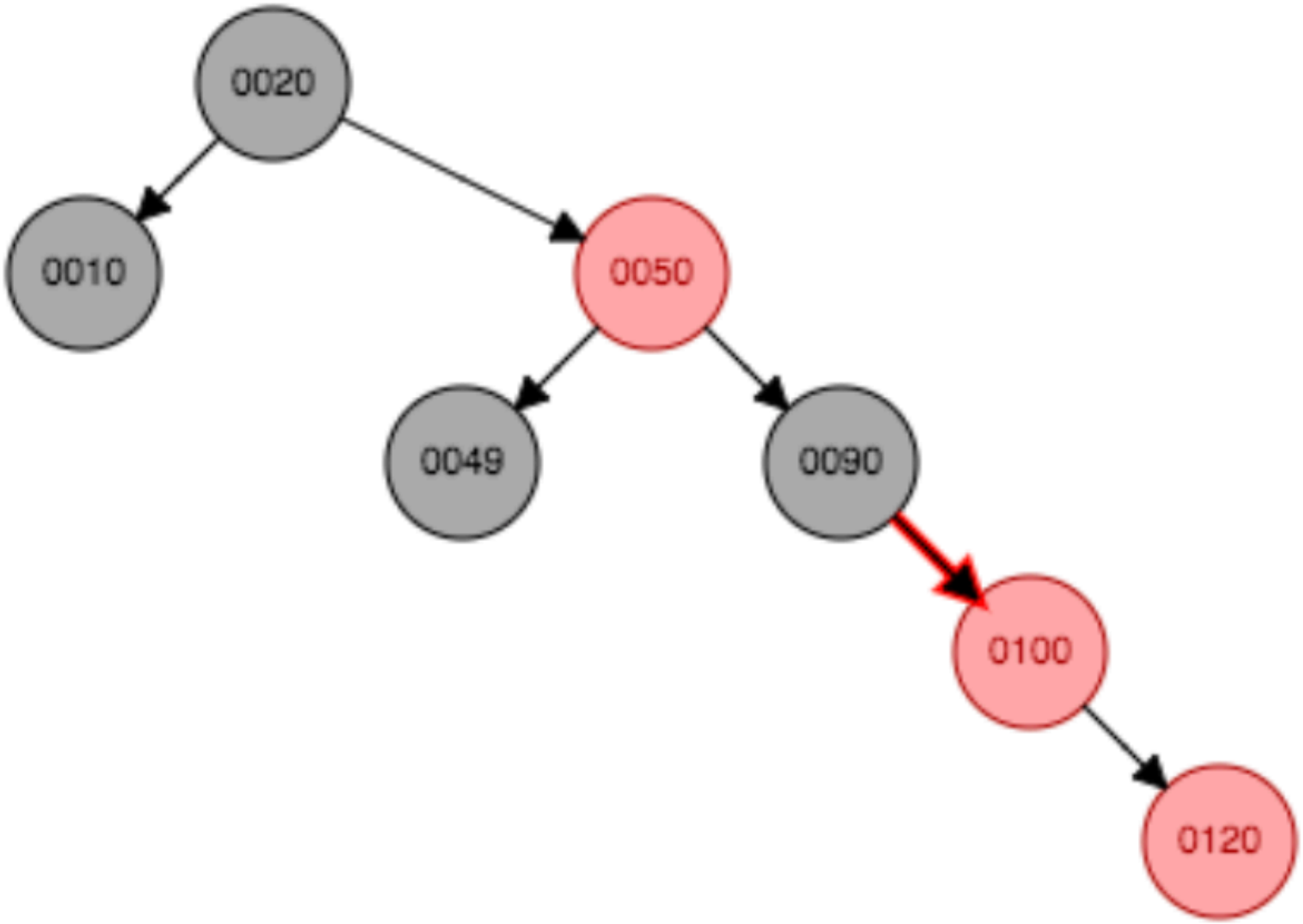


https://www.cs.usfca.edu/~galles/visualization/RedBlack.html

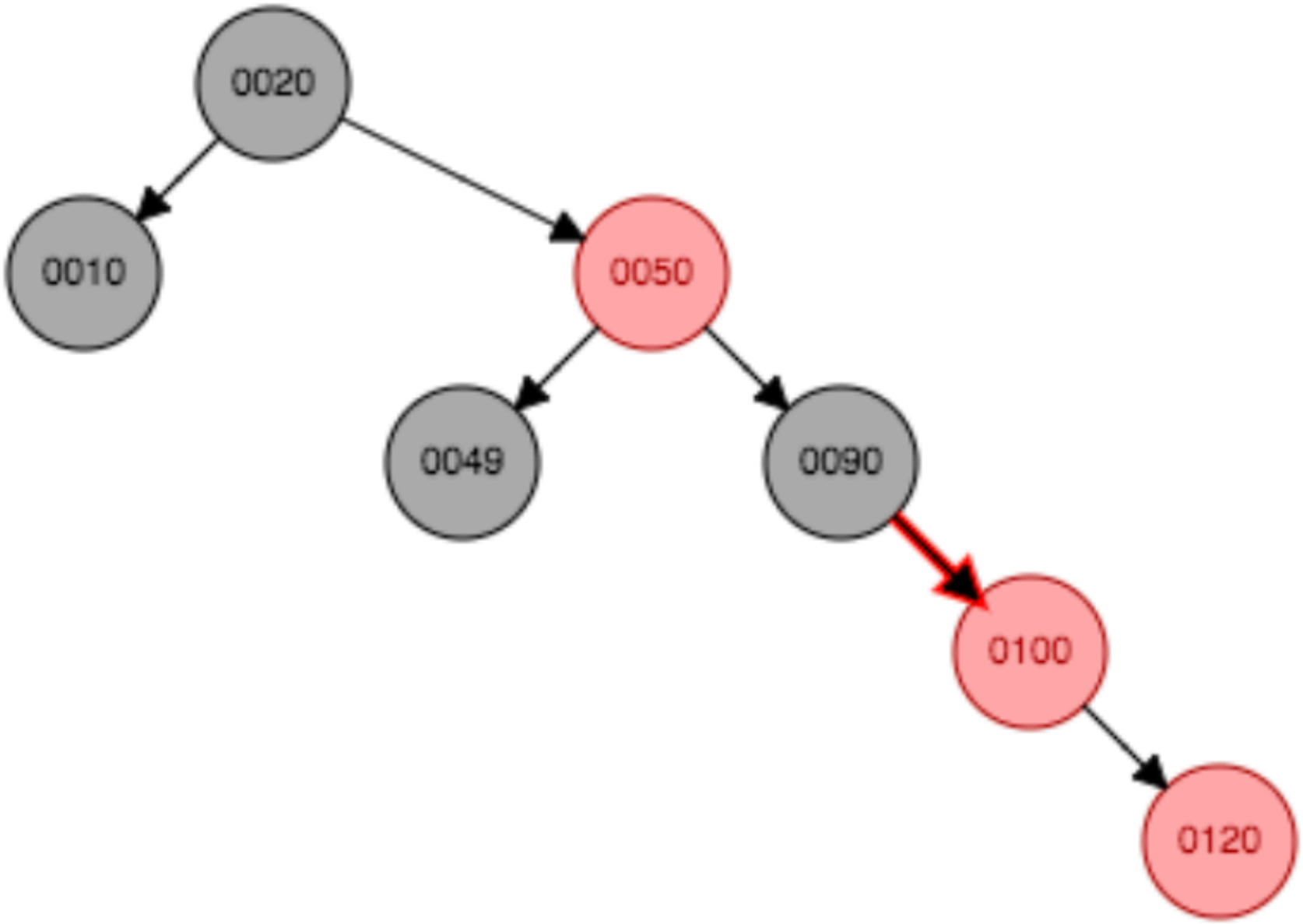
Красно-черное дерево



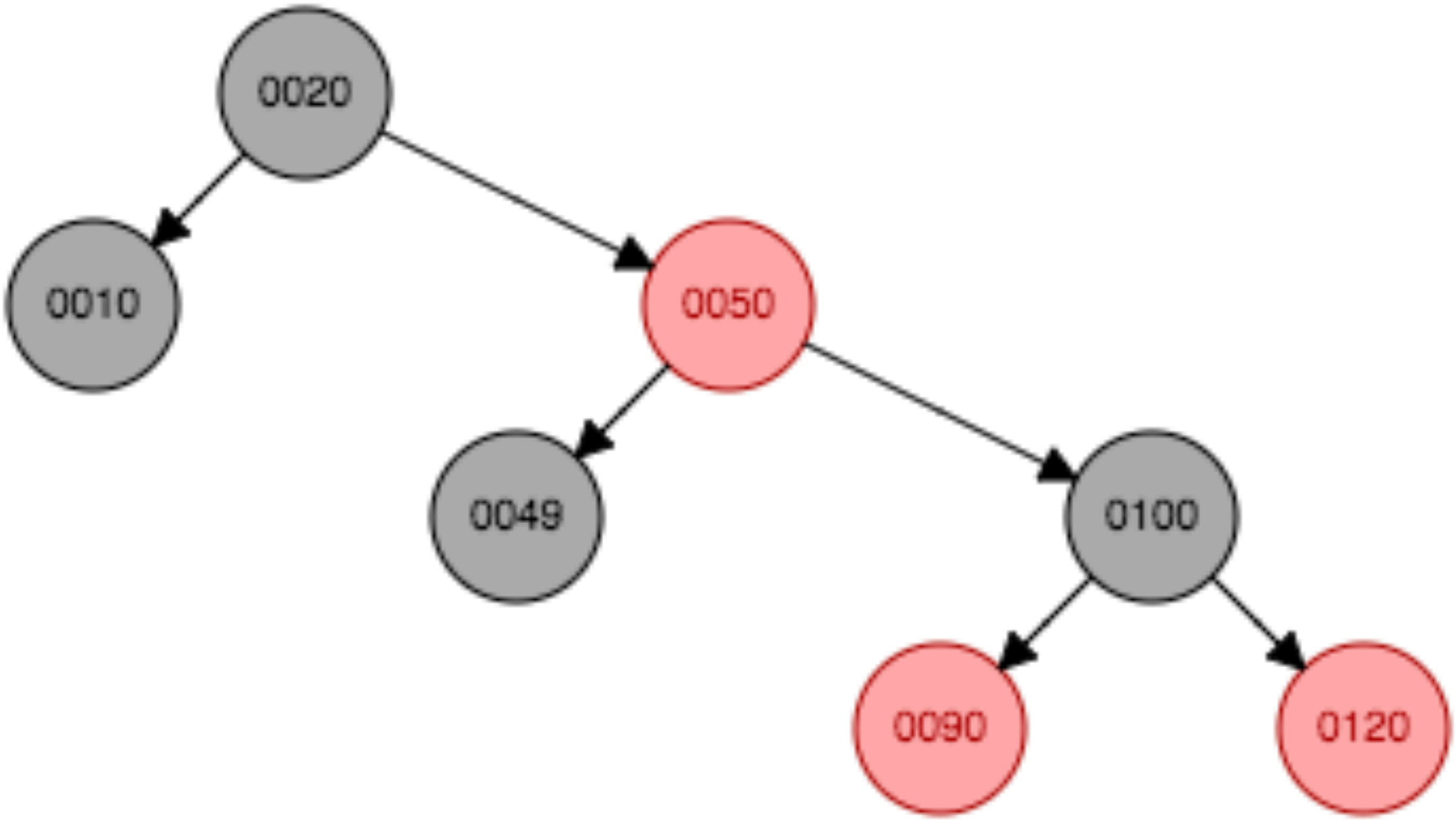
Красно-черное дерево



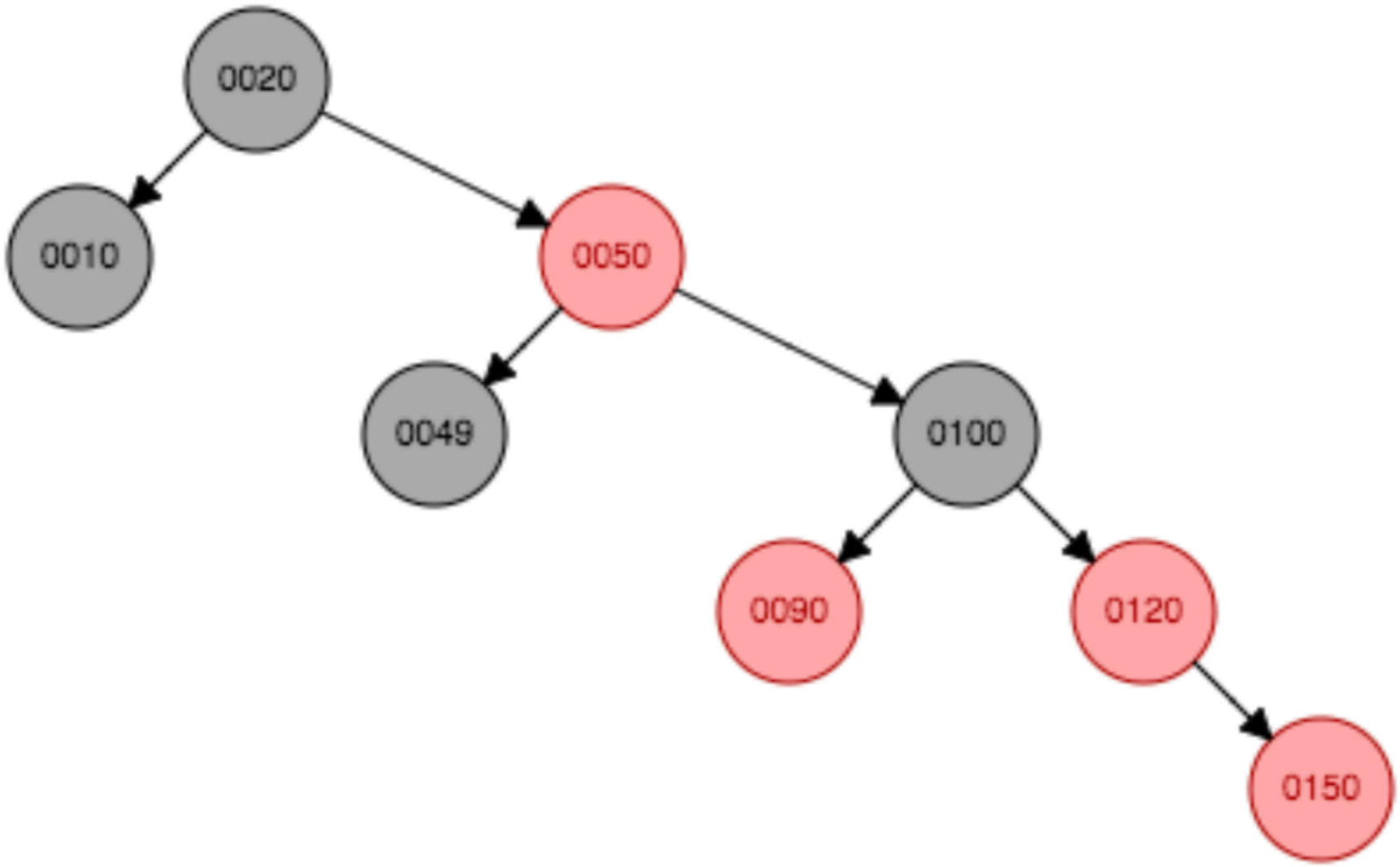
Красно-черное дерево



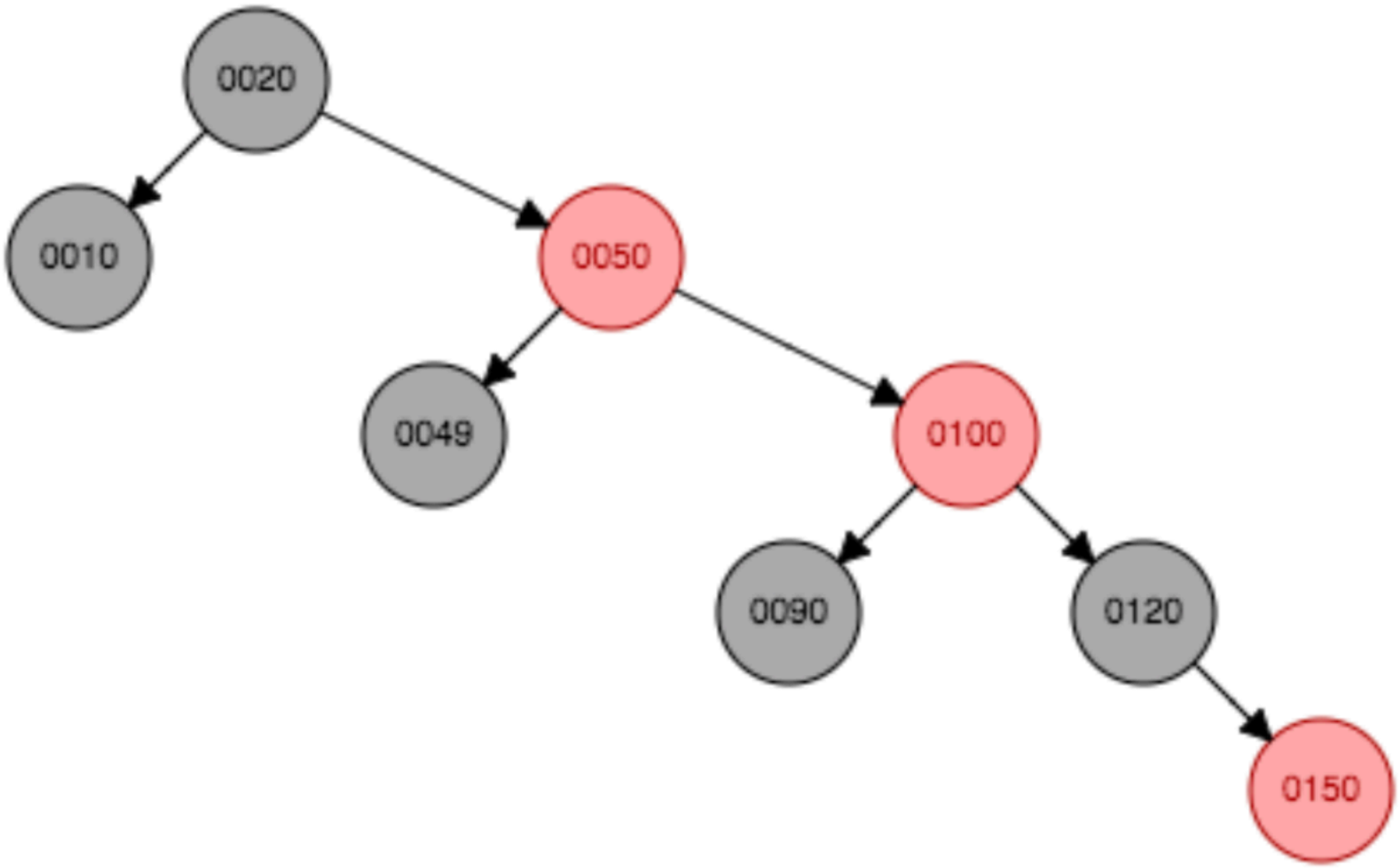
Красно-черное дерево



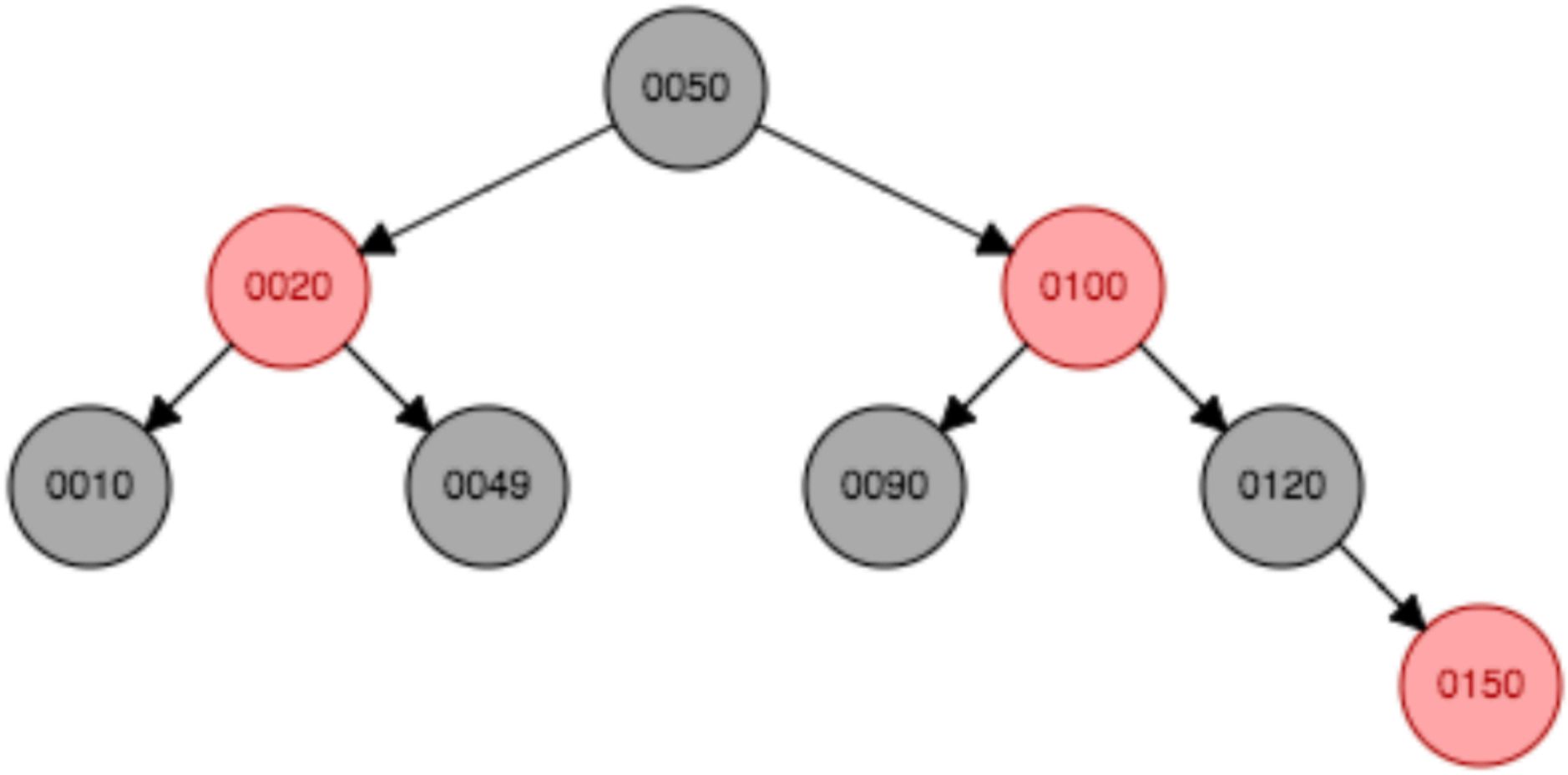
Красно-черное дерево



Красно-черное дерево



Красно-черное дерево



В чем неудобство данного псевдокода?

```
fn leftRotate(tree: Tree, node: Node) -> None {
    new_subroot = node.right;
    node.right = new_subroot.left;
    if (new_subroot.left != None){
        new_subroot.left.parent = node;
    }

    new_subroot.parent = node.parent;
    if (node.parent == None){
        root = new_subroot;
    }else{
        if node.parent.left == node{
            node.parent.left == new_subroot;
        }else{
            node.parent.right = new_subroot;
        }
    }

    node.parent = new_subroot;
    new_subroot.left = node;
}
```

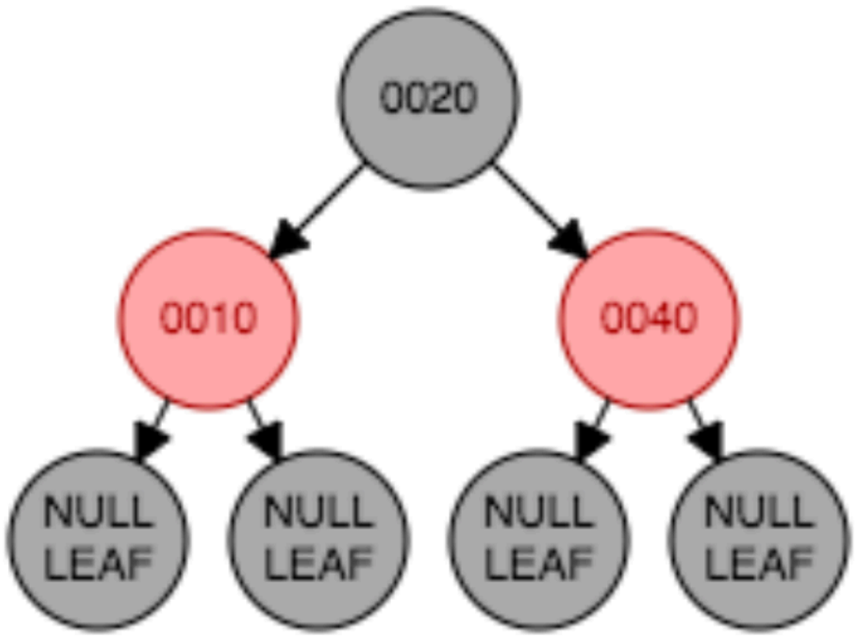
Проверки на то, что у нас нет None.

В случае красно-черного дерева куча проверок на то, что не является ли левый/правый потом узла пустым.

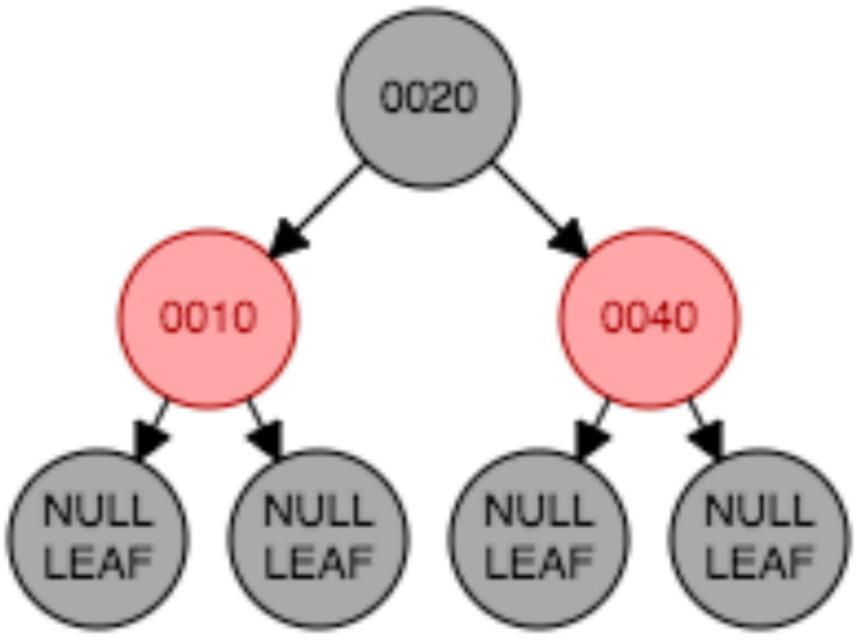
Красно-черное дерево

5. *Каждый лист имеет черный цвет

- пусть у любого листа в дереве, хранящего значение есть фиктивные потомки черного цвета - NIL. Они ничего не хранят
- свойств дерева это не нарушит
- Пусть эти NIL работают так же, как обычные узлы. У них можно менять цвет и тд. Но любые операции с ними никак на них не влияют. То есть NIL.color = RED не возвращает ошибки, но и цвет NIL остается черным
- Так как NIL одинаковы по свойствам и не меняются кодом, то сделаем их все одним объектом.

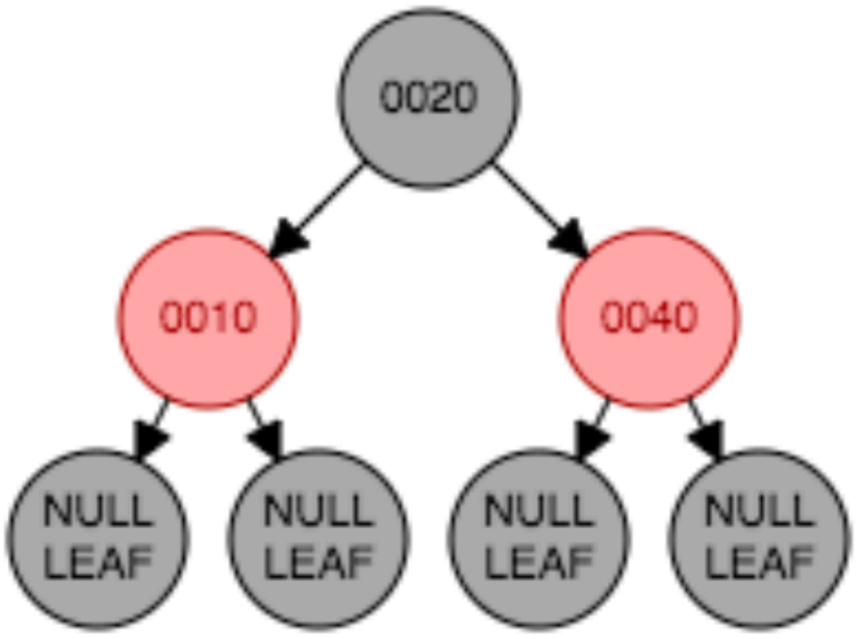


Красно-черное дерево



Операция	Время работы в среднем случае	Время работы в худшем случае
Вставка	O(logN)	O(logN)
Поиск	O(logN)	O(logN)
Удаление	O(logN)	O(logN)

Красно-черное дерево



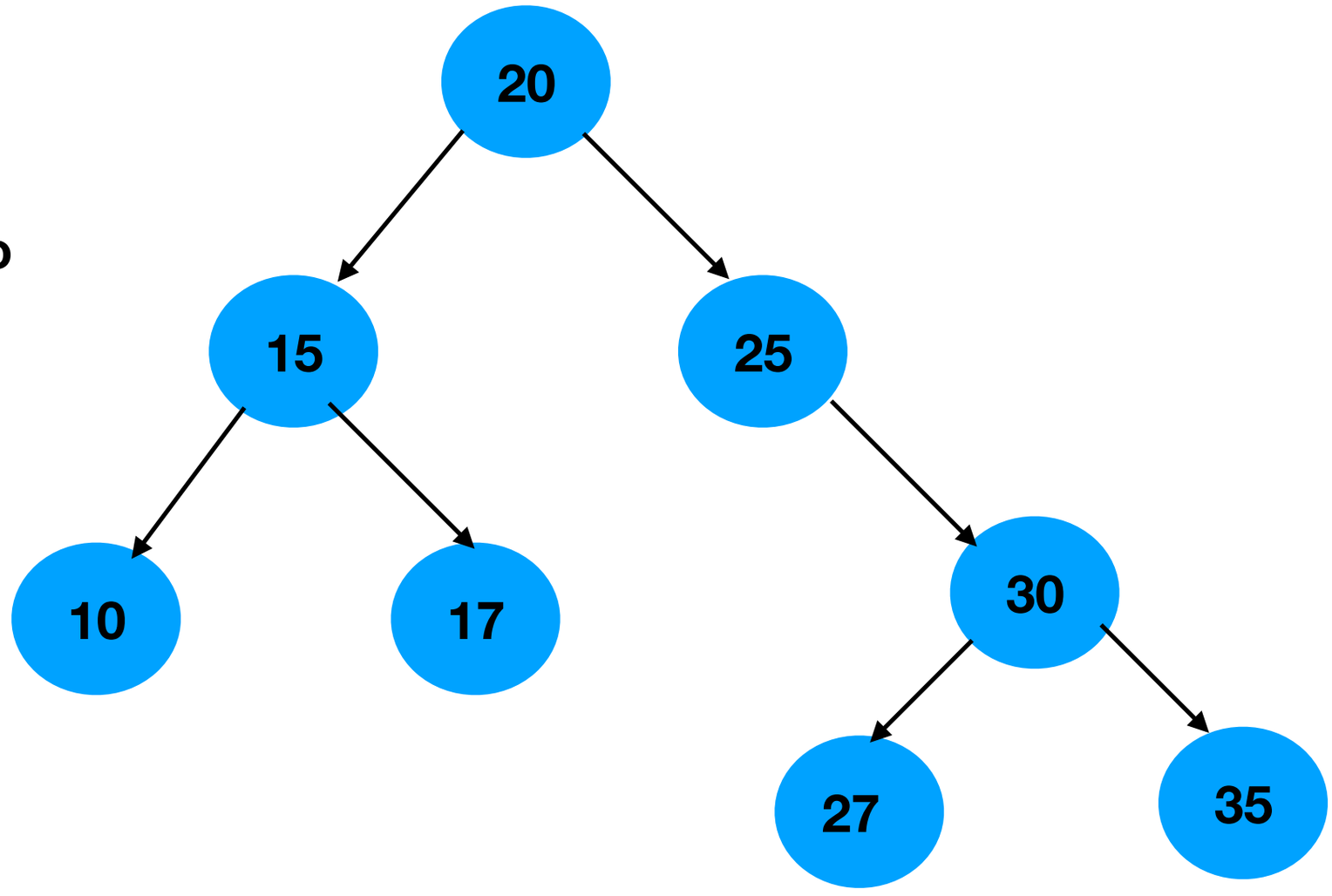
Операция	Время работы в среднем случае	Время работы в худшем случае
Вставка	O(logN)	O(logN)
Поиск	O(logN)	O(logN)
Удаление	O(logN)	O(logN)

Можно ли сделать вставку в дерево за O(1)?

Обходы дерева

Pre-order обход

1. Начинаем с корня
2. Посетить текущий узел
3. Посетить левое поддерево
4. Посетить правое поддерево

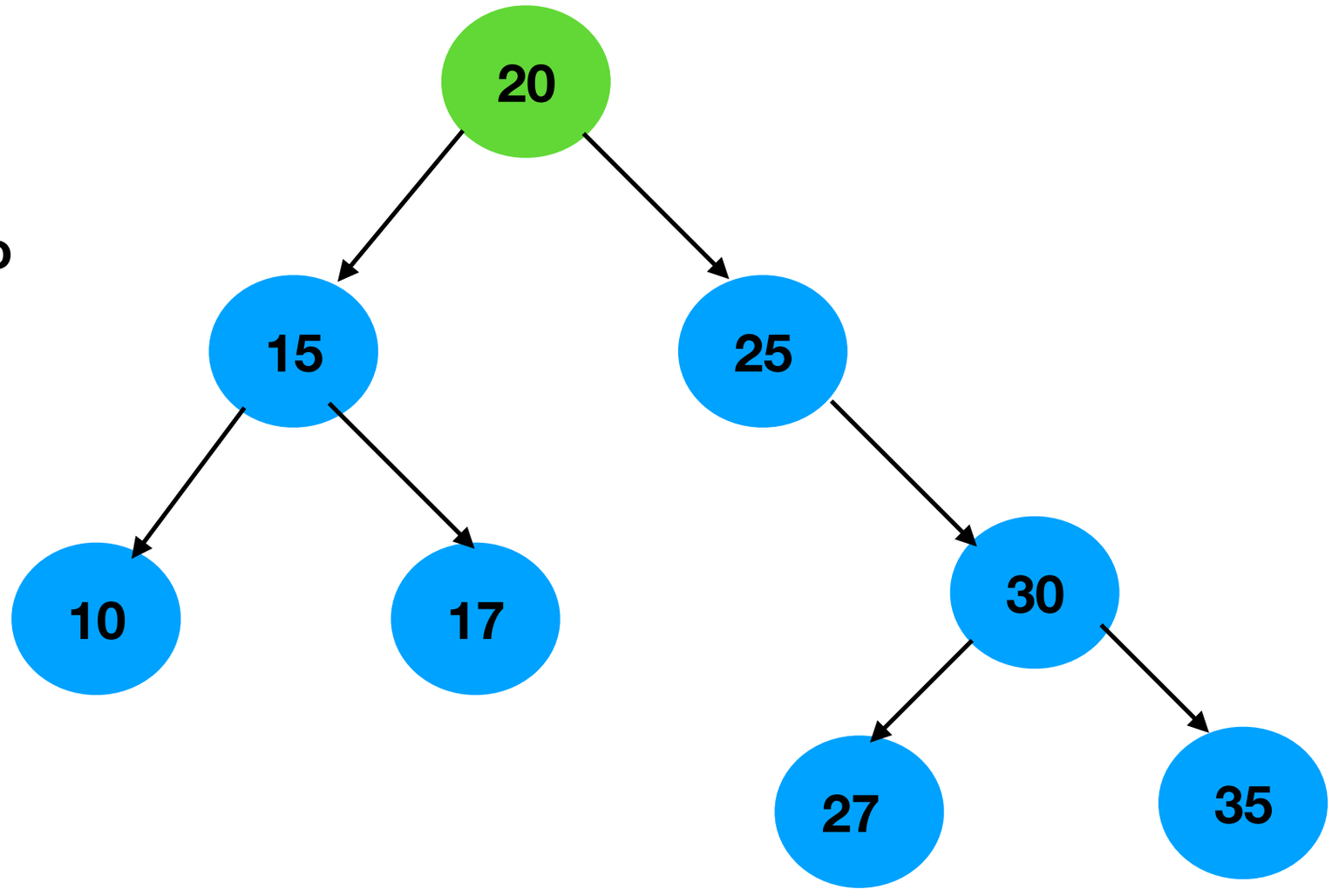


Посетить - применить какую-то функцию к значению в узле. Обычно функцию называют visit

Обходы дерева

Pre-order обход

1. Начинаем с корня
2. Посетить текущий узел
3. Посетить левое поддерево
4. Посетить правое поддерево



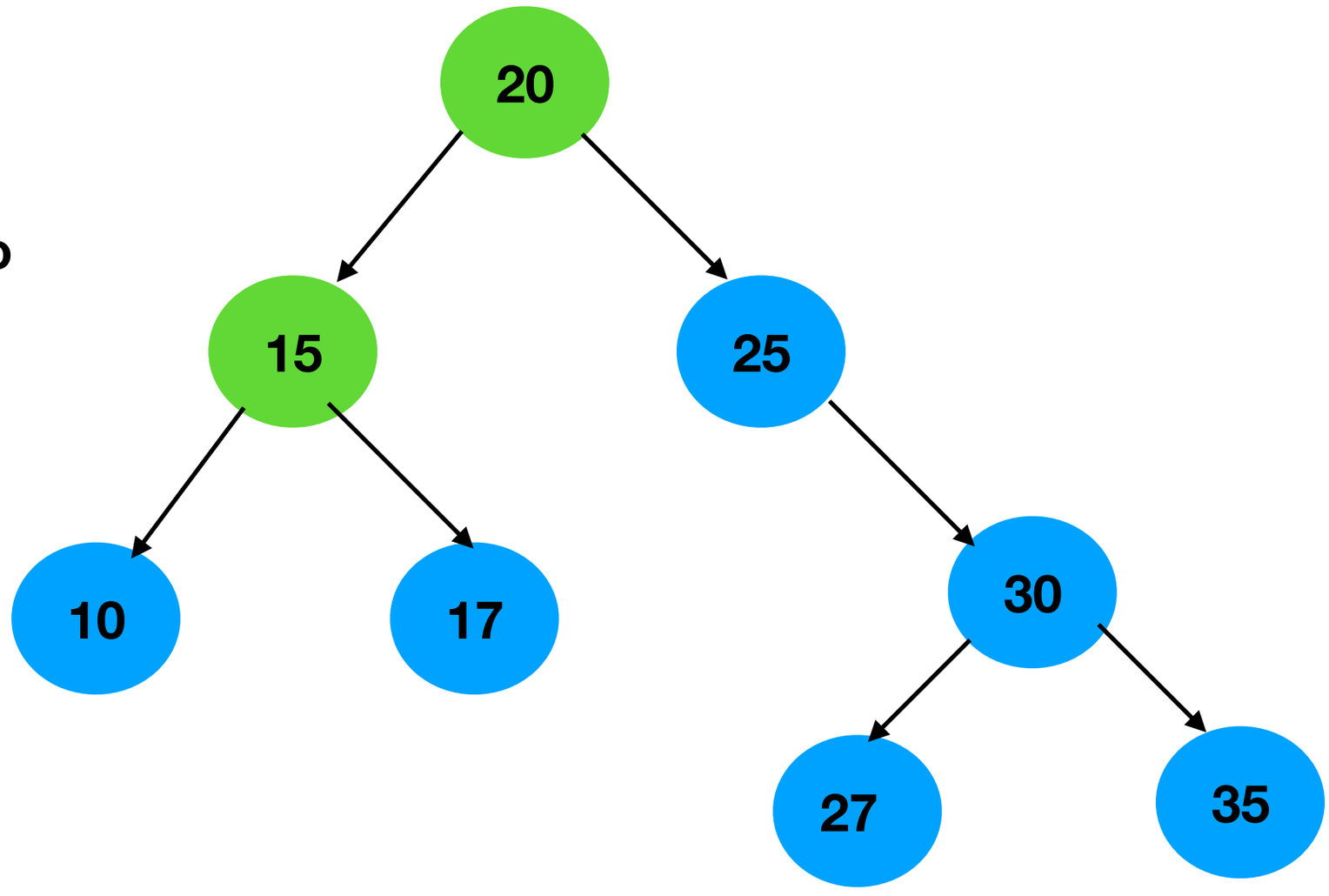
Посетить - применить какую-то функцию к значению в узле. Обычно функцию называют visit

Обходы дерева

Pre-order обход

1. Начинаем с корня
2. Посетить текущий узел
3. Посетить левое поддерево
4. Посетить правое поддерево

Посетить - применить какую-то функцию к значению в узле. Обычно функцию называют visit

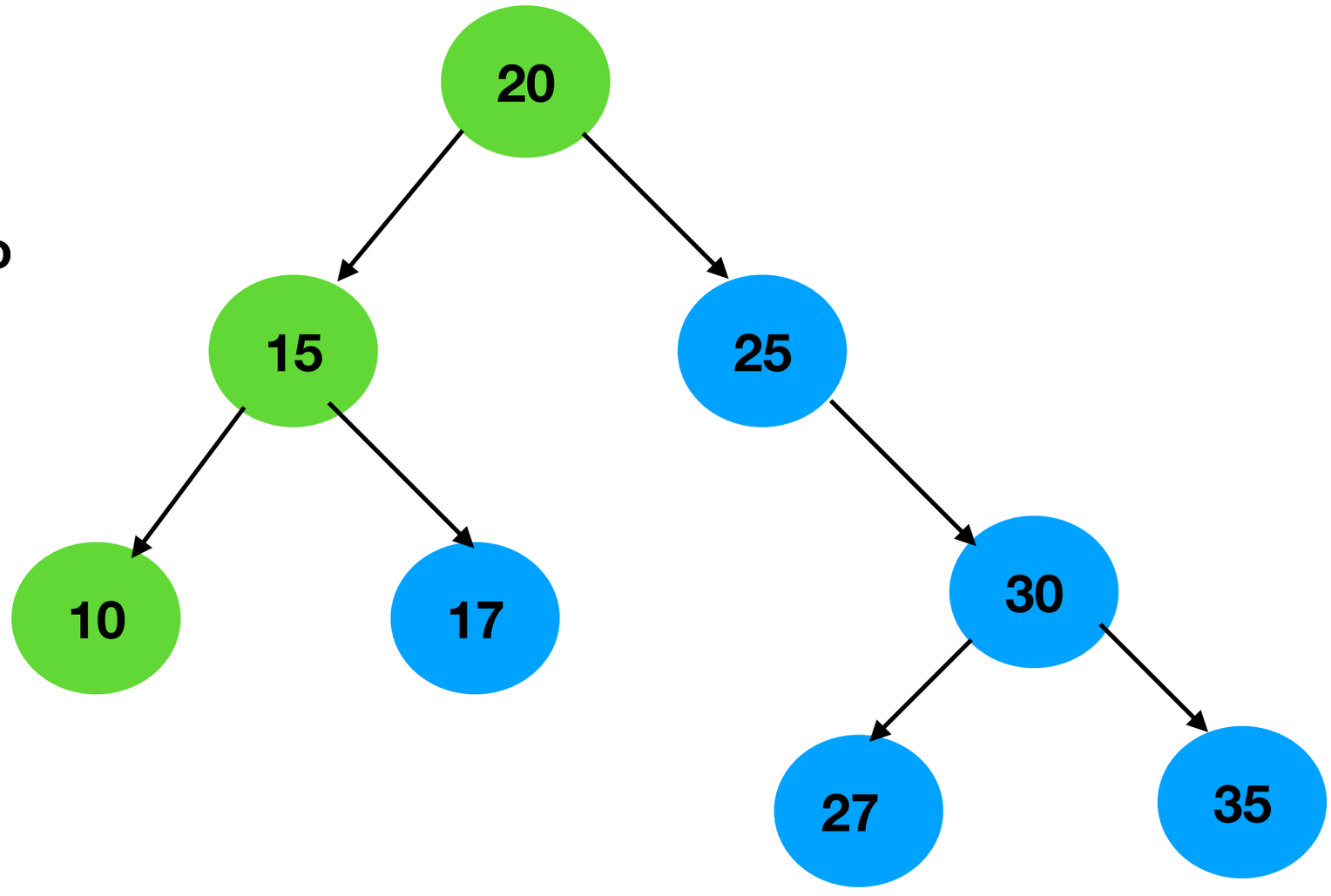


Обходы дерева

Pre-order обход

1. Начинаем с корня
2. Посетить текущий узел
3. Посетить левое поддерево
4. Посетить правое поддерево

Посетить - применить какую-то функцию к значению в узле. Обычно функцию называют visit

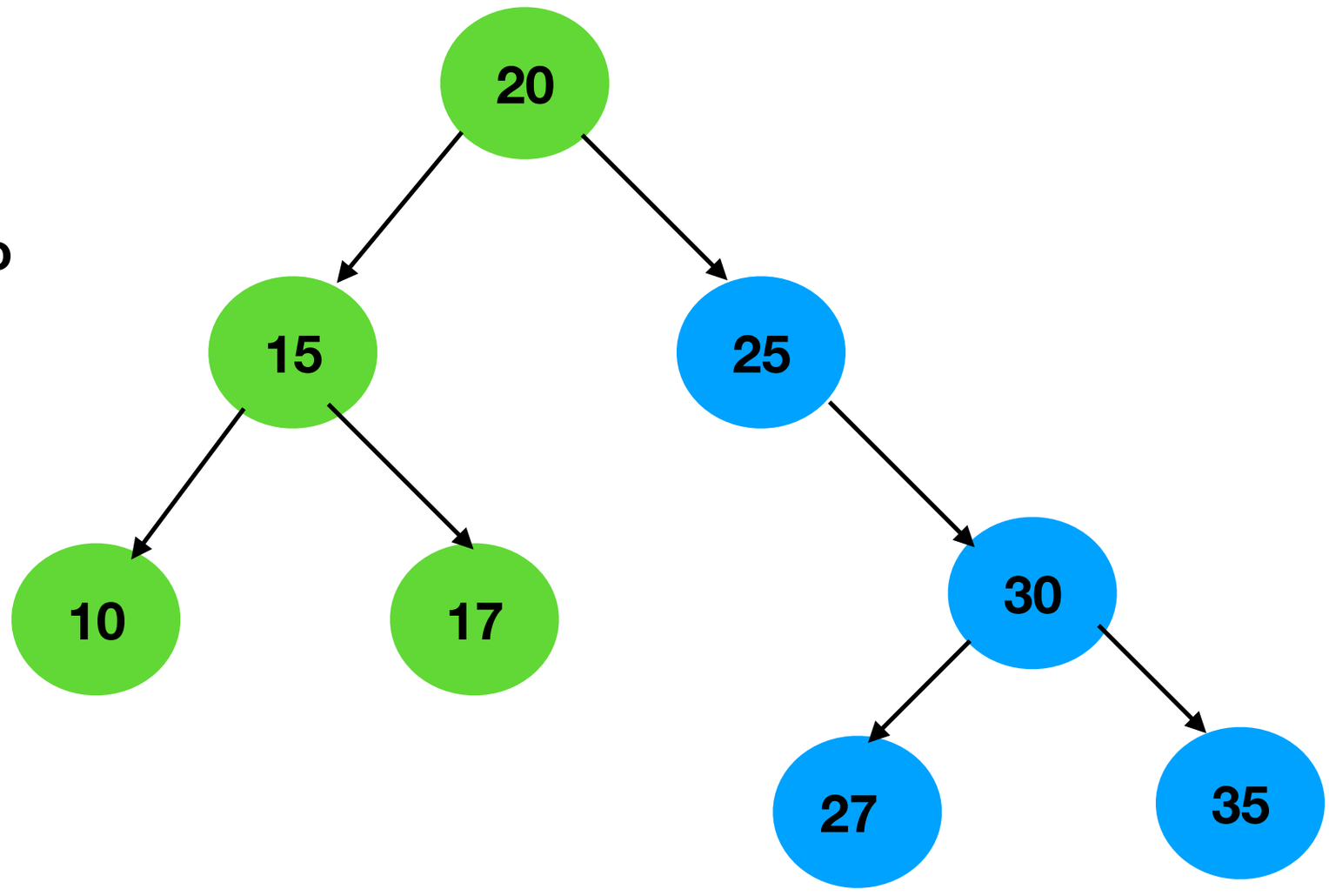


Обходы дерева

Pre-order обход

1. Начинаем с корня
2. Посетить текущий узел
3. Посетить левое поддерево
4. Посетить правое поддерево

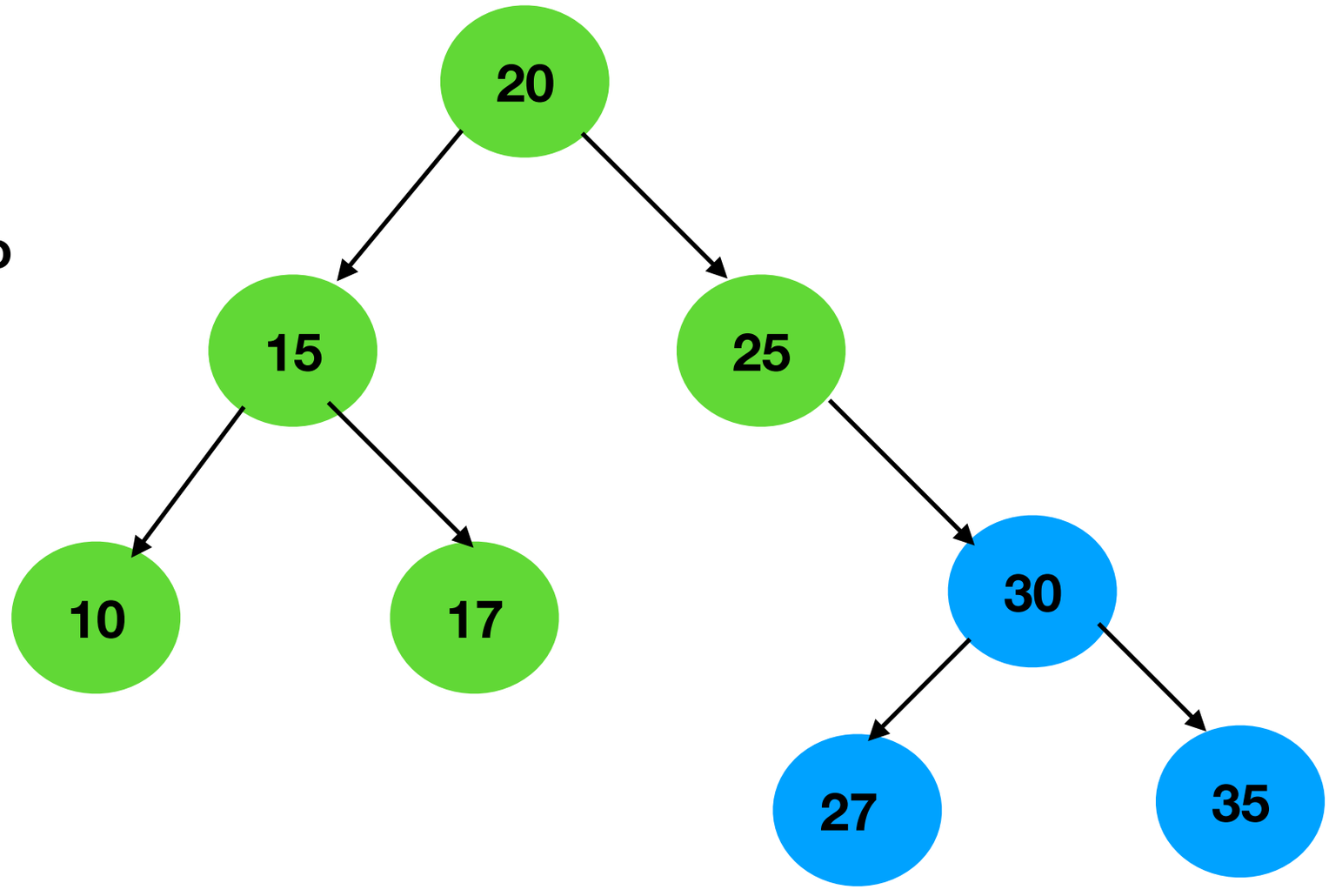
Посетить - применить какую-то функцию к значению в узле. Обычно функцию называют visit



Обходы дерева

Pre-order обход

1. Начинаем с корня
2. Посетить текущий узел
3. Посетить левое поддерево
4. Посетить правое поддерево

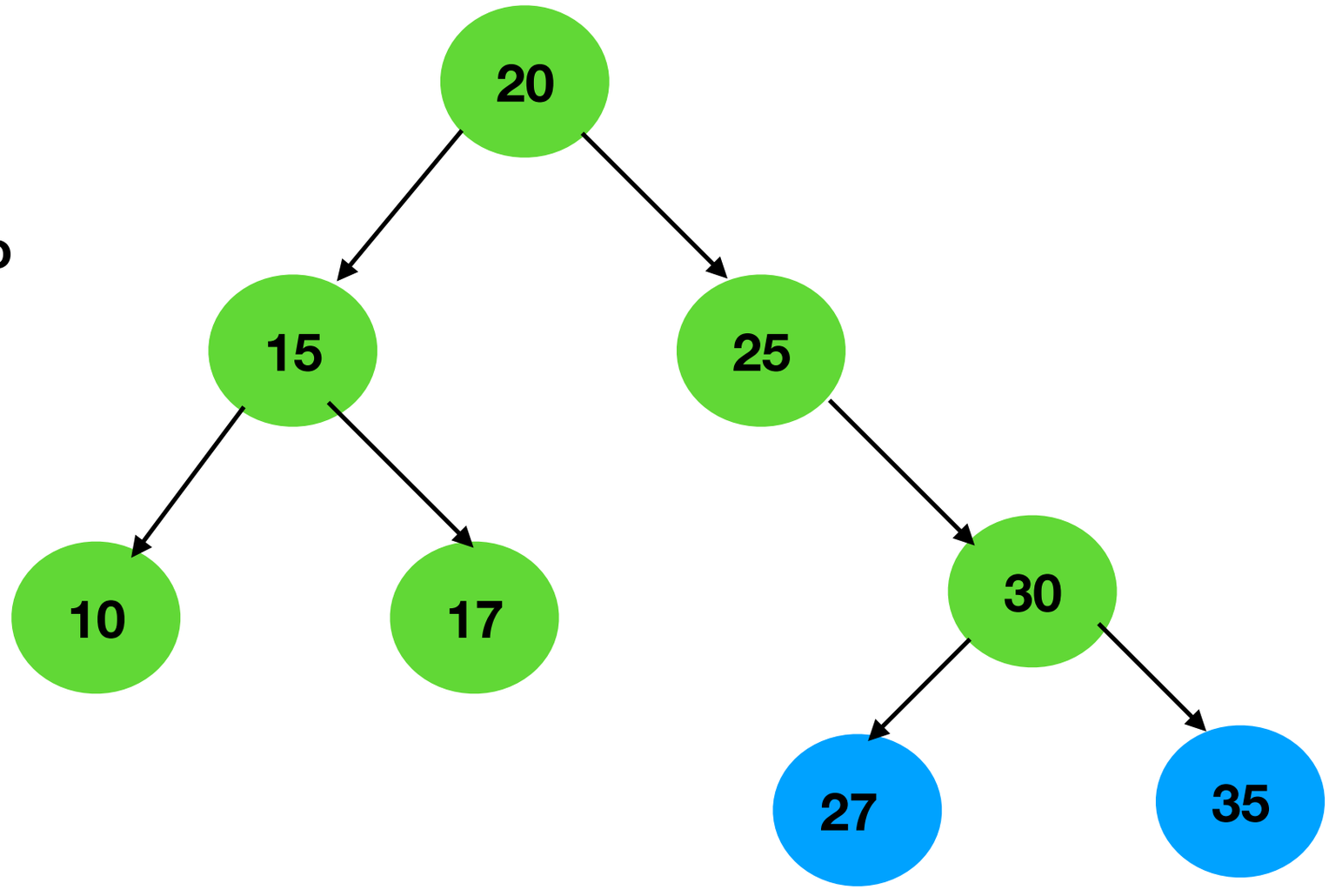


Посетить - применить какую-то функцию к значению в узле. Обычно функцию называют visit

Обходы дерева

Pre-order обход

1. Начинаем с корня
2. Посетить текущий узел
3. Посетить левое поддерево
4. Посетить правое поддерево



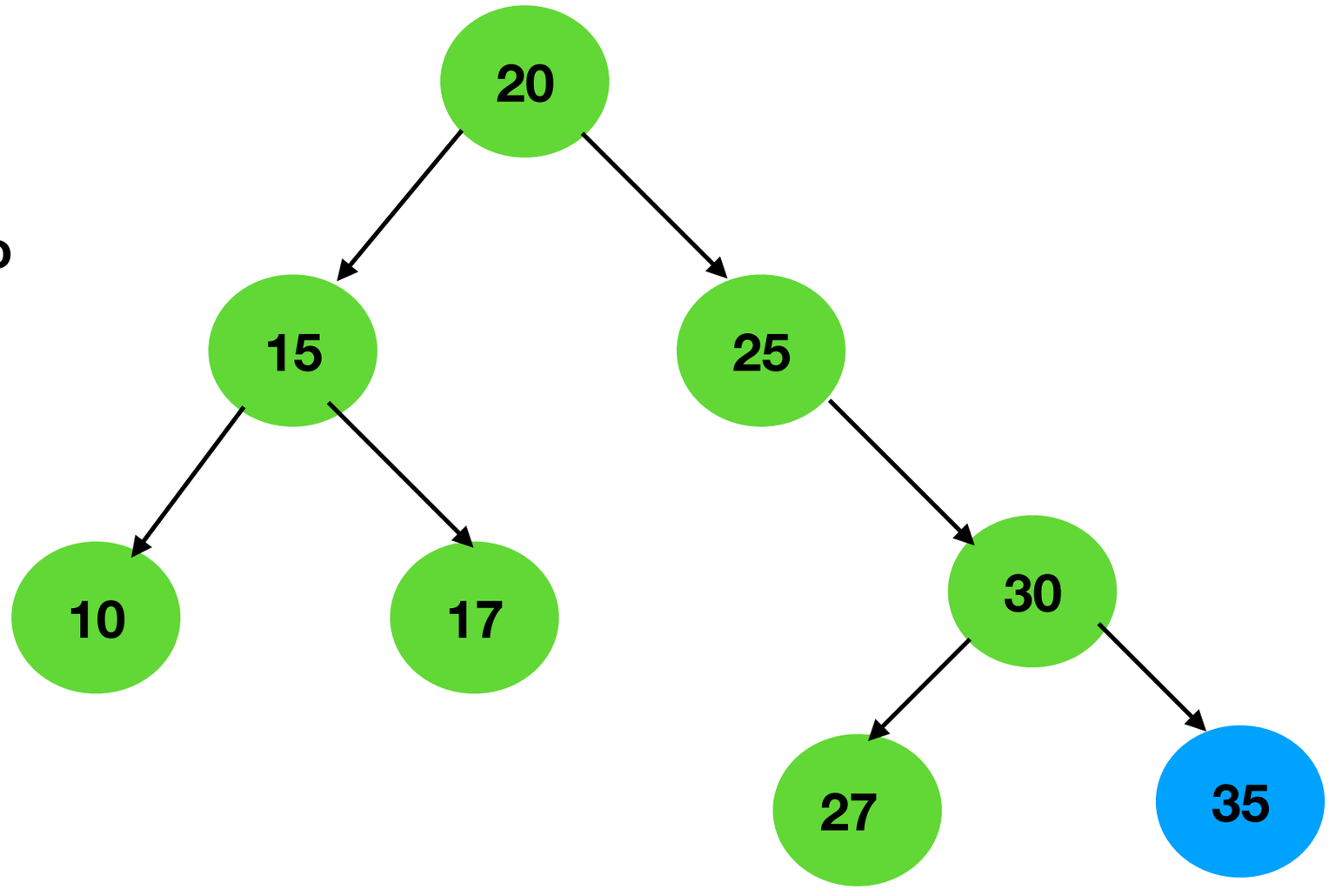
Посетить - применить какую-то функцию к значению в узле. Обычно функцию называют visit

Обходы дерева

Pre-order обход

1. Начинаем с корня
2. Посетить текущий узел
3. Посетить левое поддерево
4. Посетить правое поддерево

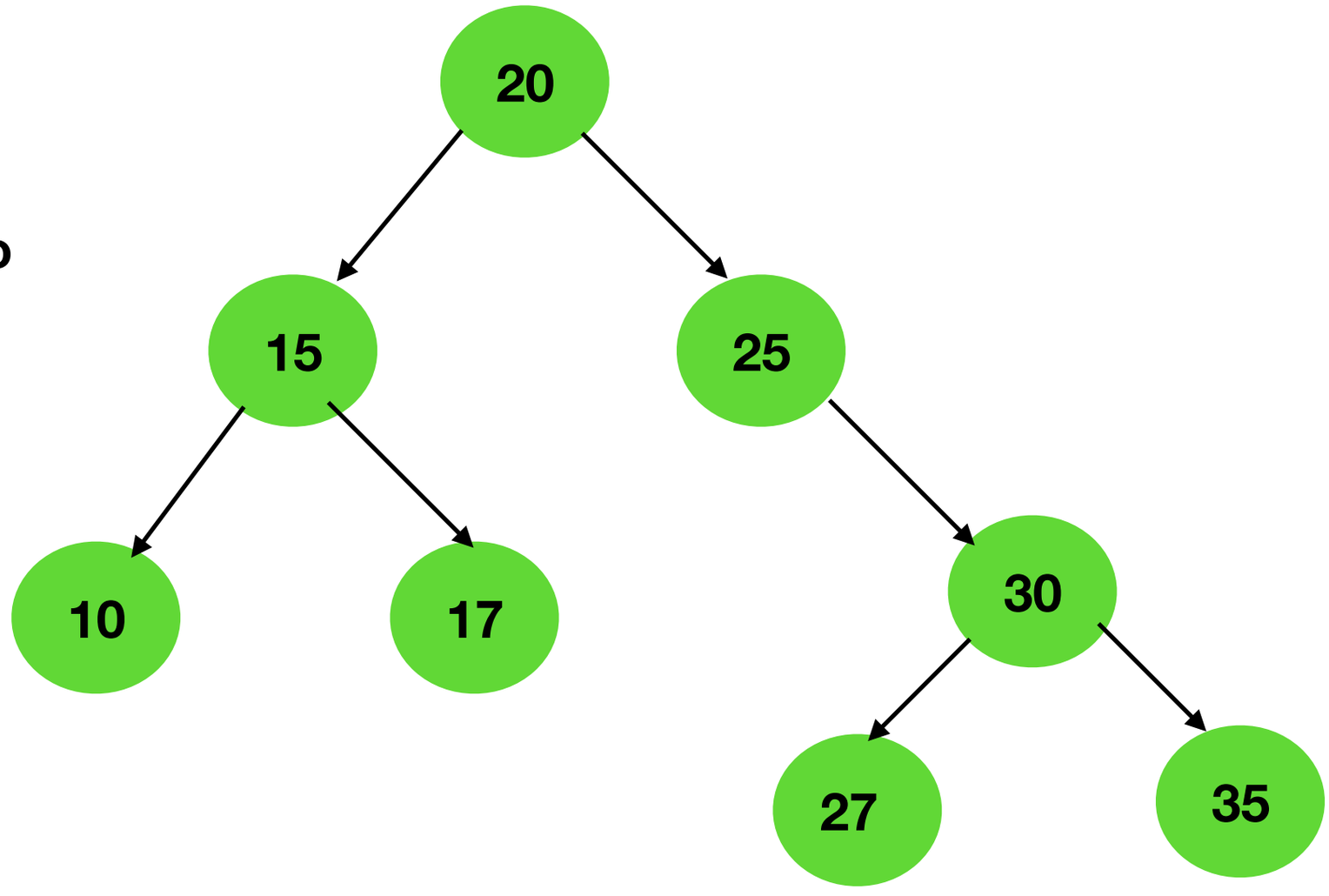
Посетить - применить какую-то функцию к значению в узле. Обычно функцию называют visit



Обходы дерева

Pre-order обход

1. Начинаем с корня
2. Посетить текущий узел
3. Посетить левое поддерево
4. Посетить правое поддерево



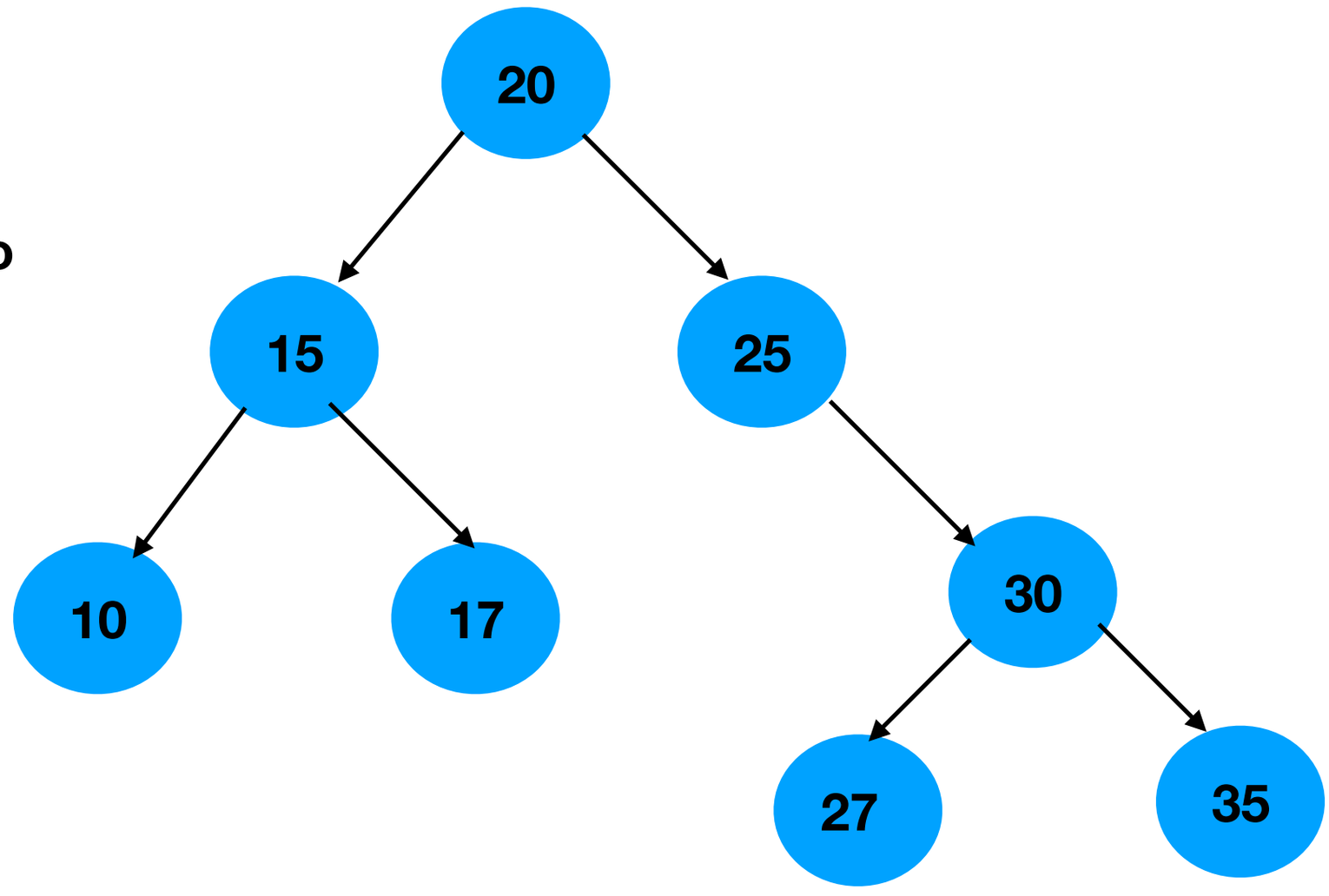
Посетить - применить какую-то функцию к значению в узле. Обычно функцию называют visit

20, 15, 10, 17,25, 30, 27, 35

Обходы дерева

In-order обход

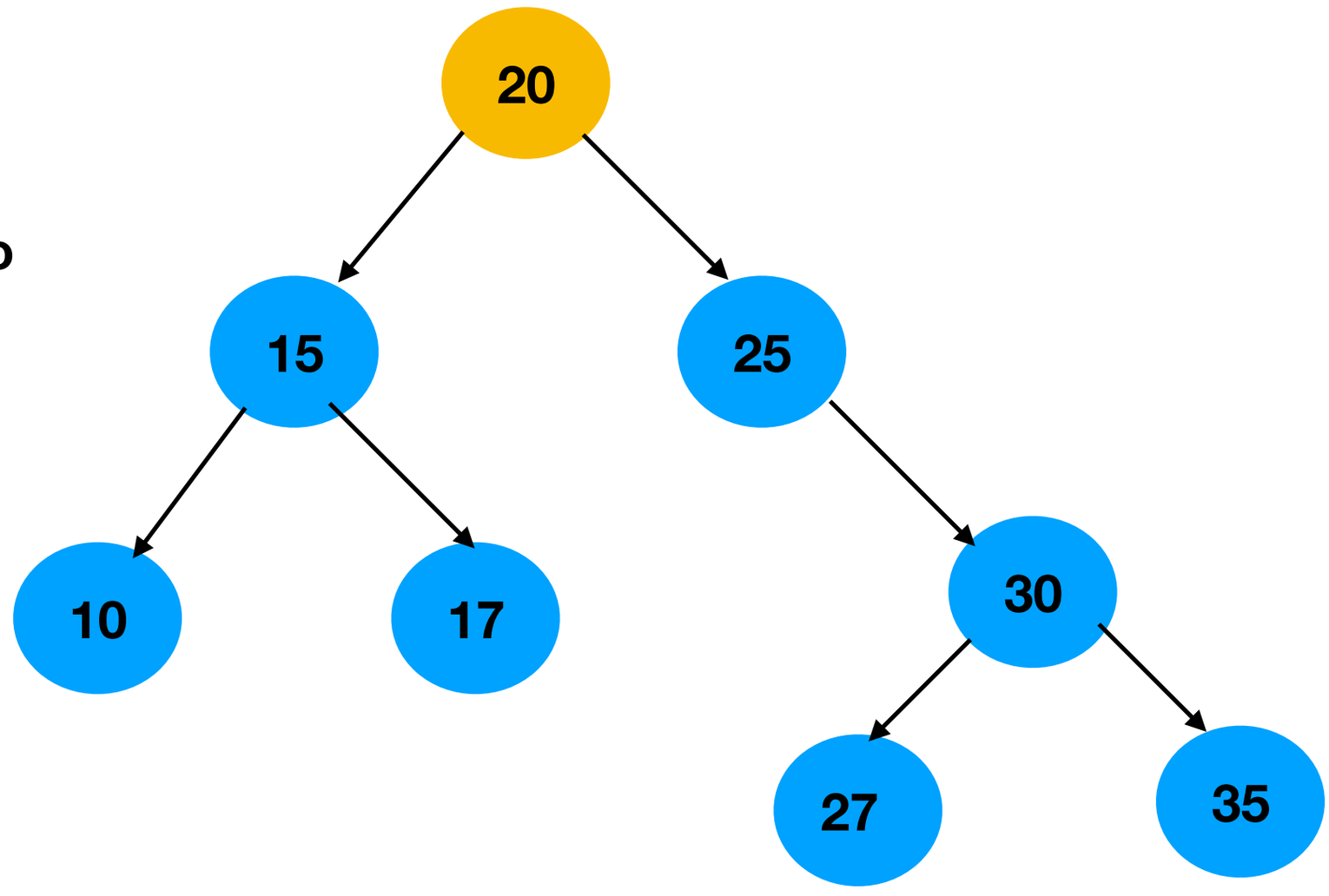
1. Начинаем с корня
2. Посетить левое поддерево
3. Посетить текущий узел
4. Посетить правое поддерево



Обходы дерева

In-order обход

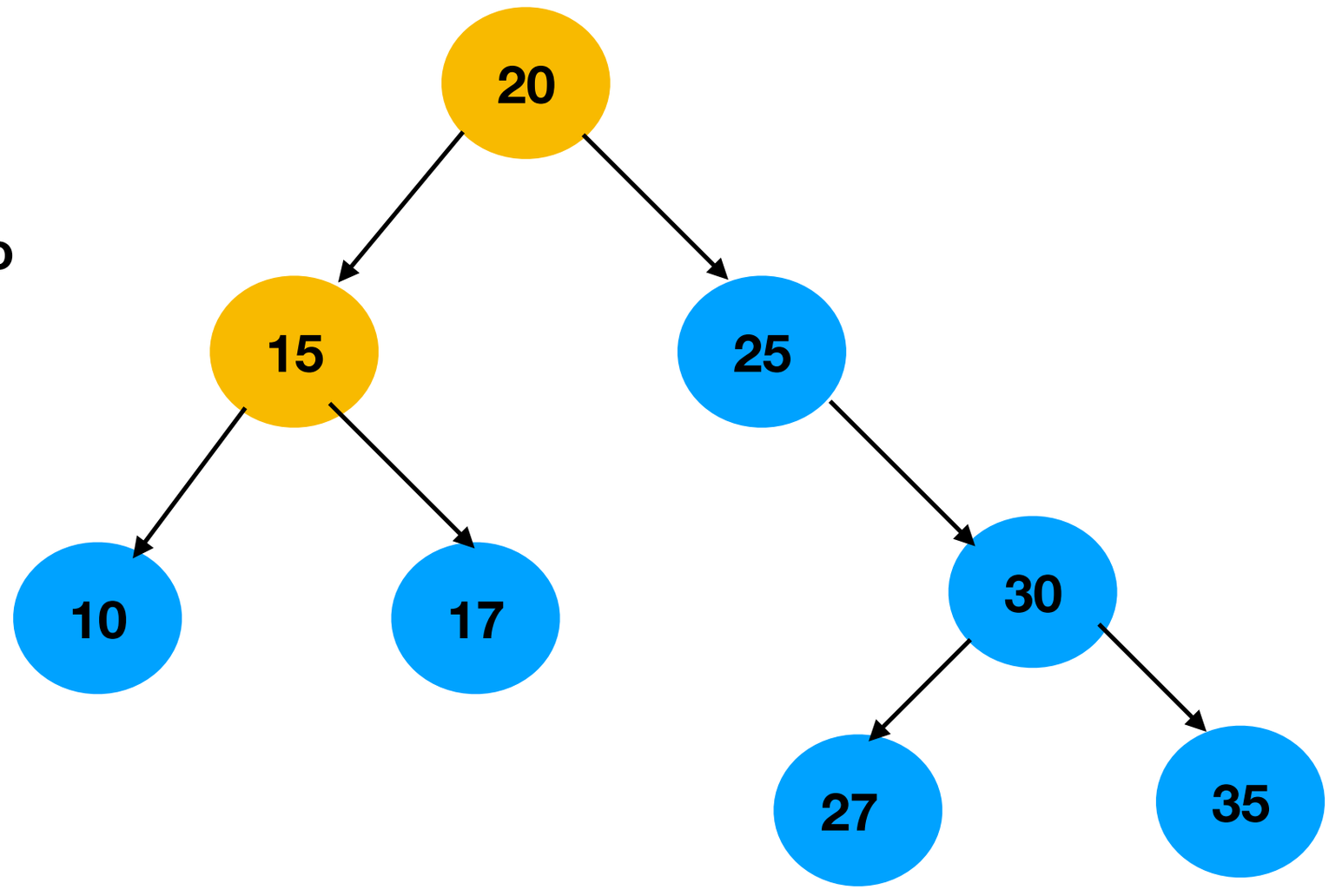
1. Начинаем с корня
2. Посетить левое поддерево
3. Посетить текущий узел
4. Посетить правое поддерево



Обходы дерева

In-order обход

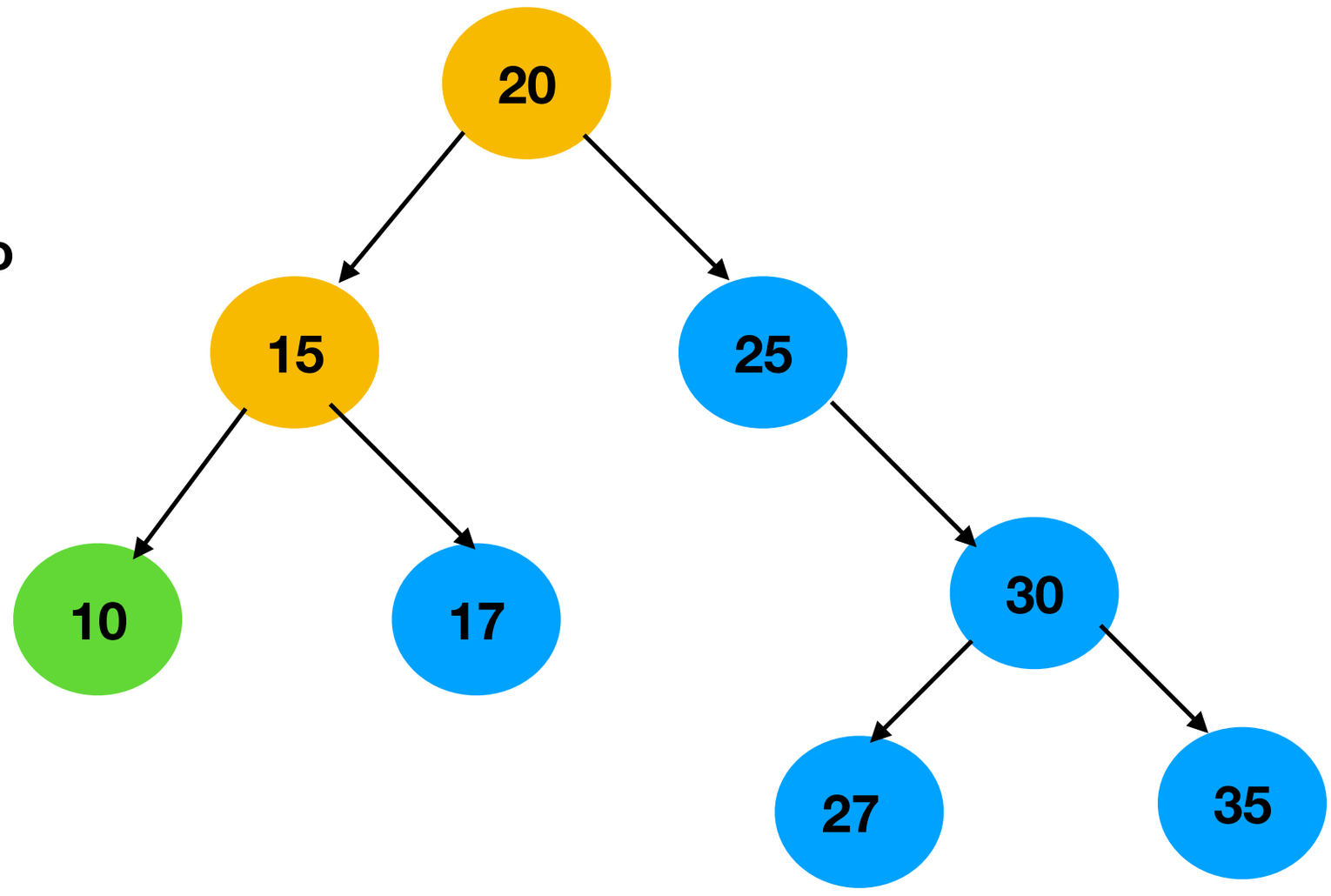
1. Начинаем с корня
2. Посетить левое поддерево
3. Посетить текущий узел
4. Посетить правое поддерево



Обходы дерева

In-order обход

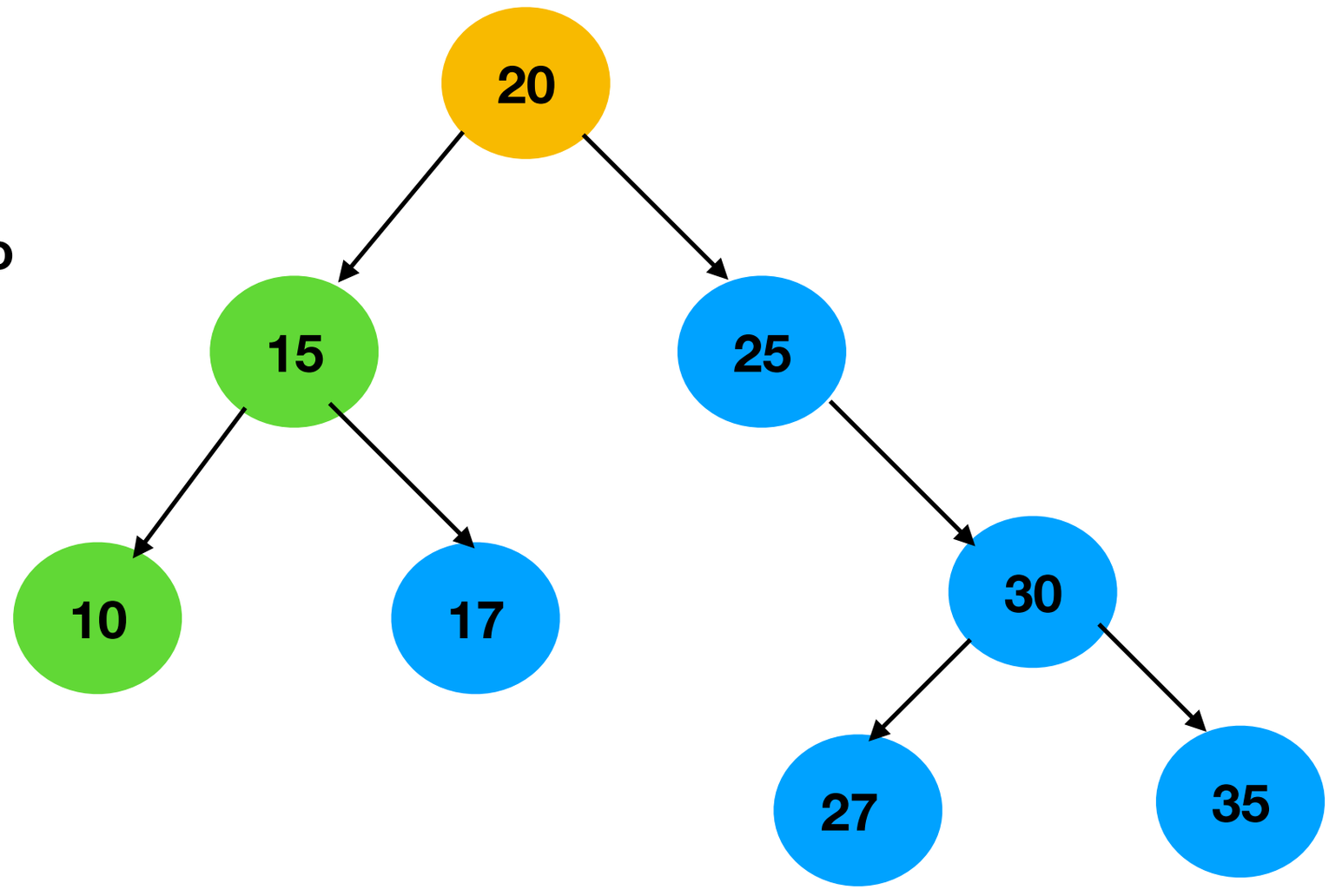
1. Начинаем с корня
2. Посетить левое поддерево
3. Посетить текущий узел
4. Посетить правое поддерево



Обходы дерева

In-order обход

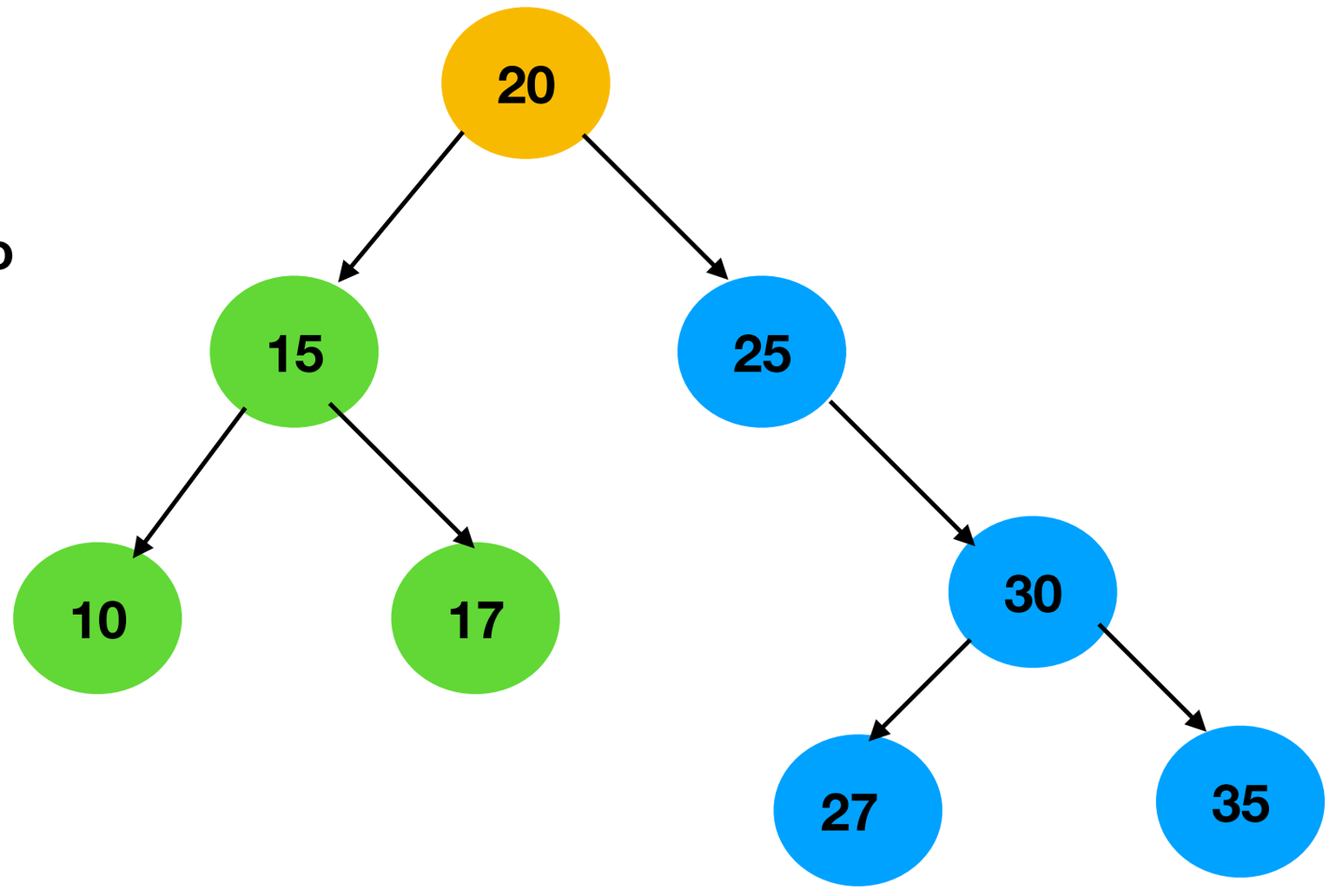
1. Начинаем с корня
2. Посетить левое поддерево
3. Посетить текущий узел
4. Посетить правое поддерево



Обходы дерева

In-order обход

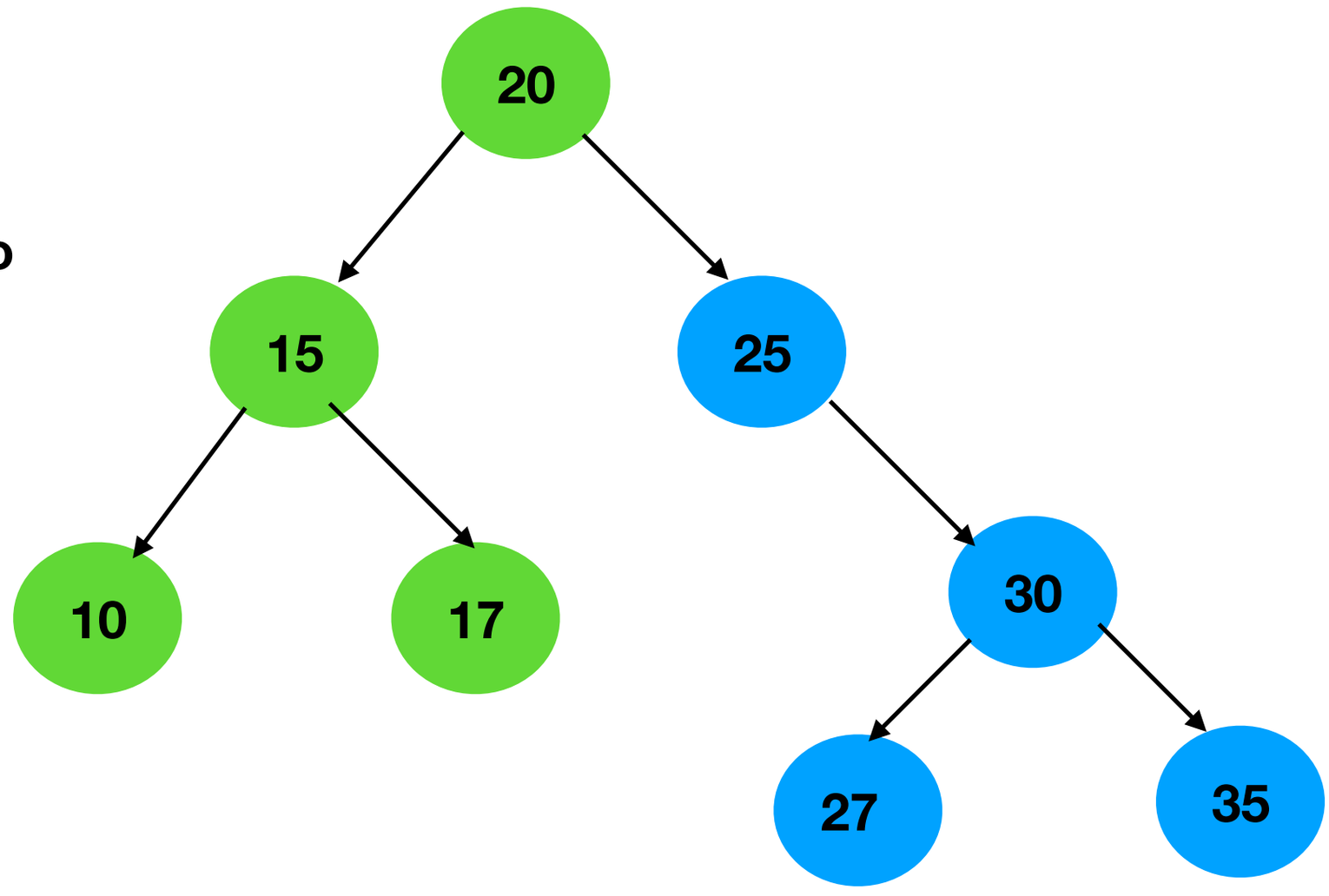
1. Начинаем с корня
2. Посетить левое поддерево
3. Посетить текущий узел
4. Посетить правое поддерево



Обходы дерева

In-order обход

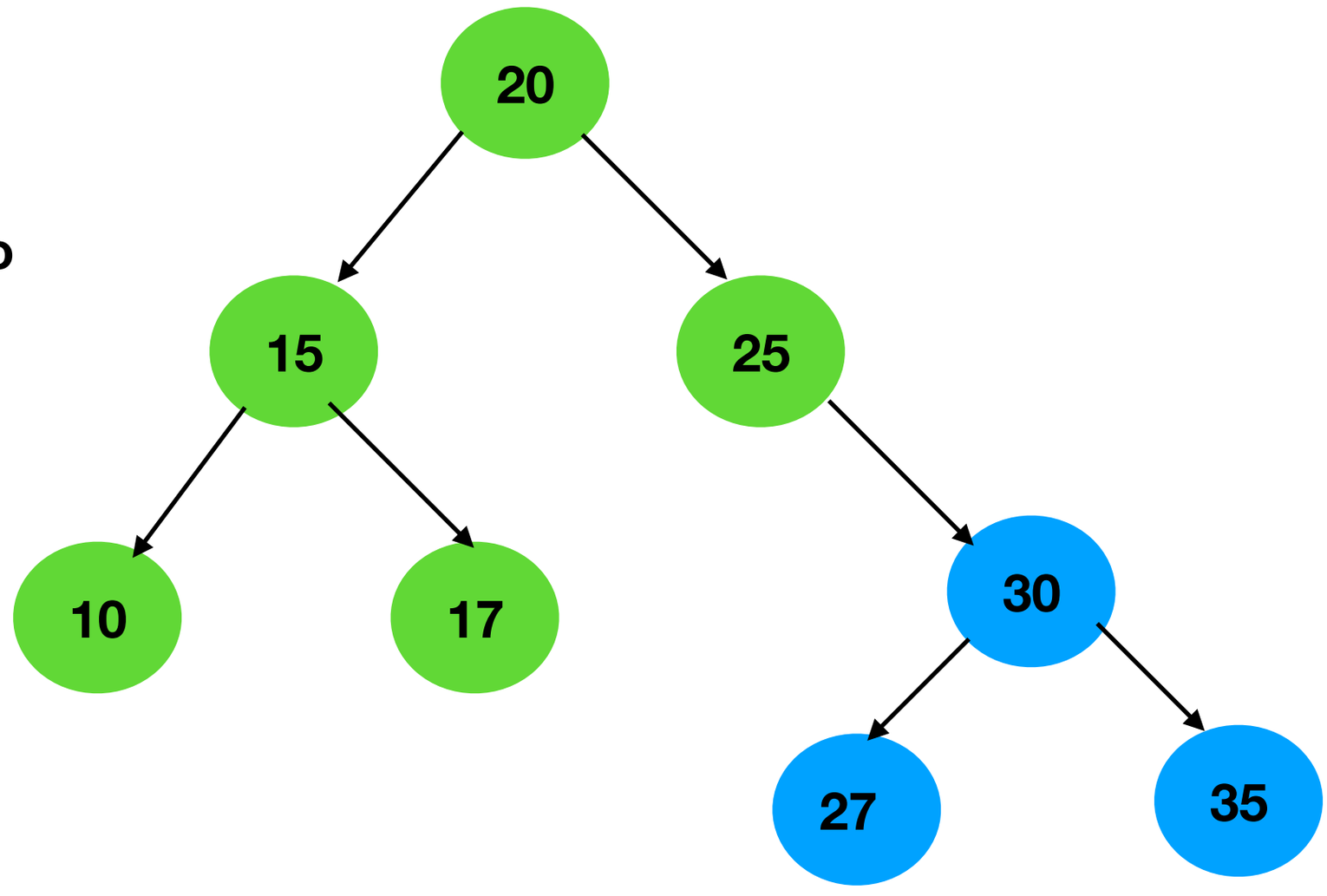
1. Начинаем с корня
2. Посетить левое поддерево
3. Посетить текущий узел
4. Посетить правое поддерево



Обходы дерева

In-order обход

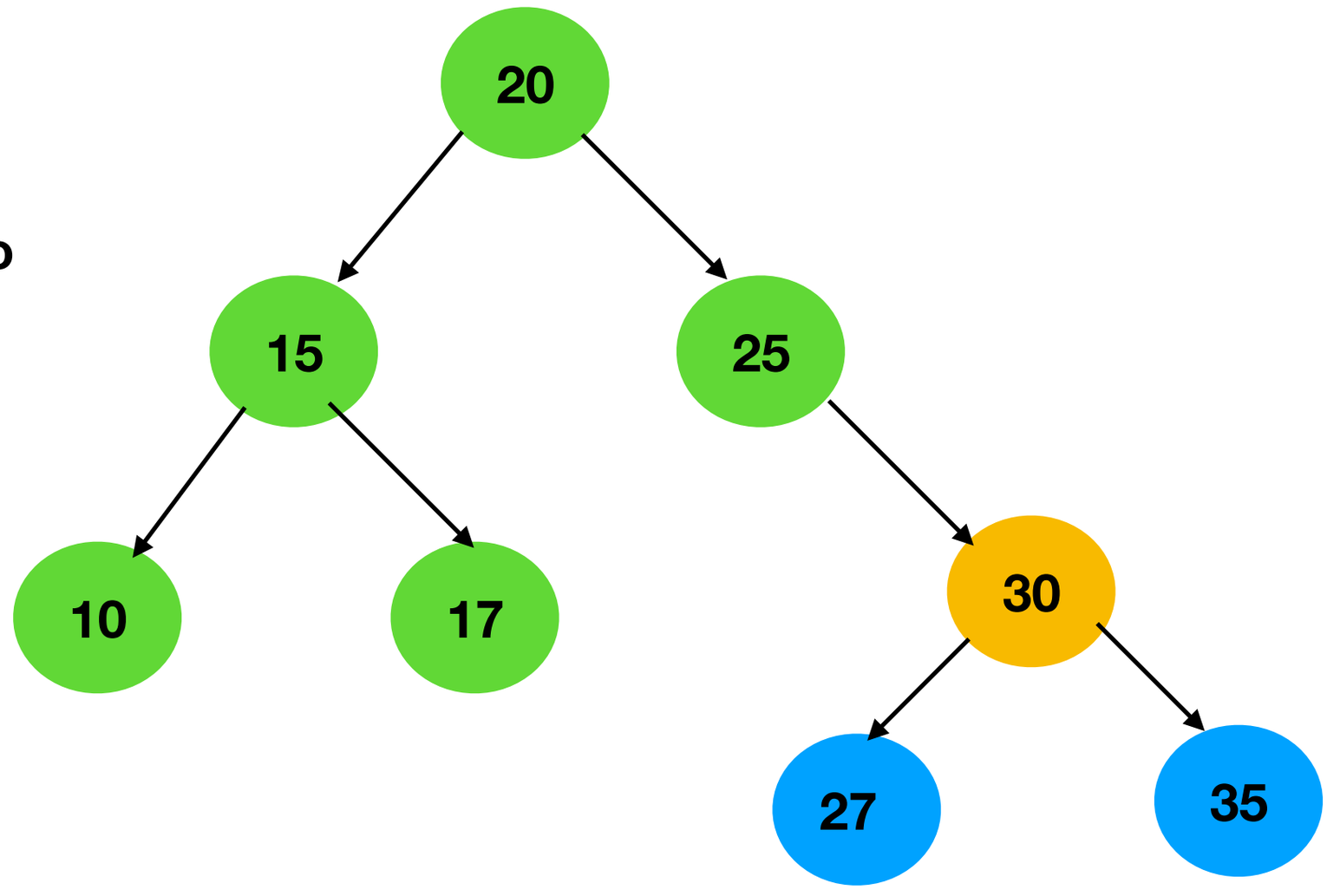
1. Начинаем с корня
2. Посетить левое поддерево
3. Посетить текущий узел
4. Посетить правое поддерево



Обходы дерева

In-order обход

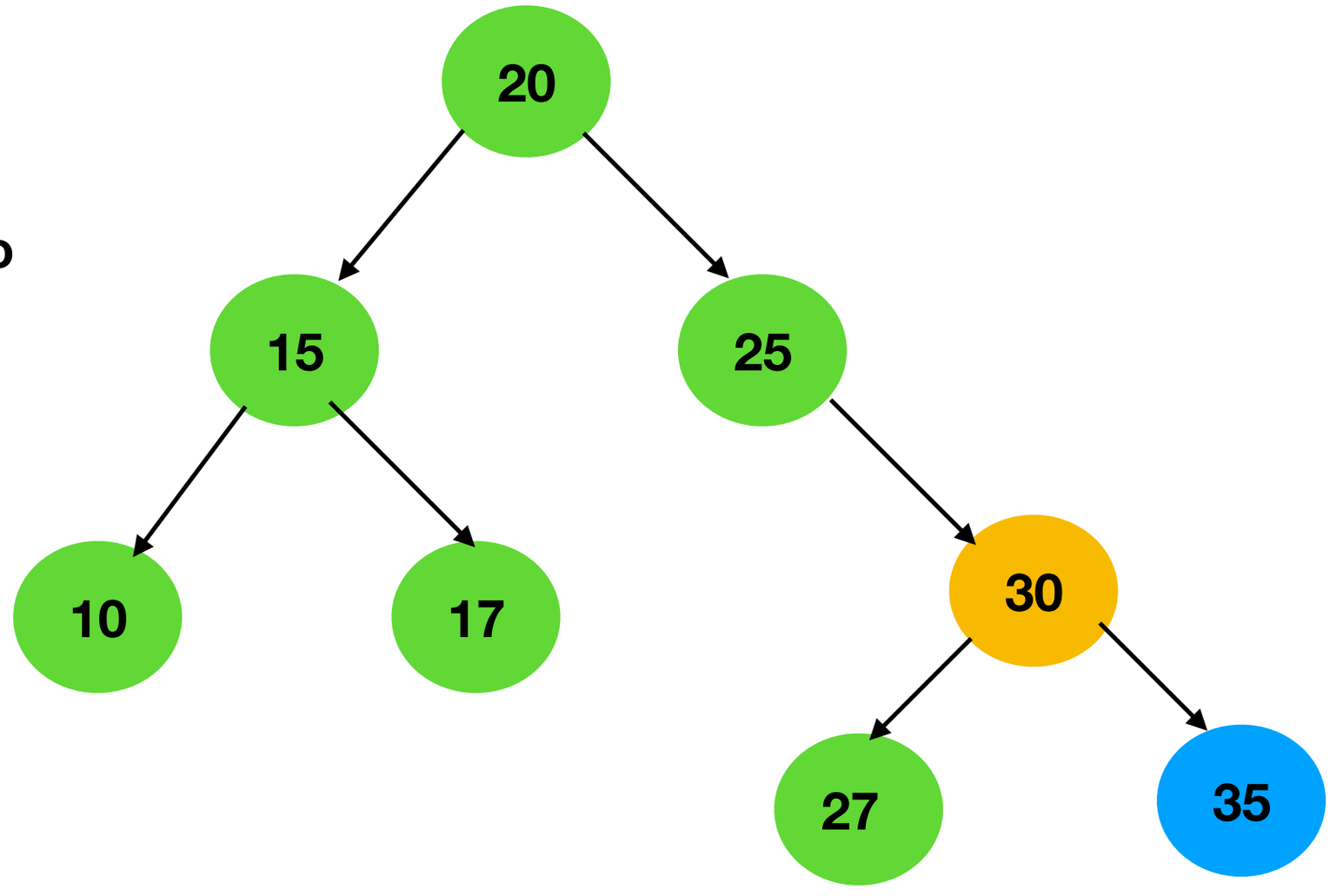
1. Начинаем с корня
2. Посетить левое поддерево
3. Посетить текущий узел
4. Посетить правое поддерево



Обходы дерева

In-order обход

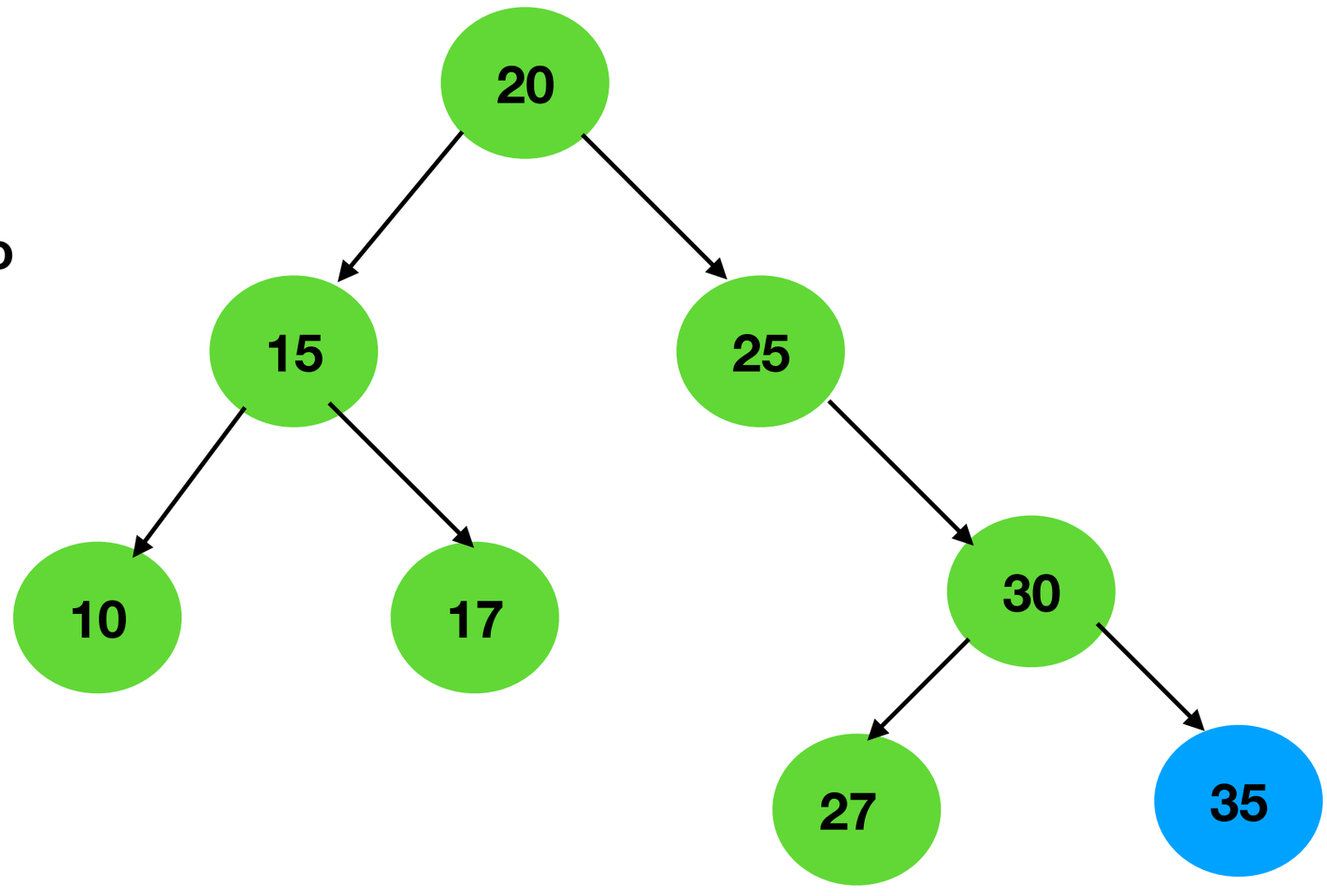
1. Начинаем с корня
2. Посетить левое поддерево
3. Посетить текущий узел
4. Посетить правое поддерево



Обходы дерева

In-order обход

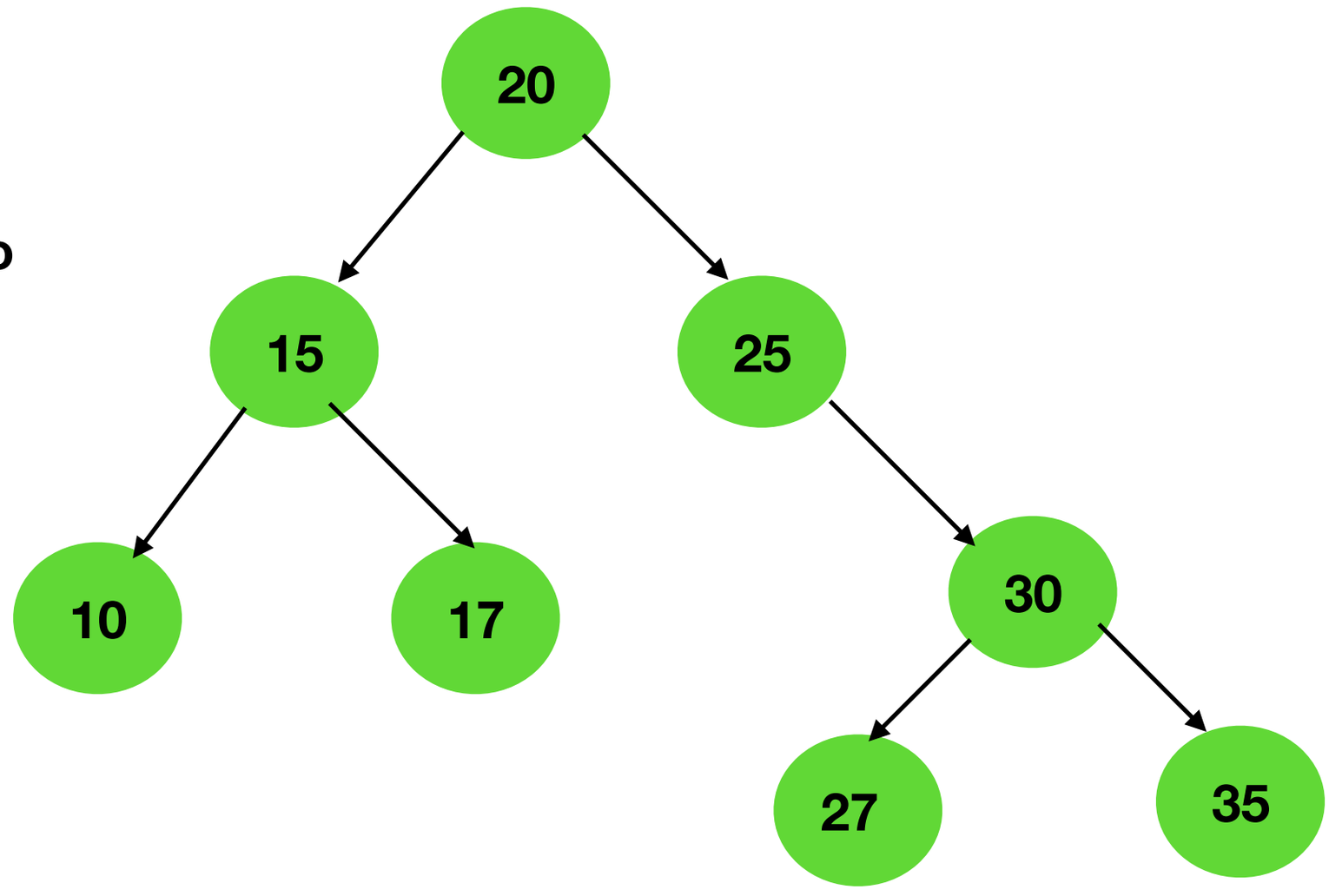
1. Начинаем с корня
2. Посетить левое поддерево
3. Посетить текущий узел
4. Посетить правое поддерево



Обходы дерева

In-order обход

1. Начинаем с корня
2. Посетить левое поддерево
3. Посетить текущий узел
4. Посетить правое поддерево

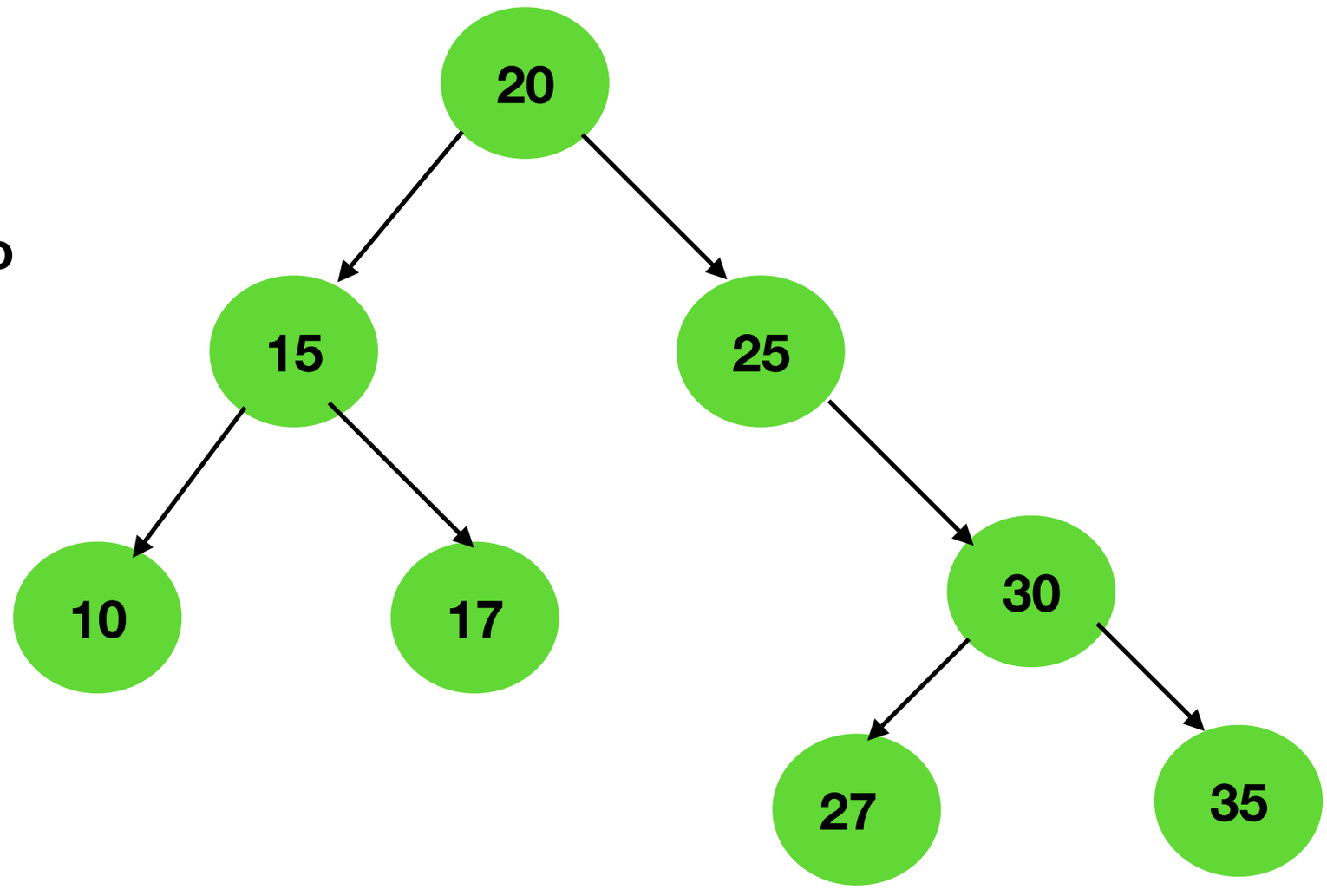


10, 15, 17, 20, 25, 27, 30, 35

Обходы дерева

In-order обход

1. Начинаем с корня
2. Посетить левое поддерево
3. Посетить текущий узел
4. Посетить правое поддерево



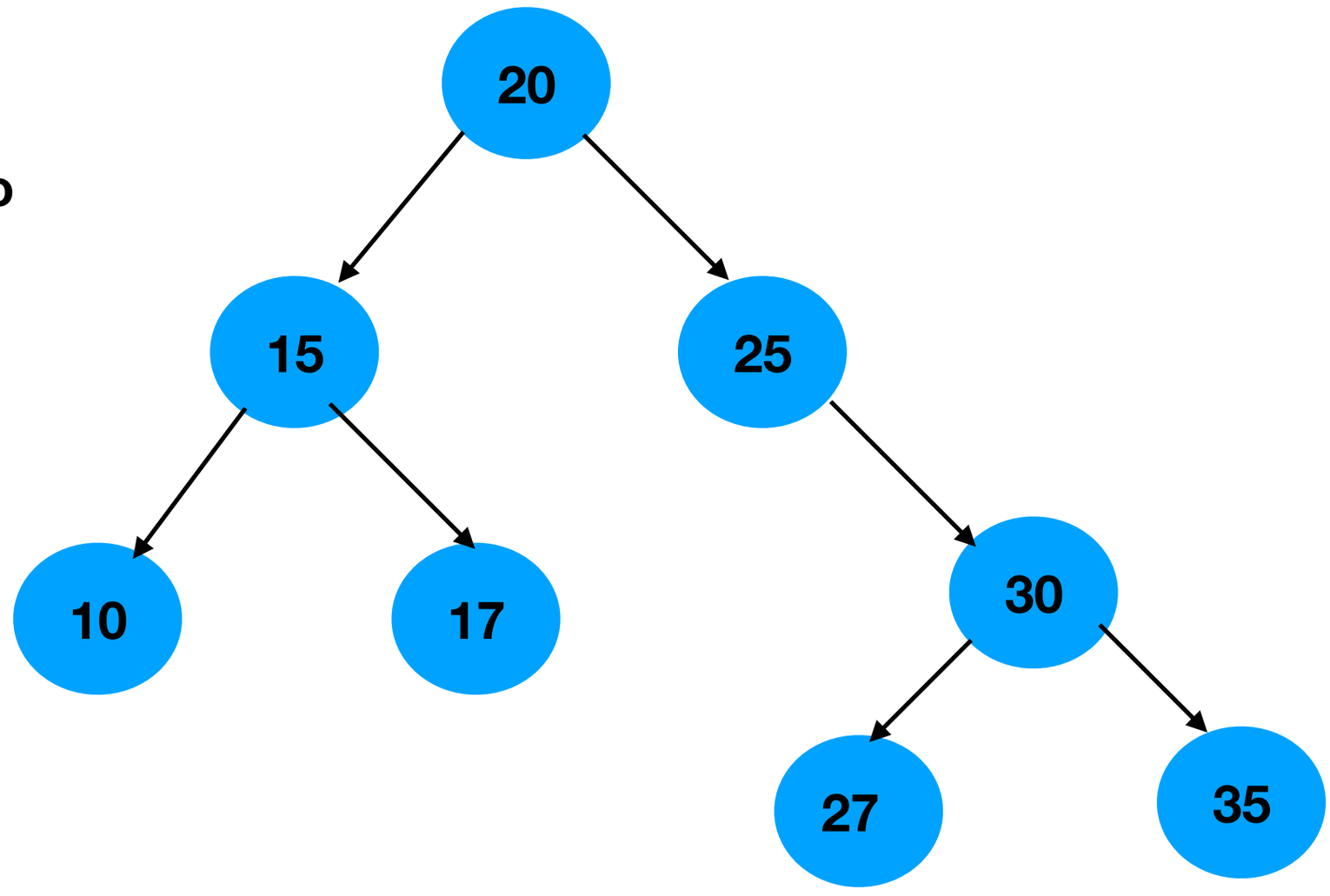
Такой обход выведет обойдет значения в дереве в порядке возрастания

10, 15, 17, 20, 25, 27, 30, 35

Обходы дерева

Post-order обход

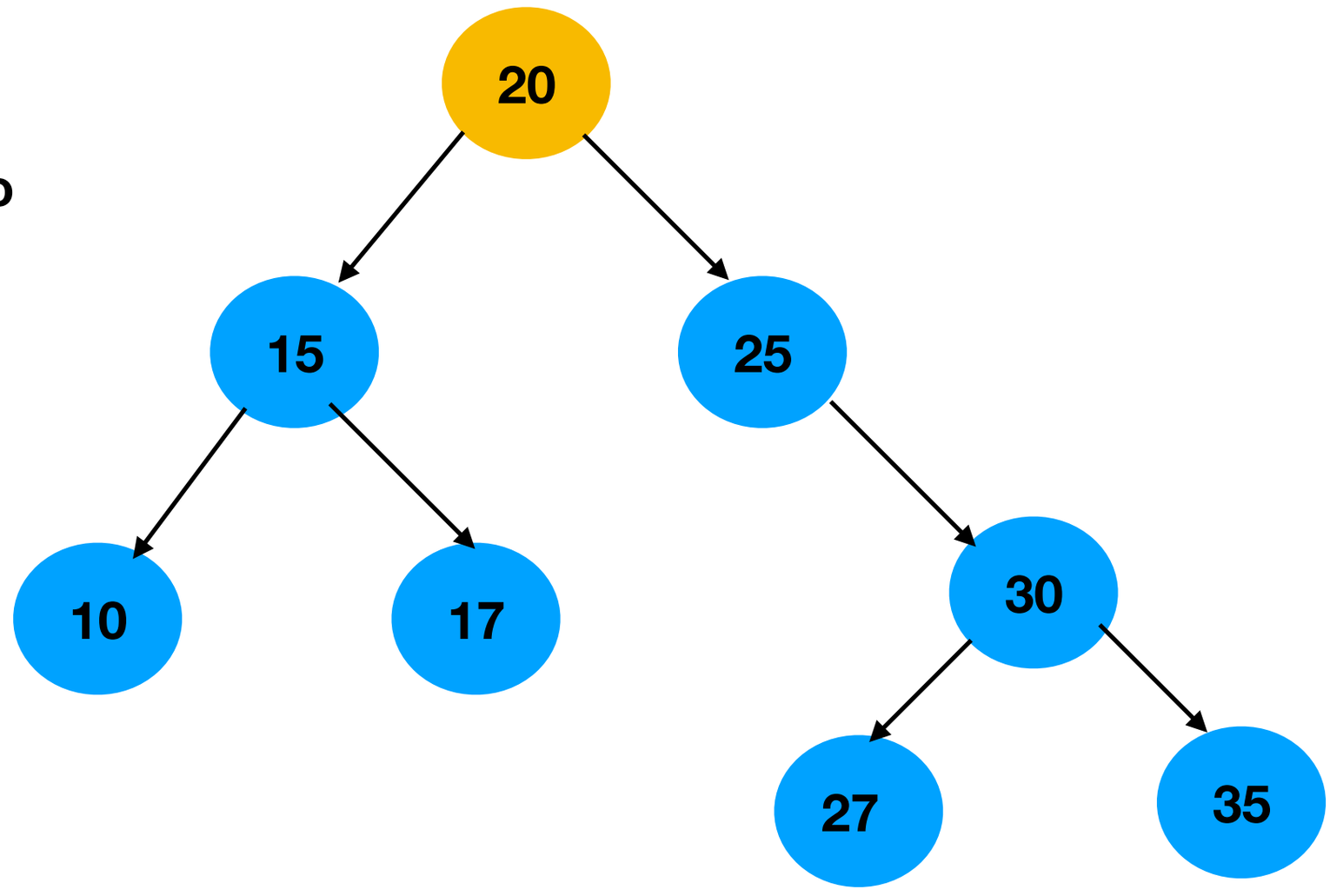
1. Начинаем с корня
2. Посетить левое поддерево
3. Посетить правое поддерево
4. Посетить текущий узел



Обходы дерева

Post-order обход

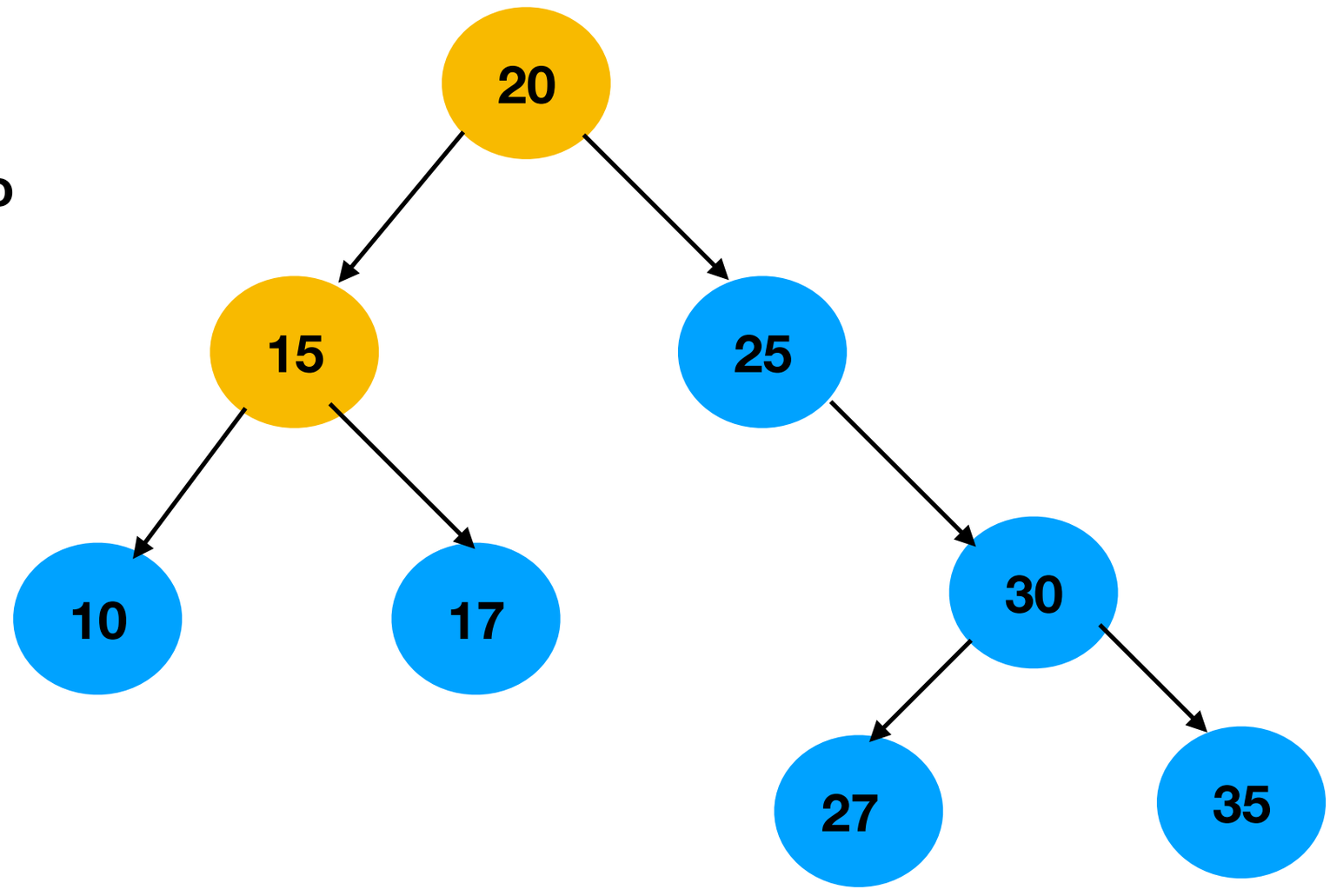
1. Начинаем с корня
2. Посетить левое поддерево
3. Посетить правое поддерево
4. Посетить текущий узел



Обходы дерева

Post-order обход

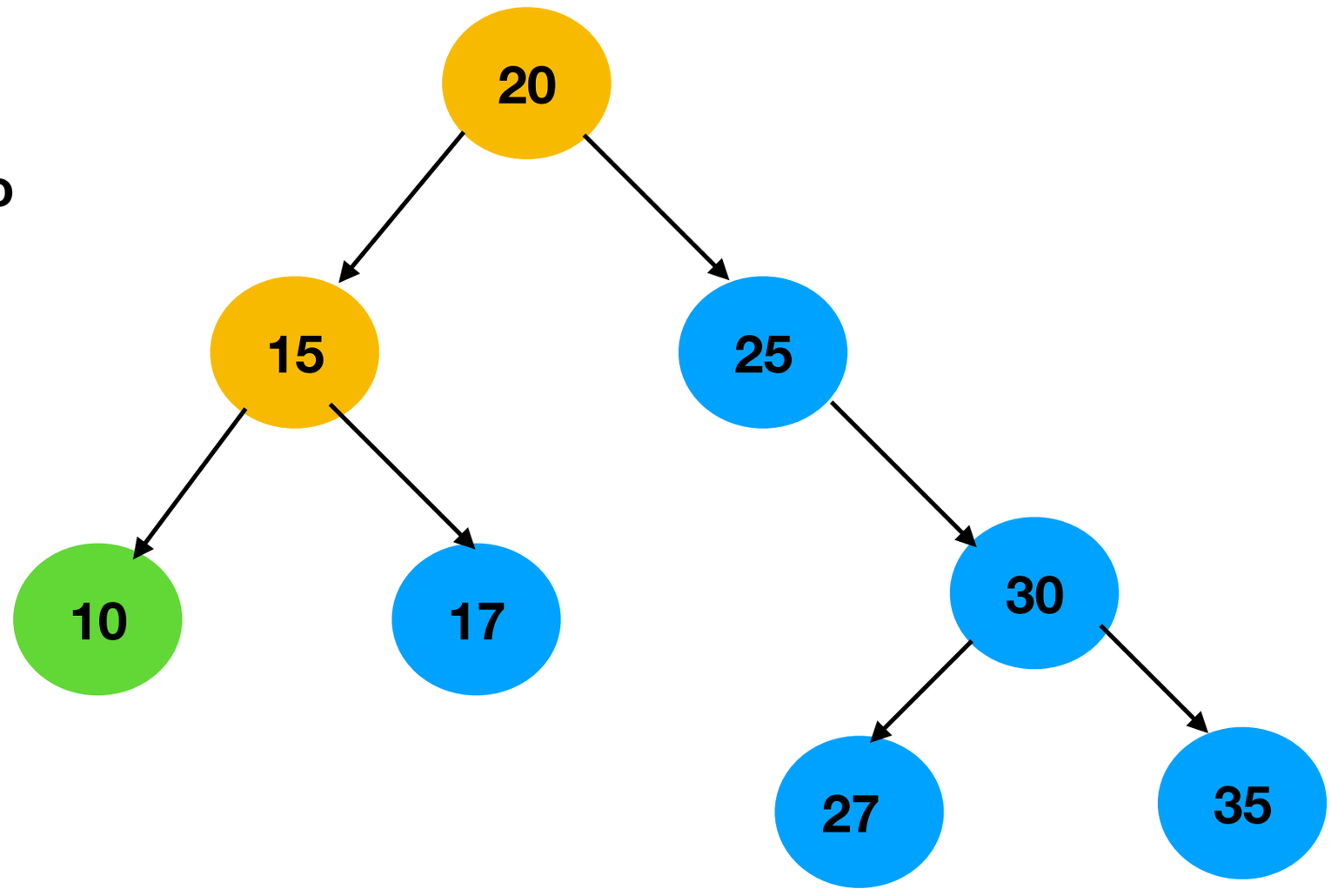
1. Начинаем с корня
2. Посетить левое поддерево
3. Посетить правое поддерево
4. Посетить текущий узел



Обходы дерева

Post-order обход

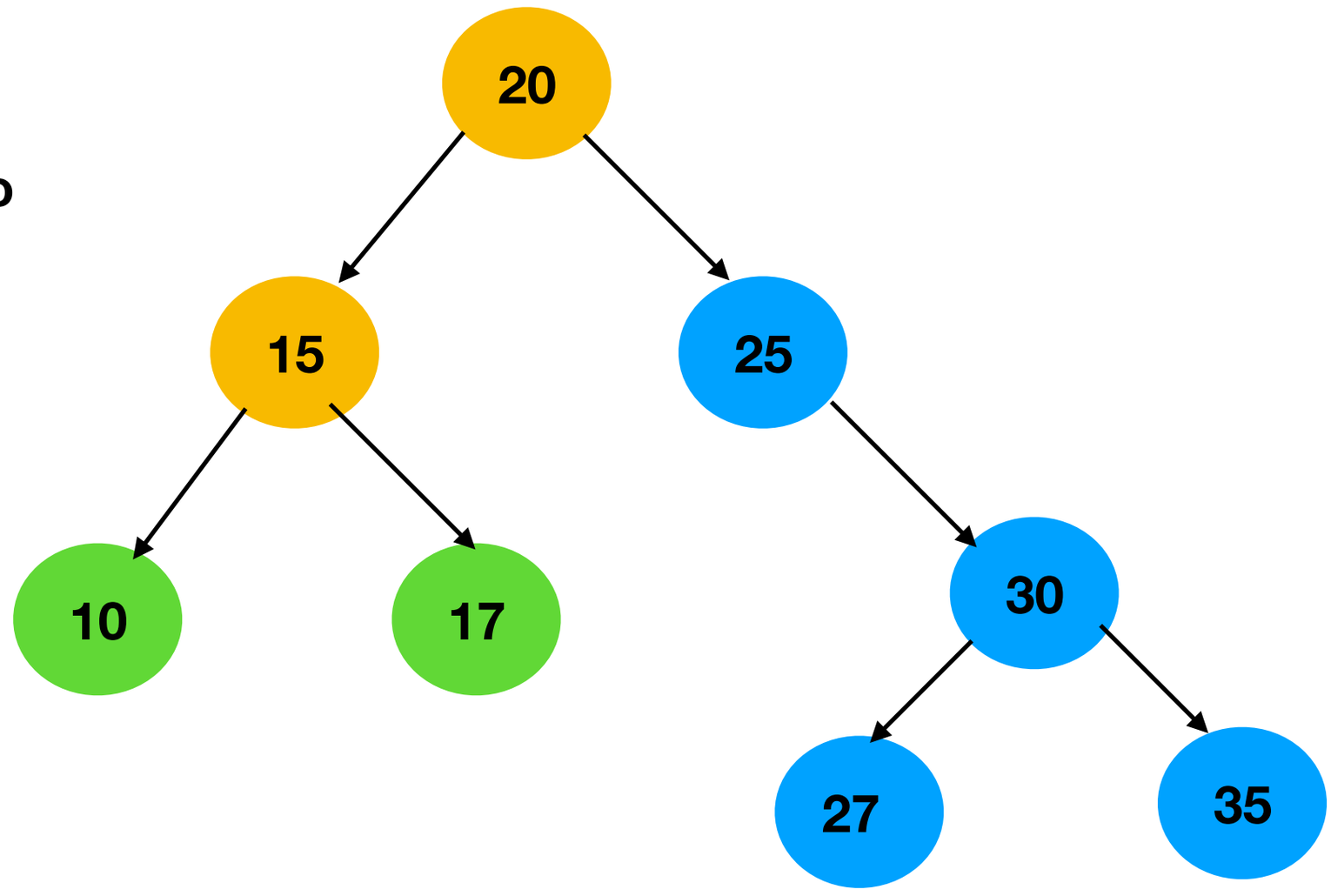
1. Начинаем с корня
2. Посетить левое поддерево
3. Посетить правое поддерево
4. Посетить текущий узел



Обходы дерева

Post-order обход

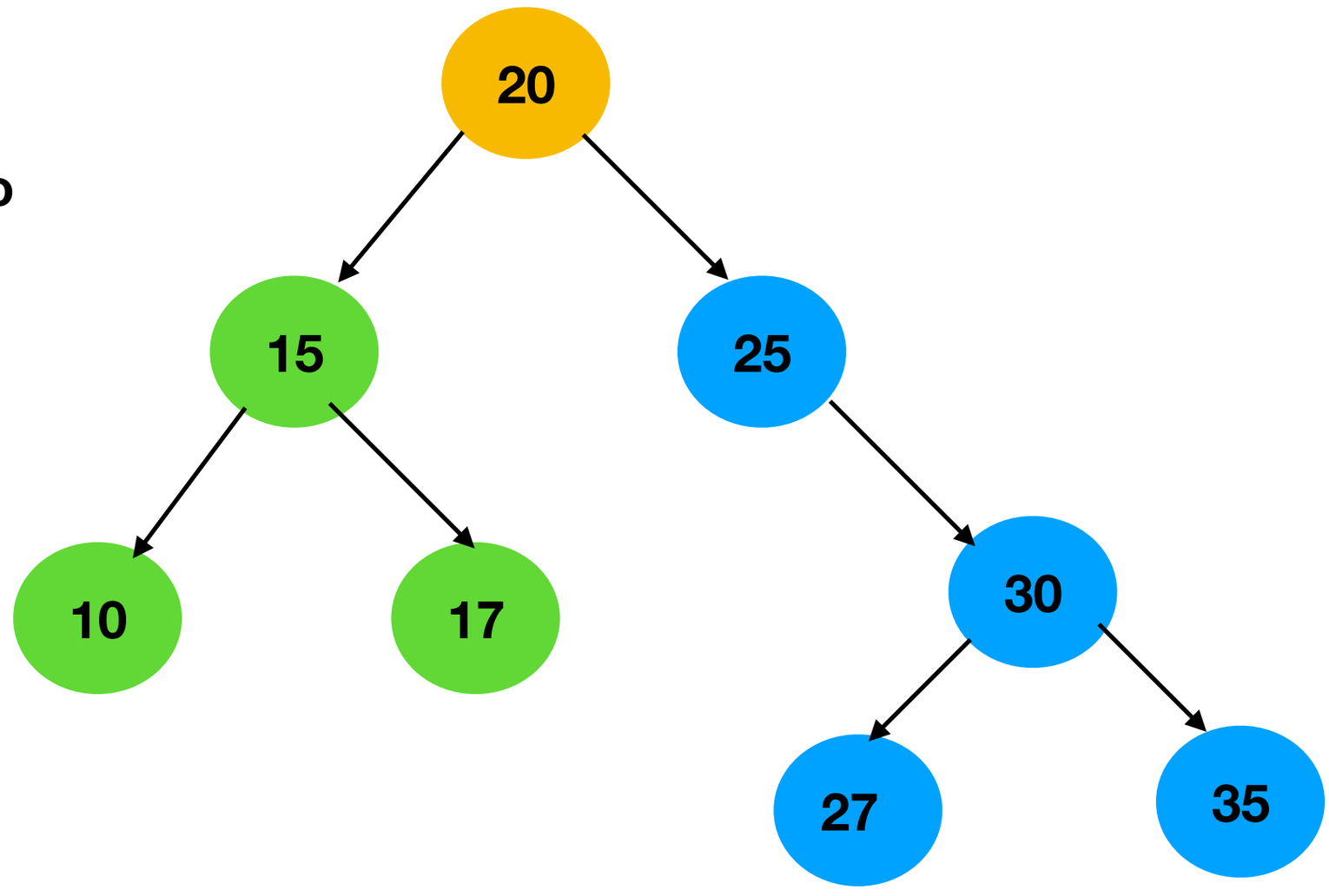
1. Начинаем с корня
2. Посетить левое поддерево
3. Посетить правое поддерево
4. Посетить текущий узел



Обходы дерева

Post-order обход

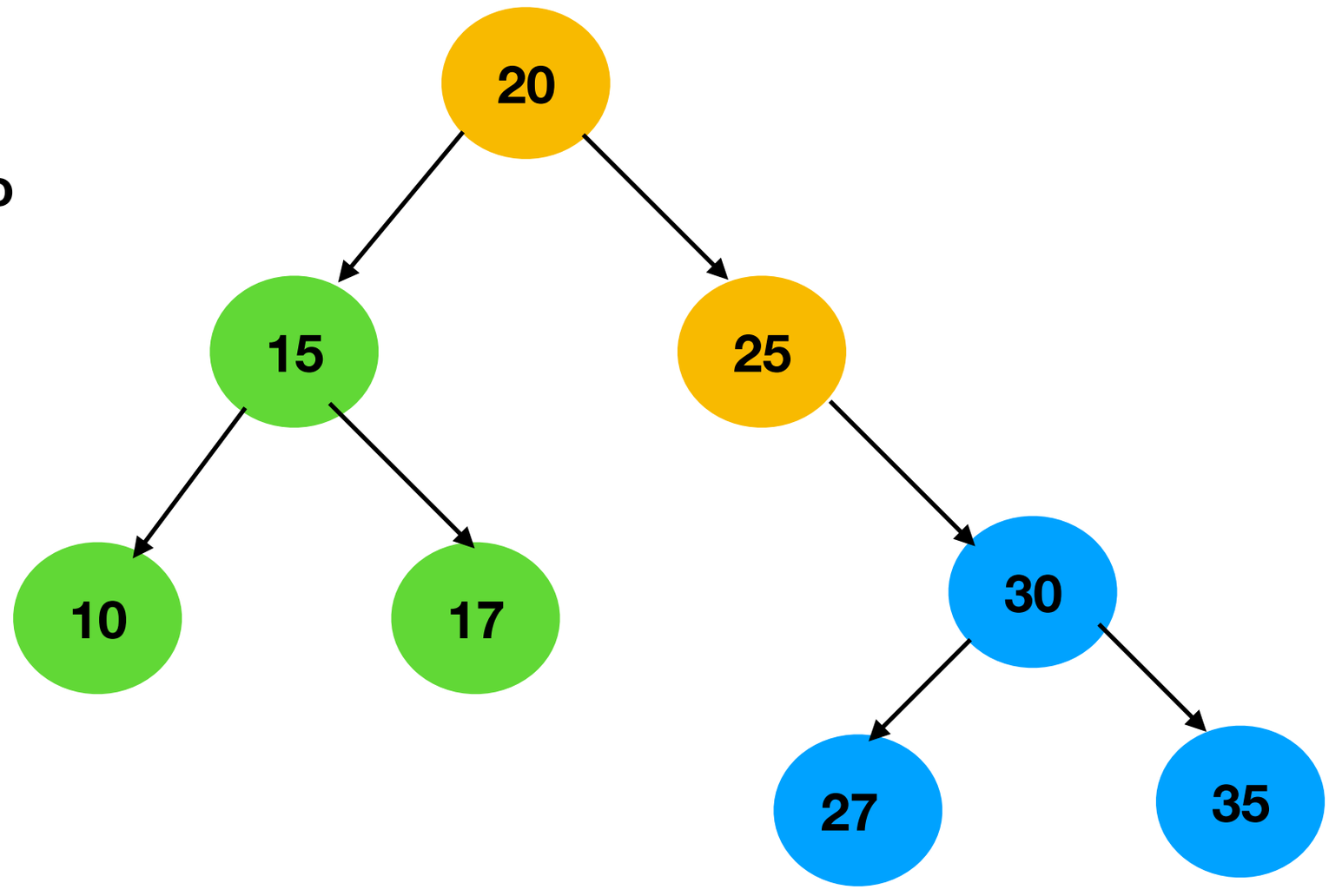
1. Начинаем с корня
2. Посетить левое поддерево
3. Посетить правое поддерево
4. Посетить текущий узел



Обходы дерева

Post-order обход

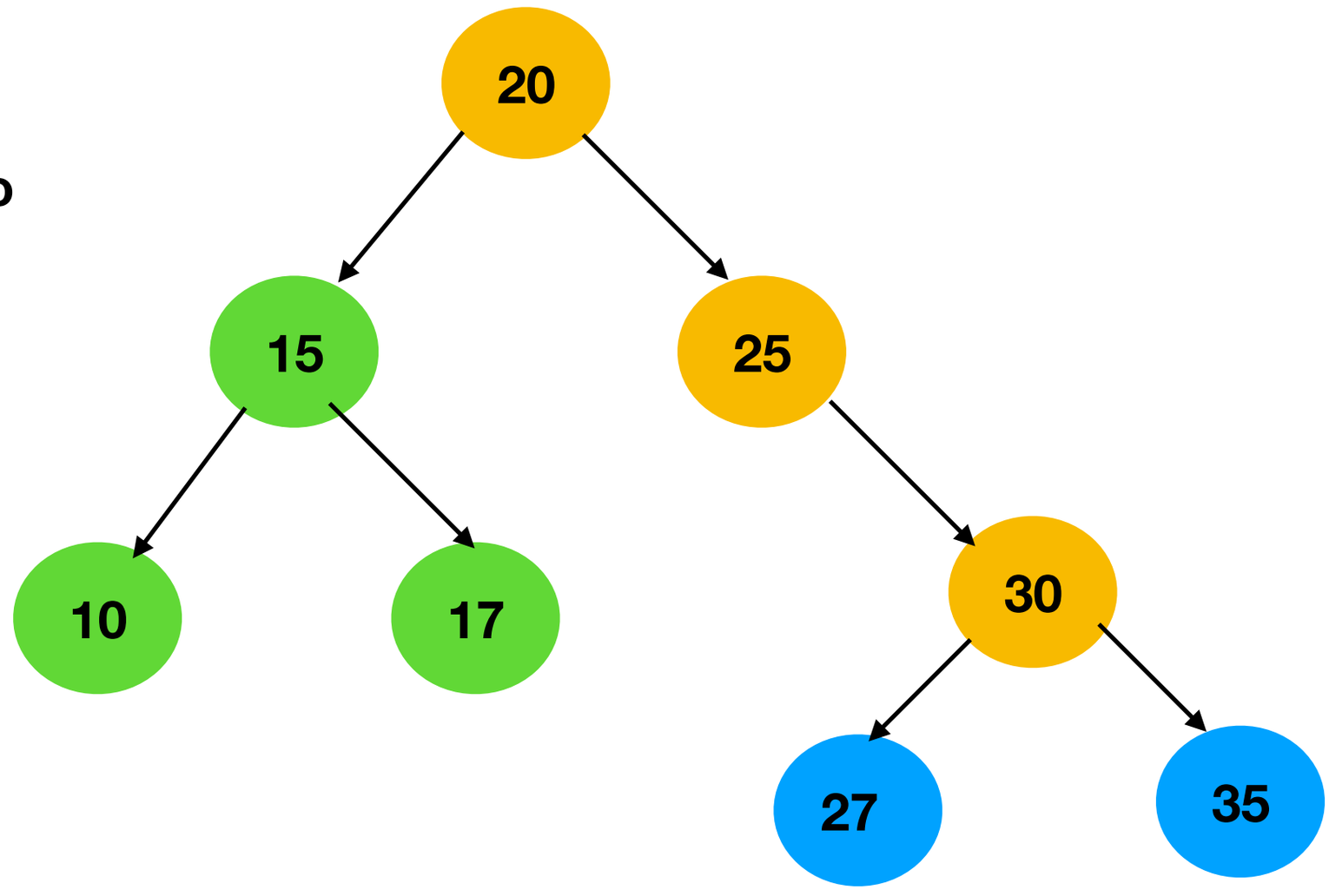
1. Начинаем с корня
2. Посетить левое поддерево
3. Посетить правое поддерево
4. Посетить текущий узел



Обходы дерева

Post-order обход

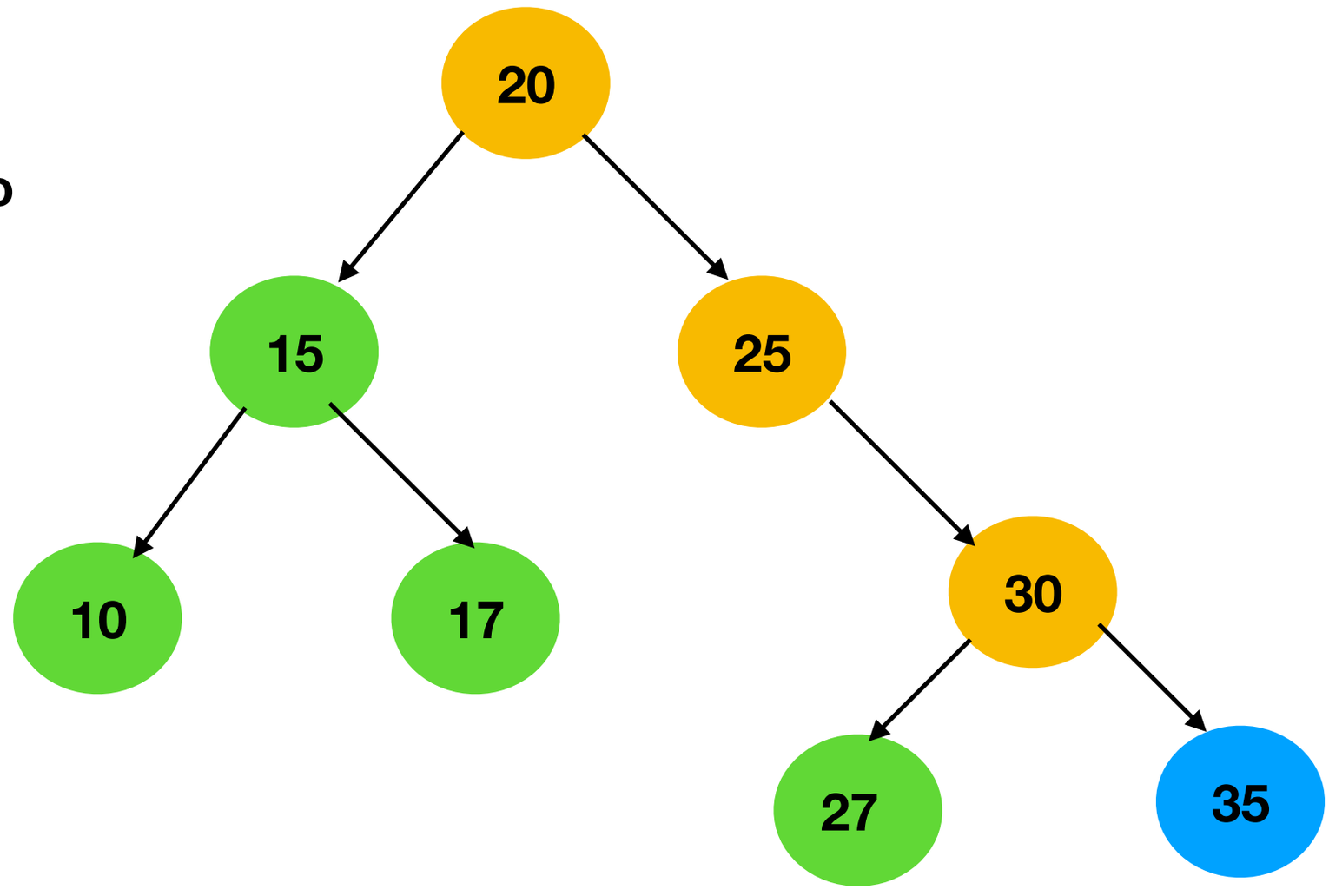
1. Начинаем с корня
2. Посетить левое поддерево
3. Посетить правое поддерево
4. Посетить текущий узел



Обходы дерева

Post-order обход

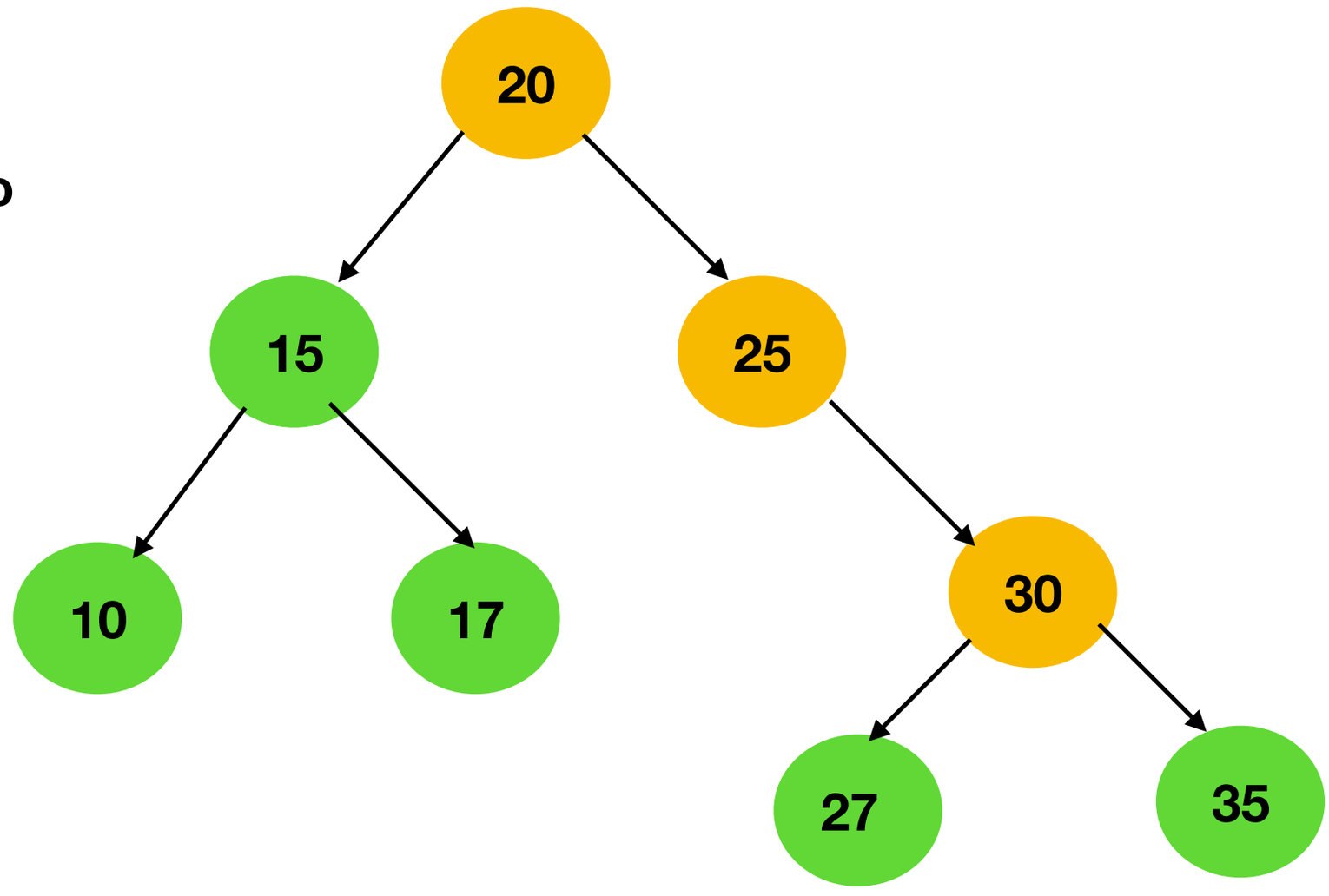
1. Начинаем с корня
2. Посетить левое поддерево
3. Посетить правое поддерево
4. Посетить текущий узел



Обходы дерева

Post-order обход

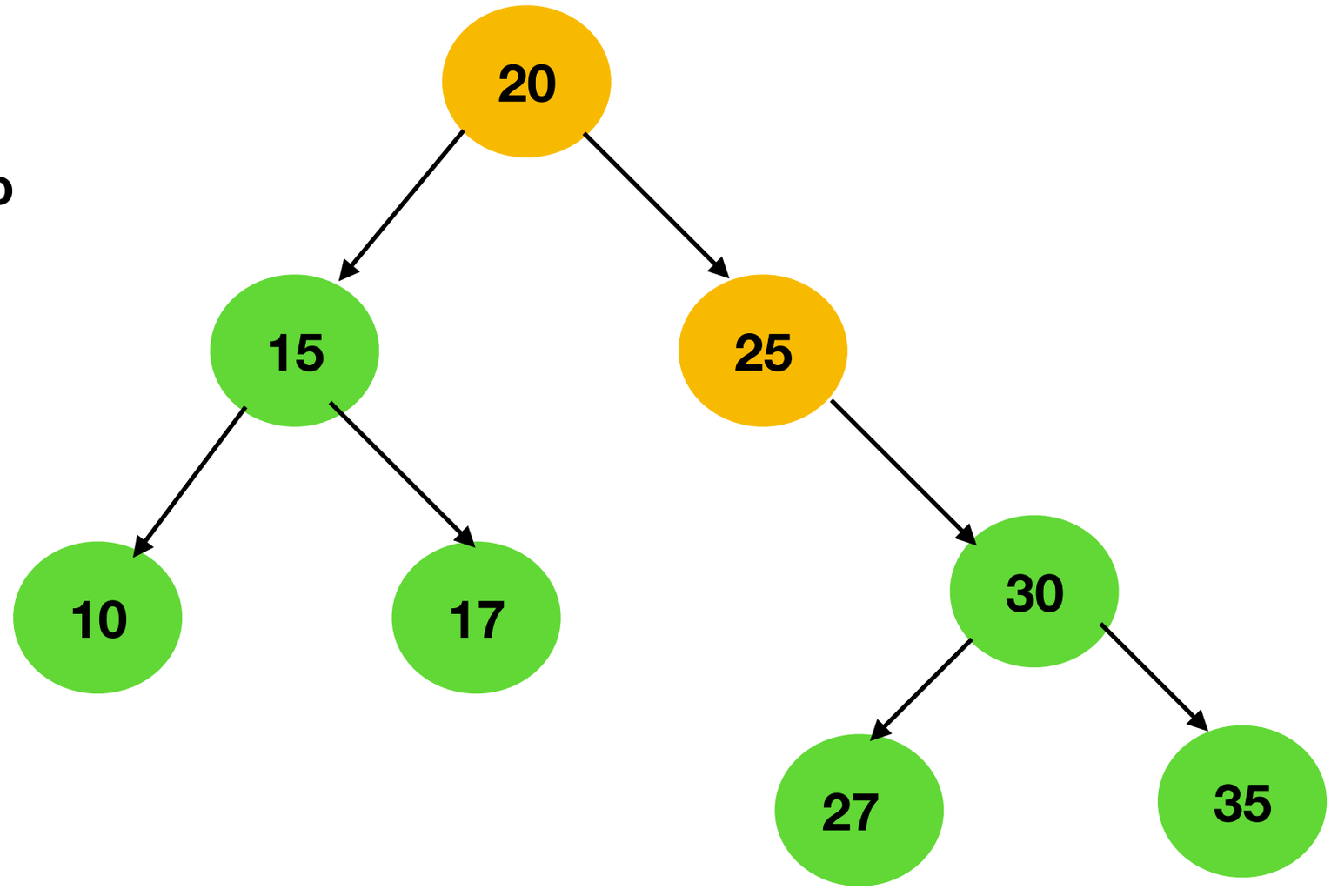
1. Начинаем с корня
2. Посетить левое поддерево
3. Посетить правое поддерево
4. Посетить текущий узел



Обходы дерева

Post-order обход

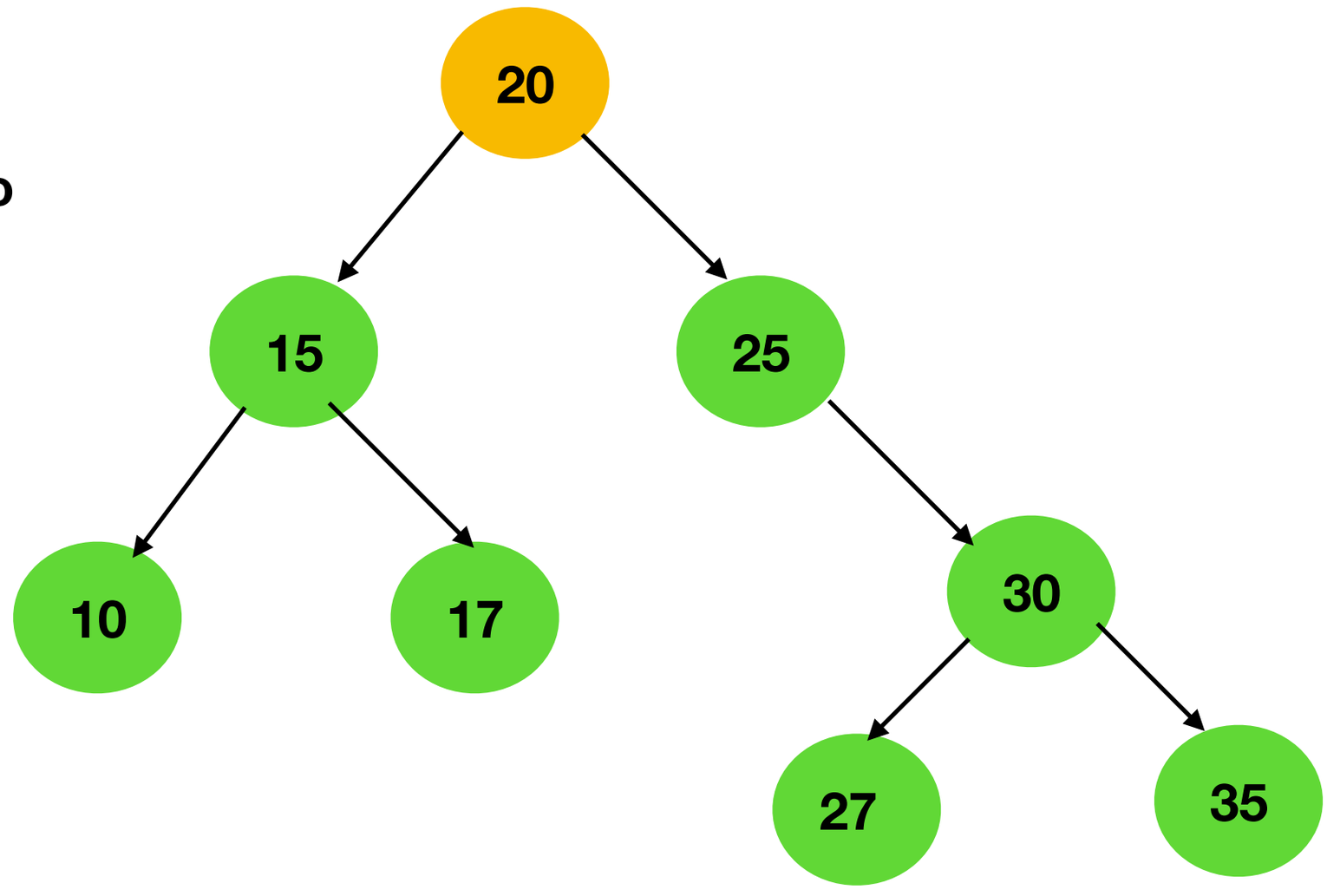
1. Начинаем с корня
2. Посетить левое поддерево
3. Посетить правое поддерево
4. Посетить текущий узел



Обходы дерева

Post-order обход

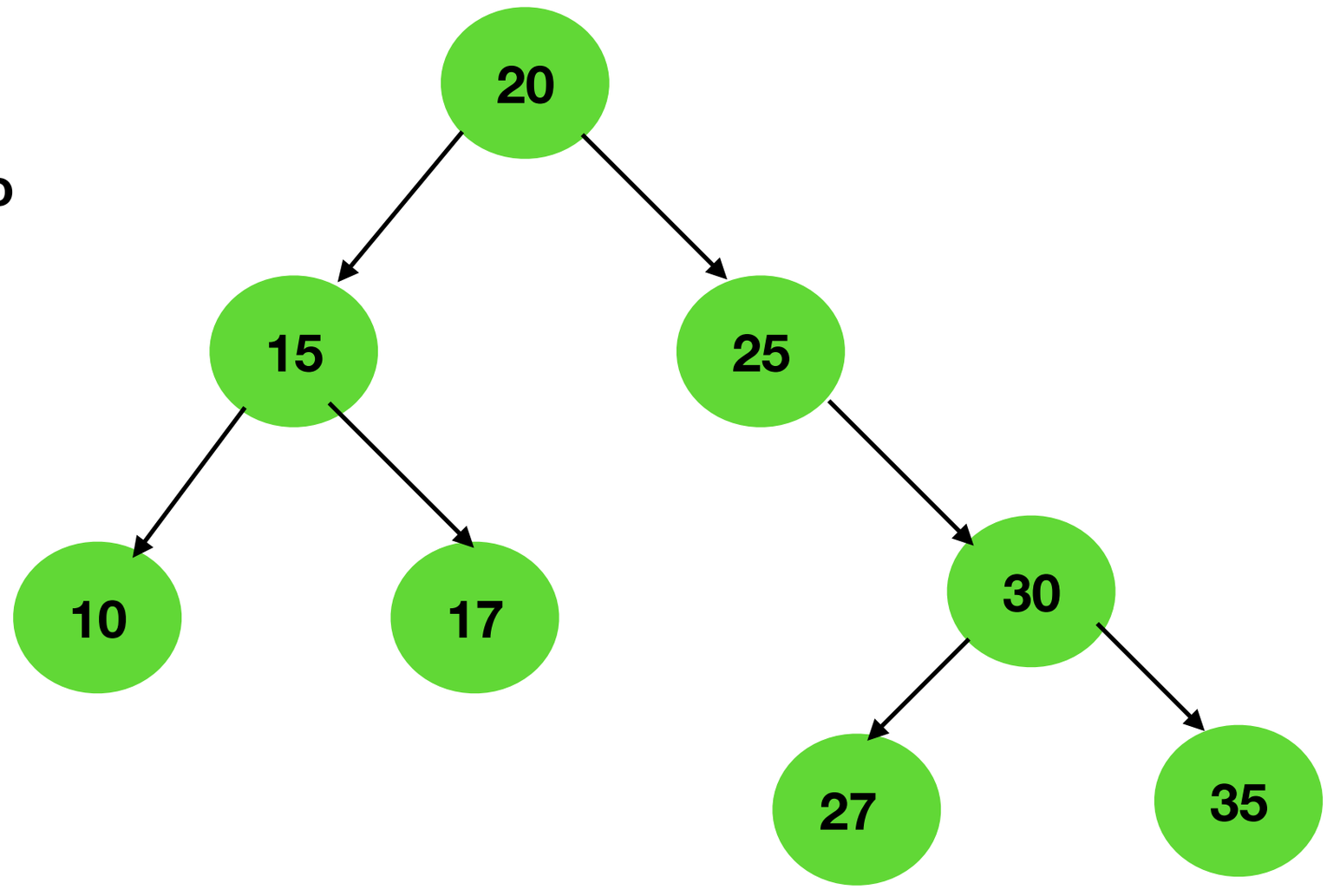
1. Начинаем с корня
2. Посетить левое поддерево
3. Посетить правое поддерево
4. Посетить текущий узел



Обходы дерева

Post-order обход

1. Начинаем с корня
2. Посетить левое поддерево
3. Посетить правое поддерево
4. Посетить текущий узел



Сначала делаем все в детях и только потом в родителях

10, 17, 15, 27, 35, 30, 25, 20

Пишем псевдокод in-order обхода

Пишем псевдокод in-order обхода

```
fn inOrderTraversal(tree: Tree, visit: function) -> None{
    inOrderHelper(tree.root, visit);
}

fn inOrderHelper(node: None, visit: function) -> None{
    if node == None{
        return;
    }
    inOrderHelper(node.left, visit);
    visit(node.value);
    inOrderHelper(node.right, visit);
}
```

Пишем псевдокод in-order обхода

Как заменить рекурсию?

Пишем псевдокод in-order обхода

```
fn inOrderTraversal(tree: Tree, visit: function) -> None{
    stack = Stack();
    if Tree.root == None{
        return;
    }

    stack.push(Tree.root);
    visited = {}
    visited.add(None)

    while !stack.isempty(){
        el = stack.peek();
        if !(el.left in visited){
            stack.push(el.left);

            continue;
        }
        visit(el.value)
        visited.add(el);
        stack.pop();
        stack.push(el.right);
    }
}
```

Как заменить рекурсию?

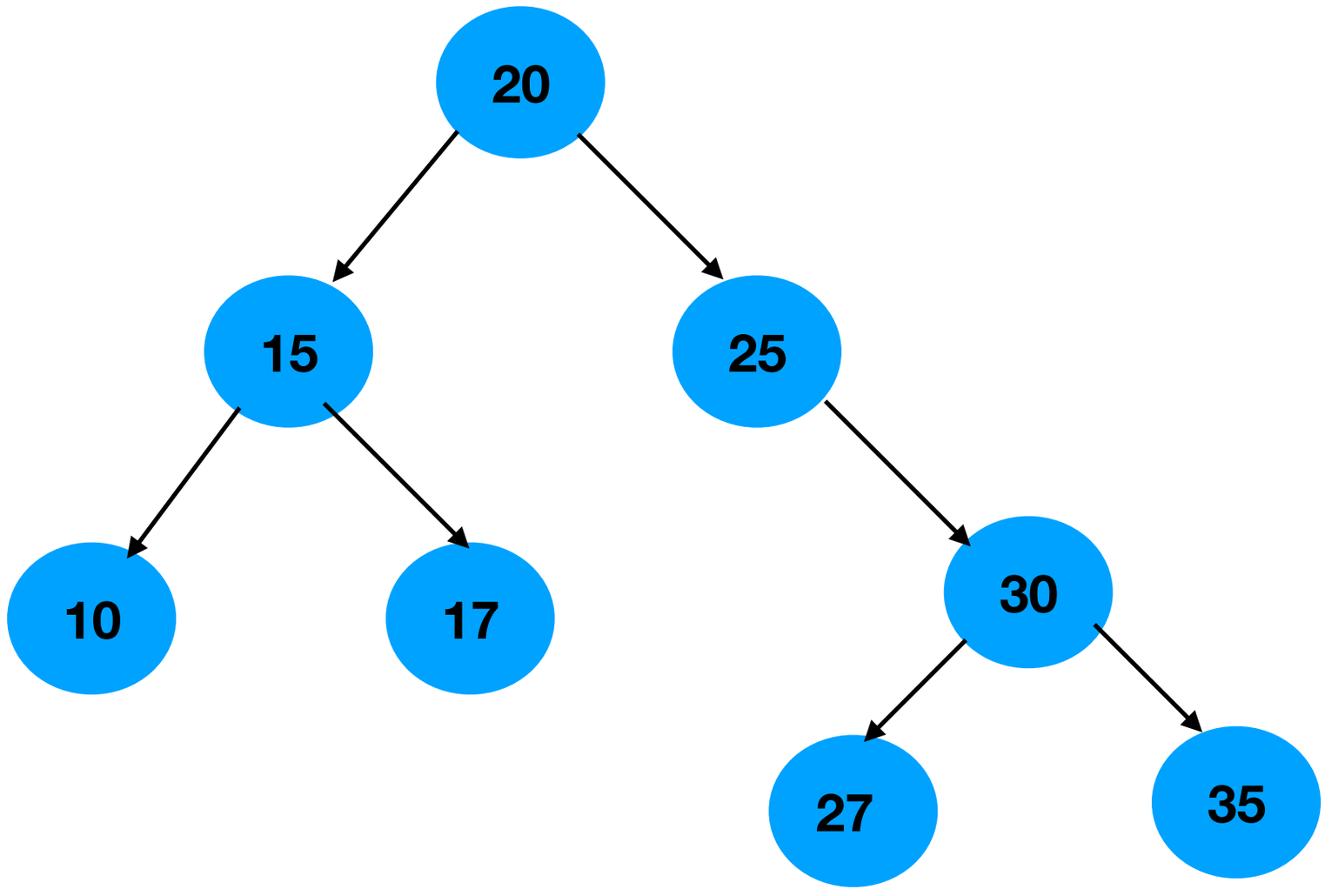
Стеком

(**peek -** посмотреть на элемент на вершине стека, но не убирать)

Обходы дерева

Поуровневый обход

1. Начинаем с корня
2. Посещаем его детей
3. Посещаем их детей
4. ...

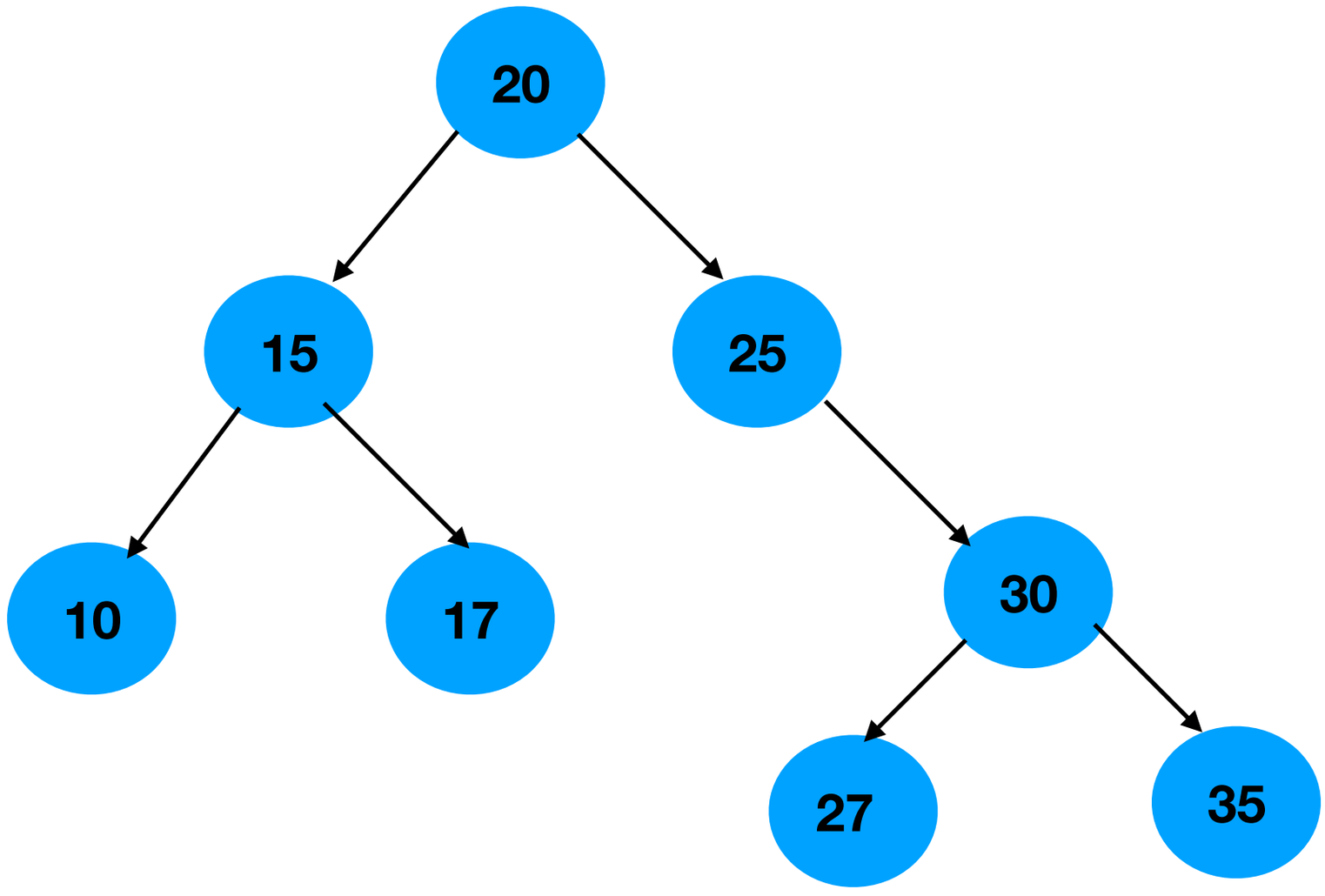


Как сделать?

Обходы дерева

Поуровневый обход

1. Начинаем с корня
2. Посещаем его детей
3. Посещаем их детей
4. ...



Как сделать?

Через очередь

Псевдокод

```
fn byLevelTraversal(tree: Tree, visit: function){
    queue = Queue();
    queue.enqueue(tree.root);
    while !queue.isempty(){
        el = queue.dequeue();
        if el == None{
            continue;
        }
        visit(el);
        queue.enqueue(el.left);
        queue.enqueue(el.right)
    }
}
```

Обход дерева

Обход	Время работы
In-order	O(N)
Post-order	O(N)
Pre-order	O(N)
By-level	O(N)

Можно ли?

Операция	Время работы в среднем случае	Время работы в худшем случае
Вставка	O(logN)	O(logN)
Поиск	O(logN)	O(logN)
Удаление	O(logN)	O(logN)

Можно ли сделать вставку в дерево за O(1)?

Можно ли?

Нет.

Если да, то мы сможем провести следующее:

1) **Вставить в дерево N элементов за O(1)**
2) **Обойти дерево за O(N)**
3) **Сортировка элементов исключительно на основе сравнений, работающая за O(N)**