

Мини-задача #22 (1 балл)

Реализовать сериализацию и десериализацию для бинарных деревьев.

Можно выбрать любую форму, какая вам больше нравится, лишь бы процесс был однозначным.

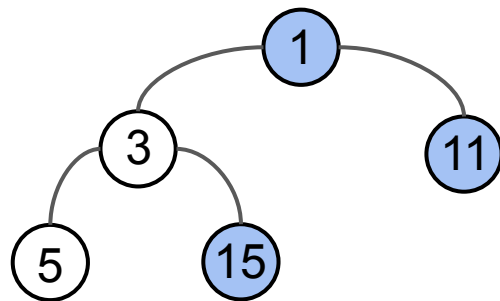
Необходимо проверить свое решение здесь:

<https://leetcode.com/problems/serialize-and-deserialize-binary-tree/>

Мини-задача #23 (1 балл)

Вы стоите справа от бинарного дерева. При этом вы можете видеть только по самой правой вершине в каждом из уровней этого дерева.

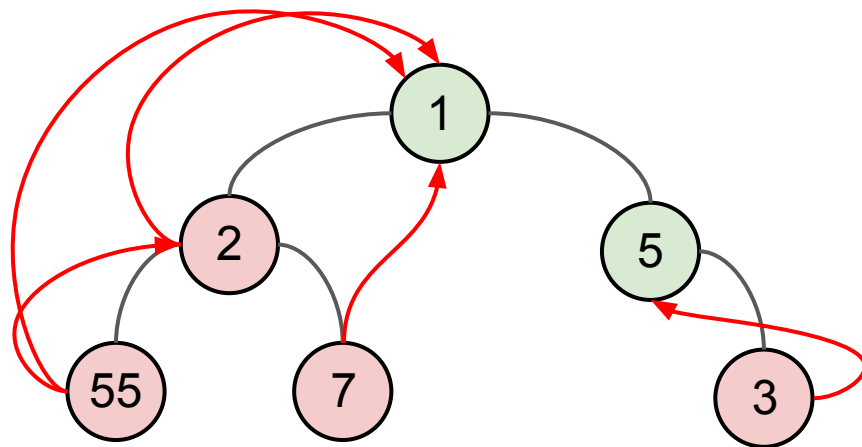
Напишите программу, которая напечатает все видимые для вас вершины в порядке от корня к листьям.



<https://leetcode.com/problems/binary-tree-right-side-view>

Мини-задача #24 (1 балл)

Проверить, является ли данное вам бинарное дерево бинарным деревом **поиска**.



Мини-задача #25 (1 балл)

Реализовать процедуру `trim(root, left, right)`, которая принимает BST и **удаляет** из него все вершины, значения которых не попадают в интервал `[left, right]`.

При этом относительный порядок оставшихся вершин должен совпадать с изначальным, т.е. если какая-то вершина попадала была наследником другой выжившей вершины, то так и должно остаться.

Утверждается, что решение существует единственное.

<https://leetcode.com/problems/trim-a-binary-search-tree/>

Алгоритмы и структуры данных

Деревья поиска, сбалансированные деревья



Деревья



Деревья

Поиска



Задача

Нужна структура данных с **быстрым поиском**
элементов **по значению**.

Как раз то, с чем у
пирамид были проблемы

Задача

Нужна структура данных с **быстрым поиском** элементов **по значению**.

Как реализовывать?

Как раз то, с чем у
пирамид были проблемы

Задача

Нужна структура данных с **быстрым поиском** элементов **по значению**.

Как реализовывать?

Попробуем отсортированный массив.

Как раз то, с чем у
пирамид были проблемы

Отсортированный массив

0	2	12	17	33	45	54	92
---	---	----	----	----	----	----	----

Отсортированный массив

0	2	12	17	33	45	54	92
---	---	----	----	----	----	----	----

Операции:

1. `find(value)` -> поиск элемента в массиве. Сложность?

Отсортированный массив

0	2	12	17	33	45	54	92
---	---	----	----	----	----	----	----

Операции:

1. `find(value)` $\rightarrow O(\log N)$ // бинарный поиск

Отсортированный массив

0	2	12	17	33	45	54	92
---	---	----	----	----	----	----	----

Операции:

1. `find(value)` $\rightarrow O(\log N)$ // **бинарный** поиск
2. `select(i)` \rightarrow взятие i -ой статистики. Сложность?

Отсортированный массив

0	2	12	17	33	45	54	92
---	---	----	----	----	----	----	----

Операции:

1. `find(value)` $\rightarrow O(\log N)$ // бинарный поиск
2. `select(i)` $\rightarrow O(1)$ // 🎉 🎉 🎉

Отсортированный массив

0	2	12	17	33	45	54	92
---	---	----	----	----	----	----	----

Операции:

1. `find(value)` $\rightarrow O(\log N)$ // бинарный поиск
2. `select(i)` $\rightarrow O(1)$ // 🎉 🎉 🎉
3. `min/max` $\rightarrow O(1)$ // завидуйте, пирамиды

Отсортированный массив

0	2	12	17	33	45	54	92
---	---	----	----	----	----	----	----

Операции:

1. `find(value)` $\rightarrow O(\log N)$ // бинарный поиск
2. `select(i)` $\rightarrow O(1)$ // 🎉 🎉 🎉
3. `min/max` $\rightarrow O(1)$ // завидуйте, пирамиды
4. `pred/succ` \rightarrow по элементу в массиве найти
ближайшие по значению к нему элементы

Отсортированный массив

0	2	12	17	33	45	54	92
---	---	----	----	----	----	----	----

Операции:

1. `find(value)` $\rightarrow O(\log N)$ // бинарный поиск
2. `select(i)` $\rightarrow O(1)$ // 🎉 🎉 🎉
3. `min/max` $\rightarrow O(1)$ // завидуйте, пирамиды
4. `pred/succ` $\rightarrow O(1)$ // 🎉 🎉 🎉

Отсортированный массив

0	2	12	17	33	45	54	92
---	---	----	----	----	----	----	----

Операции:

1. `find(value)` $\rightarrow O(\log N)$ // бинарный поиск
2. `select(i)` $\rightarrow O(1)$ // 🎉 🎉 🎉
3. `min/max` $\rightarrow O(1)$ // завидуйте, пирамиды
4. `pred/succ(ptr)` $\rightarrow O(1)$ // 🎉 🎉 🎉
5. `rank(value)` \rightarrow сколько есть элементов, меньше либо равно, чем наш элемент

Отсортированный массив

0	2	12	17	33	45	54	92
---	---	----	----	----	----	----	----

Операции:

1. `find(value)` $\rightarrow O(\log N)$ // бинарный поиск
2. `select(i)` $\rightarrow O(1)$ // 🎉 🎉 🎉
3. `min/max` $\rightarrow O(1)$ // завидуйте, пирамиды
4. `pred/succ(ptr)` $\rightarrow O(1)$ // 🎉 🎉 🎉
5. `rank(value)` \rightarrow сколько есть элементов, меньше либо равно, чем наш элемент
`rank(45) == 6`

Отсортированный массив

0	2	12	17	33	45	54	92
---	---	----	----	----	----	----	----

Операции:

1. `find(value)` $\rightarrow O(\log N)$ // бинарный поиск
2. `select(i)` $\rightarrow O(1)$ // 🎉 🎉 🎉
3. `min/max` $\rightarrow O(1)$ // завидуйте, пирамиды
4. `pred/succ(ptr)` $\rightarrow O(1)$ // 🎉 🎉 🎉
5. `rank(value)` \rightarrow сколько есть элементов, меньше либо равно, чем наш элемент
`rank(40) == 5`

Отсортированный массив

0	2	12	17	33	45	54	92
---	---	----	----	----	----	----	----

Операции:

1. `find(value)` $\rightarrow O(\log N)$ // бинарный поиск
2. `select(i)` $\rightarrow O(1)$ // 🎉 🎉 🎉
3. `min/max` $\rightarrow O(1)$ // завидуйте, пирамиды
4. `pred/succ(ptr)` $\rightarrow O(1)$ // 🎉 🎉 🎉
5. `rank(value)` $\rightarrow O(\log N)$ // по сути - бинарный поиск

Отсортированный массив

0	2	12	17	33	45	54	92
---	---	----	----	----	----	----	----

Операции:

1. `find(value)` $\rightarrow O(\log N)$ // бинарный поиск
2. `select(i)` $\rightarrow O(1)$ // 🎉 🎉 🎉
3. `min/max` $\rightarrow O(1)$ // завидуйте, пирамиды
4. `pred/succ(ptr)` $\rightarrow O(1)$ // 🎉 🎉 🎉
5. `rank(value)` $\rightarrow O(\log N)$ // по сути - бинарный поиск
6. вывод в порядке возрастания

Отсортированный массив

0	2	12	17	33	45	54	92
---	---	----	----	----	----	----	----

Операции:

1. `find(value)` $\rightarrow O(\log N)$ // бинарный поиск
2. `select(i)` $\rightarrow O(1)$ // 🎉 🎉 🎉
3. `min/max` $\rightarrow O(1)$ // завидуйте, пирамиды
4. `pred/succ(ptr)` $\rightarrow O(1)$ // 🎉 🎉 🎉
5. `rank(value)` $\rightarrow O(\log N)$ // по сути - бинарный поиск
6. вывод в порядке возрастания $\rightarrow O(N)$ // 🎉 🎉 🎉

0	2	12	17	33	45	54	92
---	---	----	----	----	----	----	----

Операции:

1. find(value) -> $O(\log N)$ // бинарный поиск
2. select(i) -> $O(1)$ // 🎉 🎉 🎉
3. min/max -> $O(1)$ // завидуйте, пирамиды
4. pred/succ(ptr) -> $O(1)$ // 🎉 🎉 🎉
5. rank(value) -> $O(\log N)$ // по сути - бинарный поиск
6. вывод в пор. возрастания -> $O(N)$ // 🎉 🎉 🎉

0	2	12	17	33	45	54	92
---	---	----	----	----	----	----	----

Операции:

1. `find(value)` $\rightarrow O(\log N)$ // бинарный поиск
 2. `select(i)` $\rightarrow O(1)$ // 🎉 🎉 🎉
 3. `min/max` $\rightarrow O(1)$ // завидуйте, пирамиды
 4. `pred/succ(ptr)` $\rightarrow O(1)$ // 🎉 🎉 🎉
 5. `rank(value)` $\rightarrow O(\log N)$ // по сути - бинарный поиск
 6. вывод в пор. возрастания $\rightarrow O(N)$ // 🎉 🎉 🎉
-

0	2	12	17	33	45	54	92
---	---	----	----	----	----	----	----

Операции:

1. `find(value)` $\rightarrow O(\log N)$ // бинарный поиск
 2. `select(i)` $\rightarrow O(1)$ // 🎉 🎉 🎉
 3. `min/max` $\rightarrow O(1)$ // завидуйте, пирамиды
 4. `pred/succ(ptr)` $\rightarrow O(1)$ // 🎉 🎉 🎉
 5. `rank(value)` $\rightarrow O(\log N)$ // по сути - бинарный поиск
 6. вывод в пор. возрастания $\rightarrow O(N)$ // 🎉 🎉 🎉
-

7. `insert(value)` \rightarrow Сложность?

insert(20)?

0	2	12	17	33	45	54	92
---	---	----	----	----	----	----	----

Операции:

1. find(value) $\rightarrow O(\log N)$ // бинарный поиск
 2. select(i) $\rightarrow O(1)$ // 🎉 🎉 🎉
 3. min/max $\rightarrow O(1)$ // завидуйте, пирамиды
 4. pred/succ(ptr) $\rightarrow O(1)$ // 🎉 🎉 🎉
 5. rank(value) $\rightarrow O(\log N)$ // по сути - бинарный поиск
 6. вывод в пор. возрастания $\rightarrow O(N)$ // 🎉 🎉 🎉
-

7. insert(value) \rightarrow Сложность?

insert(20)?

0	2	12	17	33	45	54	92
---	---	----	----	----	----	----	----



Операции:

1. find(value) $\rightarrow O(\log N)$ // бинарный поиск
 2. select(i) $\rightarrow O(1)$ // 🎉 🎉 🎉
 3. min/max $\rightarrow O(1)$ // завидуйте, пирамиды
 4. pred/succ(ptr) $\rightarrow O(1)$ // 🎉 🎉 🎉
 5. rank(value) $\rightarrow O(\log N)$ // по сути - бинарный поиск
 6. вывод в пор. возрастания $\rightarrow O(N)$ // 🎉 🎉 🎉
-

7. insert(value) \rightarrow Сложность?

insert(20)?

0	2	12	17
---	---	----	----

33	45	54	92
----	----	----	----



Операции:

1. find(value) $\rightarrow O(\log N)$ // бинарный поиск
 2. select(i) $\rightarrow O(1)$ // 🎉 🎉 🎉
 3. min/max $\rightarrow O(1)$ // завидуйте, пирамиды
 4. pred/succ(ptr) $\rightarrow O(1)$ // 🎉 🎉 🎉
 5. rank(value) $\rightarrow O(\log N)$ // по сути - бинарный поиск
 6. вывод в пор. возрастания $\rightarrow O(N)$ // 🎉 🎉 🎉
-

7. insert(value) \rightarrow Сложность?

insert(20)?

0	2	12	17	20	33	45	54	92
---	---	----	----	----	----	----	----	----

Операции:

1. find(value) $\rightarrow O(\log N)$ // бинарный поиск
 2. select(i) $\rightarrow O(1)$ // 🎉 🎉 🎉
 3. min/max $\rightarrow O(1)$ // завидуйте, пирамиды
 4. pred/succ(ptr) $\rightarrow O(1)$ // 🎉 🎉 🎉
 5. rank(value) $\rightarrow O(\log N)$ // по сути - бинарный поиск
 6. вывод в пор. возрастания $\rightarrow O(N)$ // 🎉 🎉 🎉
-

7. insert(value) \rightarrow Сложность?

insert(20)?

0	2	12	17	20	33	45	54	92
---	---	----	----	----	----	----	----	----

Операции:

1. find(value) $\rightarrow O(\log N)$ // бинарный поиск
 2. select(i) $\rightarrow O(1)$ // 🎉 🎉 🎉
 3. min/max $\rightarrow O(1)$ // завидуйте, пирамиды
 4. pred/succ(ptr) $\rightarrow O(1)$ // 🎉 🎉 🎉
 5. rank(value) $\rightarrow O(\log N)$ // по сути - бинарный поиск
 6. вывод в пор. возрастания $\rightarrow O(N)$ // 🎉 🎉 🎉
-

7. insert(value) $\rightarrow O(N)$ // 😞

insert(20)?

0	2	12	17	20	33	45	54	92
---	---	----	----	----	----	----	----	----

Операции:

1. find(value) $\rightarrow O(\log N)$ // бинарный поиск
 2. select(i) $\rightarrow O(1)$ // 🎉 🎉 🎉
 3. min/max $\rightarrow O(1)$ // завидуйте, пирамиды
 4. pred/succ(ptr) $\rightarrow O(1)$ // 🎉 🎉 🎉
 5. rank(value) $\rightarrow O(\log N)$ // по сути - бинарный поиск
 6. вывод в пор. возрастания $\rightarrow O(N)$ // 🎉 🎉 🎉
-
7. insert(value) $\rightarrow O(N)$ // 😞
 8. remove(value) $\rightarrow O(N)$ // 😞

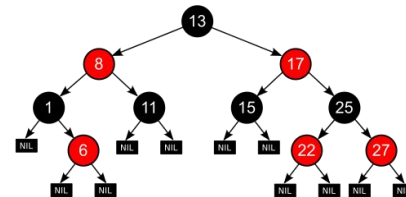
0	2	12	17	20	33	45	54	92
---	---	----	----	----	----	----	----	----

Операции:

1. `find(value)` $\rightarrow O(\log N)$ // бинарный поиск
 2. `select(i)` $\rightarrow O(1)$ // 🎉 🎉 🎉
 3. `min/max` $\rightarrow O(1)$ // завидуйте, пирамиды
 4. `pred/succ(ptr)` $\rightarrow O(1)$ // 🎉 🎉 🎉
 5. `rank(value)` $\rightarrow O(\log N)$ // по сути - бинарный поиск
 6. вывод в пор. возрастания $\rightarrow O(N)$ // 🎉 🎉 🎉
-

7. `insert(value)` $\rightarrow O(N)$ // 😞 Можем ли мы
8. `remove(value)` $\rightarrow O(N)$ // 😞 лучше?

Сбалансированные деревья поиска

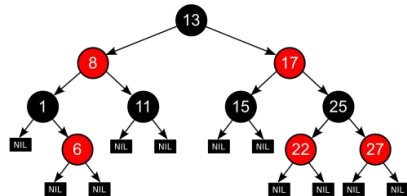


Операции:

1. `find(value)` $\rightarrow O(???)$
2. `select(i)` $\rightarrow O(???)$
3. `min/max` $\rightarrow O(???)$
4. `pred/succ(ptr)` $\rightarrow O(???)$
5. `rank(value)` $\rightarrow O(???)$
6. вывод в пор. $\rightarrow O(???)$
 возрастания

-
7. `insert(value)` $\rightarrow O(???)$
 8. `remove(value)` $\rightarrow O(???)$

Сбалансированные деревья поиска

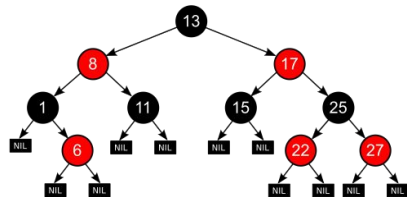


Операции:

1. `find(value)` $\rightarrow O(\log N)$
2. `select(i)` $\rightarrow O(???)$
3. `min/max` $\rightarrow O(???)$
4. `pred/succ(ptr)` $\rightarrow O(???)$
5. `rank(value)` $\rightarrow O(\log N)$
6. вывод в пор.
возрастания $\rightarrow O(N)$

-
7. `insert(value)` $\rightarrow O(\log N)$
 8. `remove(value)` $\rightarrow O(\log N)$

Сбалансированные деревья поиска



Операции:

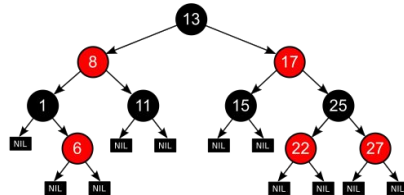
1. `find(value)` $\rightarrow O(\log N)$
2. `select(i)` $\rightarrow O(???)$
3. `min/max` $\rightarrow O(???)$
4. `pred/succ(ptr)` $\rightarrow O(???)$
5. `rank(value)` $\rightarrow O(\log N)$
6. вывод в пор.
возрастания $\rightarrow O(N)$

Поиск не хуже

-
7. `insert(value)` $\rightarrow O(\log N)$
 8. `remove(value)` $\rightarrow O(\log N)$

Добавление/удаление – лучше

Сбалансированные деревья поиска



Операции:

1. find(value) $\rightarrow O(\log N)$
2. select(i) $\rightarrow O(\log N)$
3. min/max $\rightarrow O(\log N)$
4. pred/succ(ptr) $\rightarrow O(\log N)$
5. rank(value) $\rightarrow O(\log N)$
6. вывод в пор. возрастания $\rightarrow O(N)$

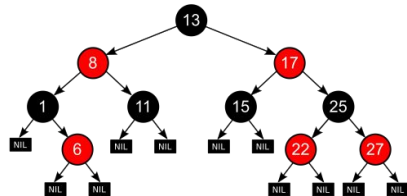
Поиск не хуже

Статистики и порядки
хуже, но не сильно

-
7. insert(value) $\rightarrow O(\log N)$
 8. remove(value) $\rightarrow O(\log N)$

Добавление/удаление - лучше

Сбалансированные деревья поиска



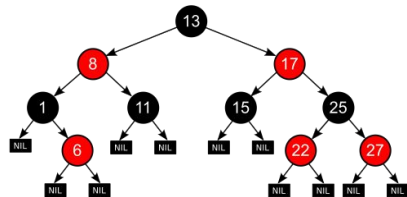
Операции:

1. find(value) $\rightarrow O(\log N)$
 2. select(i) $\rightarrow O(\log N)$
 3. min/max $\rightarrow O(\log N)$
 4. pred/succ(ptr) $\rightarrow O(\log N)$
 5. rank(value) $\rightarrow O(\log N)$
 6. вывод в пор.
возрастания $\rightarrow O(N)$
-

7. insert(value) $\rightarrow O(\log N)$
8. remove(value) $\rightarrow O(\log N)$

Пирамиды лучше здесь
(по константам или
даже по асимптотике)

Сбалансированные деревья поиска



Операции:

1. `find(value)` $\rightarrow O(\log N)$
 2. `select(i)` $\rightarrow O(\log N)$
 3. `min/max` $\rightarrow O(\log N)$
 4. `pred/succ(ptr)` $\rightarrow O(\log N)$
 5. `rank(value)` $\rightarrow O(\log N)$
 6. вывод в пор.
возрастания $\rightarrow O(N)$
-

7. `insert(value)` $\rightarrow O(\log N)$
8. `remove(value)` $\rightarrow O(\log N)$

Пирамиды лучше здесь
(по константам или
даже по асимптотике)

Спойлер: хеш-таблицы
лучше здесь (при
правильной реализации
дадут $O(1)$)

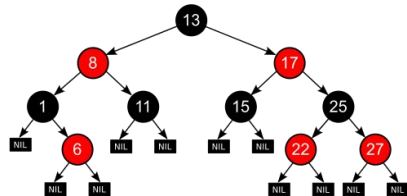
Отсортированный массив

0	2	12	17	33	45	54	92
---	---	----	----	----	----	----	----

Операции:

1. `find(value)` $\rightarrow O(\log N)$ // бинарный поиск
2. `select(i)` $\rightarrow O(1)$ // 🎉 🎉 🎉
3. `min/max` $\rightarrow O(1)$ // завидуйте, пирамиды
4. `pred/succ(ptr)` $\rightarrow O(1)$ // 🎉 🎉 🎉
5. `rank(value)` $\rightarrow O(\log N)$ // по сути - бинарный поиск
6. вывод в порядке возрастания $\rightarrow O(N)$ // 🎉 🎉 🎉

Сбалансированные деревья поиска



Операции:

1. `find(value)` $\rightarrow O(\log N)$
 2. `select(i)` $\rightarrow O(\log N)$
 3. `min/max` $\rightarrow O(\log N)$
 4. `pred/succ(ptr)` $\rightarrow O(\log N)$
 5. `rank(value)` $\rightarrow O(\log N)$
 6. вывод в пор.
возрастания $\rightarrow O(N)$
-

7. `insert(value)` $\rightarrow O(\log N)$
8. `remove(value)` $\rightarrow O(\log N)$

Деревья поиска дают самый широкий набор операций

Пирамиды лучше здесь
(по константам или даже по асимптотике)

Спойлер: хеш-таблицы лучше здесь (при правильной реализации дадут $O(1)$)

Деревья

Дерево

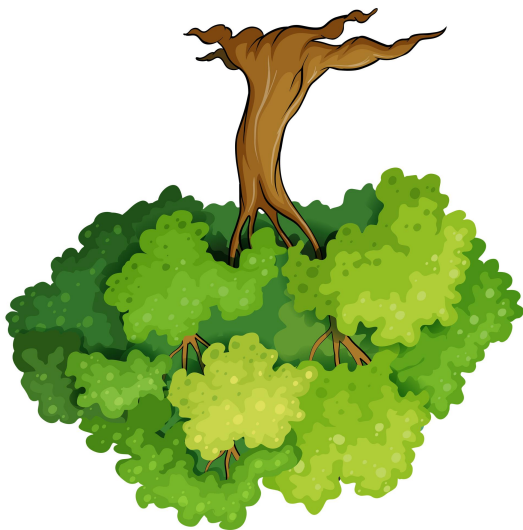


Деревья

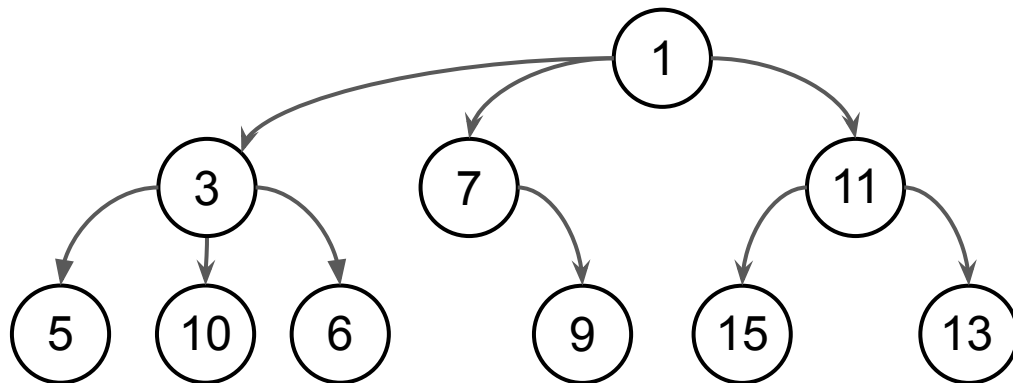


Дерево – связный
ациклический граф

Деревья



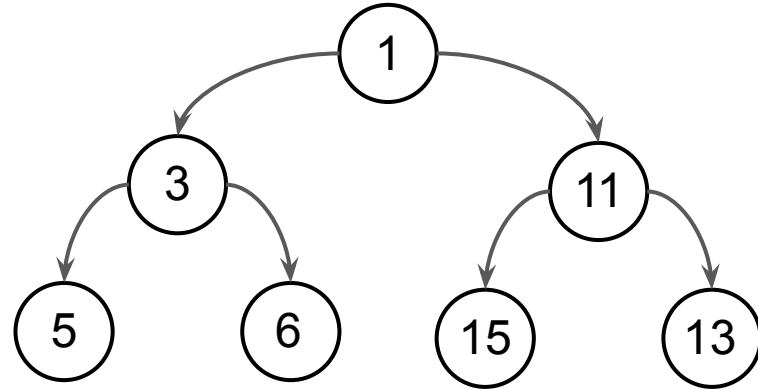
Дерево – связный
ациклический граф



Бинарные деревья



Бинарное дерево — связный ациклический граф, у которого $\text{degree}(v) \leq 2$ для любого узла v



Бинарные деревья (альтернативное определение)

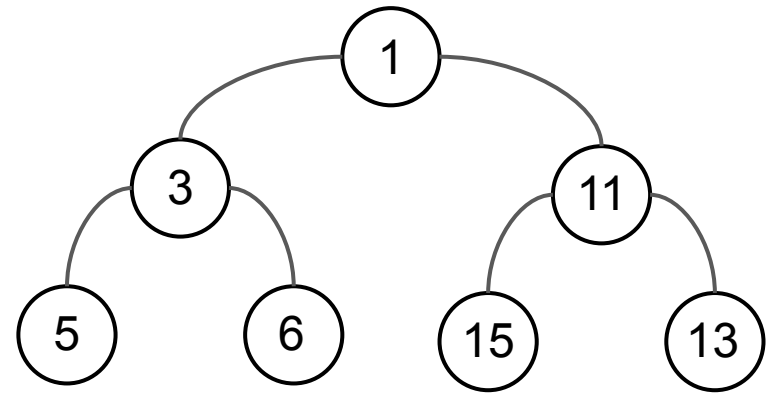
Бинарное дерево – $\begin{cases} \emptyset \\ \{\text{value}, \text{left}, \text{right}\} \end{cases}$

где **value** – значение в вершине, а **left** и **right** также являются бинарными деревьями

Бинарные деревья (альтернативное определение)

Бинарное дерево – $\begin{cases} \emptyset \\ \{\text{value}, \text{left}, \text{right}\} \end{cases}$

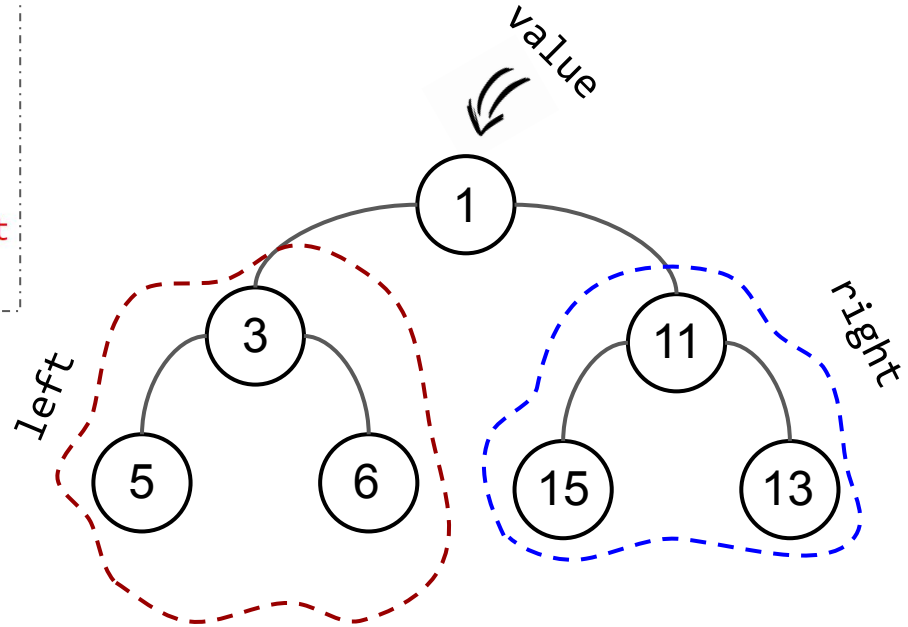
где **value** – значение в вершине, а **left** и **right** также являются бинарными деревьями



Бинарные деревья (альтернативное определение)

Бинарное дерево – $\begin{cases} \emptyset \\ \{\text{value}, \text{left}, \text{right}\} \end{cases}$

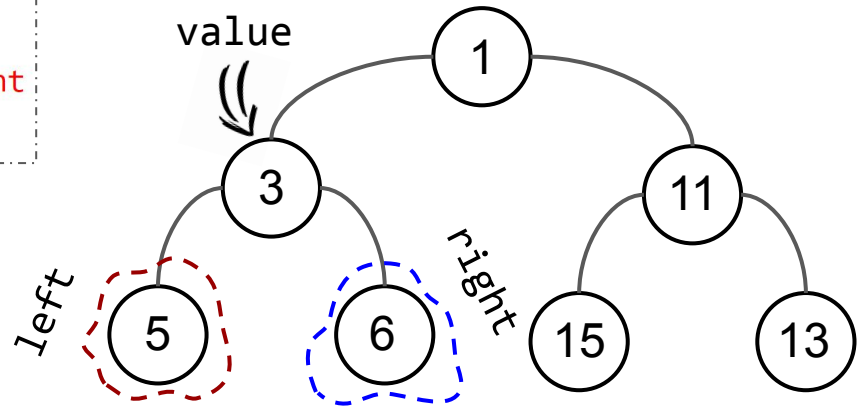
где **value** – значение в вершине, а **left** и **right** также являются бинарными деревьями



Бинарные деревья (альтернативное определение)

Бинарное дерево – $\begin{cases} \emptyset \\ \{\text{value}, \text{left}, \text{right}\} \end{cases}$

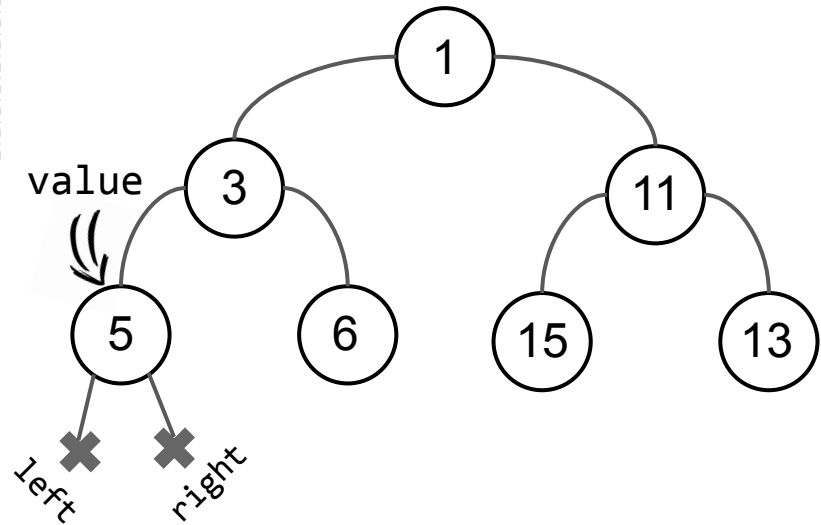
где **value** – значение в вершине, а **left** и **right** также являются бинарными деревьями



Бинарные деревья (альтернативное определение)

Бинарное дерево – $\begin{cases} \emptyset \\ \{\text{value}, \text{left}, \text{right}\} \end{cases}$

где **value** – значение в вершине, а **left** и **right** также являются бинарными деревьями

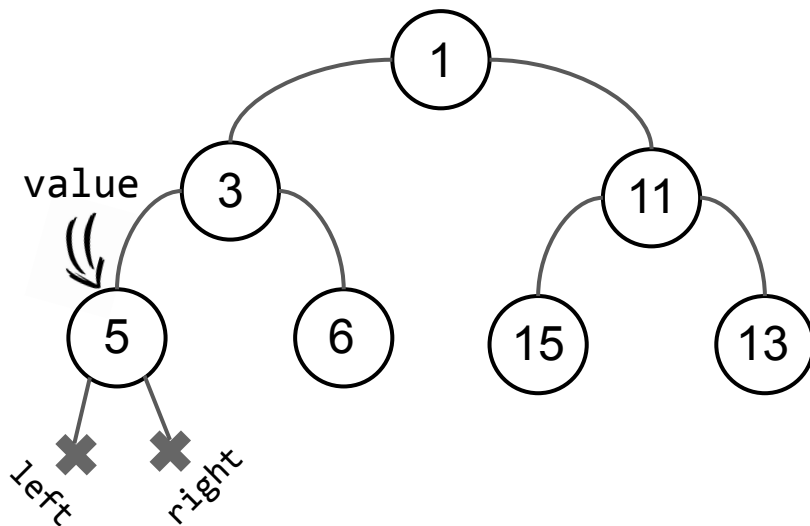


Бинарные деревья (альтернативное определение)

Бинарное дерево – $\begin{cases} \emptyset \\ \{\text{value}, \text{left}, \text{right}\} \end{cases}$

где **value** – значение в вершине, а **left** и **right** также являются бинарными деревьями

Рекурсивность определения сразу показывает, как решать многие задачи на деревья



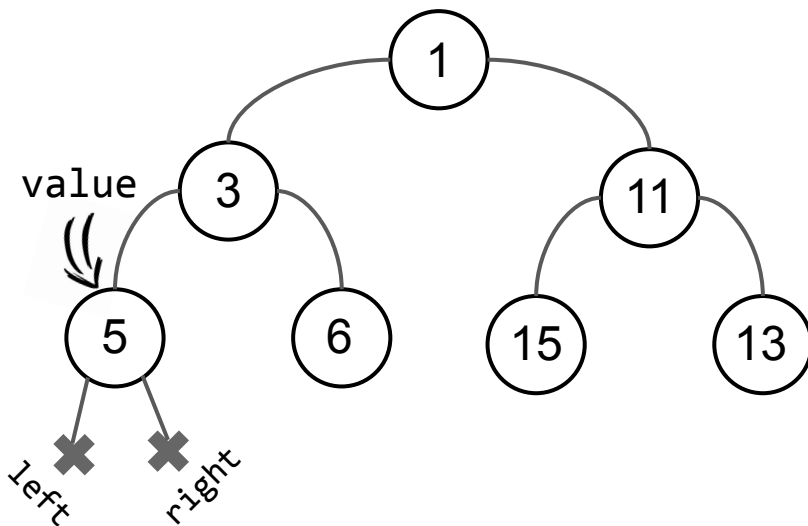
Бинарные деревья (альтернативное определение)

Бинарное дерево – $\begin{cases} \emptyset \\ \{\text{value}, \text{left}, \text{right}\} \end{cases}$

где **value** – значение в вершине, а **left** и **right** также являются бинарными деревьями

Рекурсивность определения сразу показывает, как решать многие задачи на деревья

Через рекурсию!



Бинарные деревья

```
class TreeNode:  
    val: int  
    left: TreeNode  
    right: TreeNode  
  
    def init(self, val: int):  
        self.val = val  
        self.left = None  
        self.right = None
```

Бинарные деревья

Задача: посчитать сумму значений элементов в бинарном дереве

Бинарные деревья

Задача: посчитать сумму значений элементов в бинарном дереве

```
def sum(root: TreeNode):
```

Бинарные деревья

Задача: посчитать сумму значений элементов в бинарном дереве

```
def sum(root: TreeNode):  
    if not root:  
        return 0
```

Бинарные деревья

Задача: посчитать сумму значений элементов в бинарном дереве

```
def sum(root: TreeNode):  
    if not root:  
        return 0  
  
    return root.val +
```

Бинарные деревья

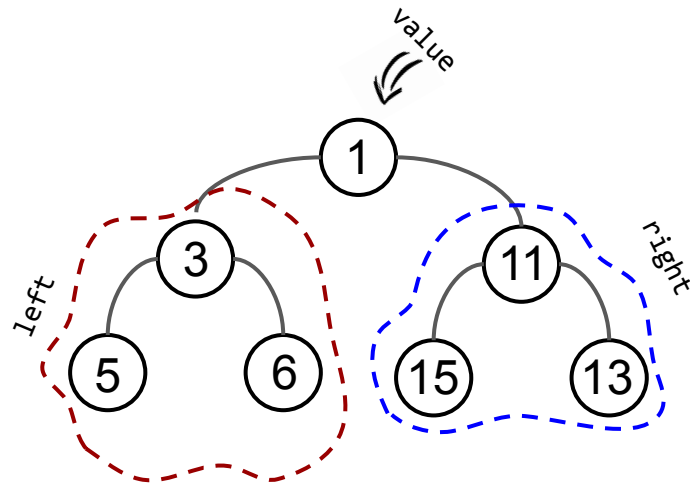
Задача: посчитать сумму значений элементов в бинарном дереве

```
def sum(root: TreeNode):  
    if not root:  
        return 0  
  
    return root.val +  
           sum(root.left) +  
           sum(root.right)
```

Бинарные деревья

Задача: посчитать сумму значений элементов в бинарном дереве

```
def sum(root: TreeNode):  
    if not root:  
        return 0  
  
    return root.val +  
           sum(root.left) +  
           sum(root.right)
```



Бинарные деревья

Задача: сериализовать и десериализовать бинарное дерево.

Бинарные деревья

Задача: сериализовать и десериализовать бинарное дерево.

Сериализовать - представить структуру данных в виде некоторых **сырых** данных на диске или в памяти (обычно в виде строки)

Бинарные деревья

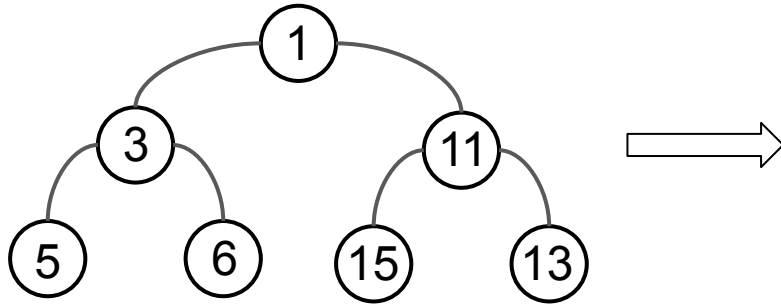
Задача: сериализовать и десериализовать бинарное дерево.

Сериализовать - представить структуру данных в виде некоторых **сырых** данных на диске или в памяти (обычно в виде строки)

Десериализовать - построить структуру данных по сырому её представлению.

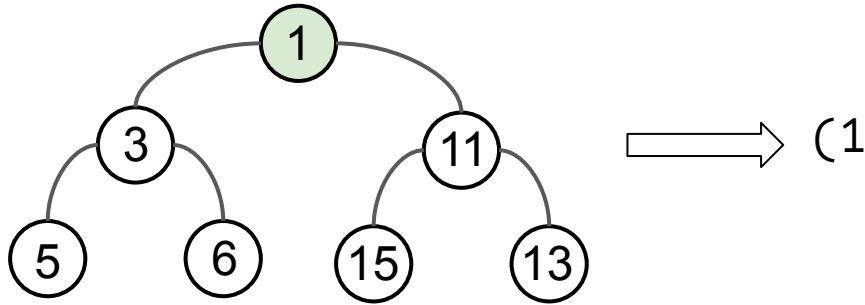
Бинарные деревья

Задача: сериализовать и десериализовать бинарное дерево.



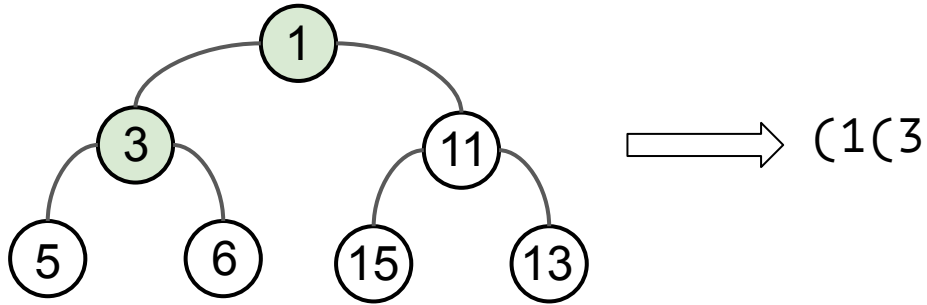
Бинарные деревья

Задача: сериализовать и десериализовать бинарное дерево.



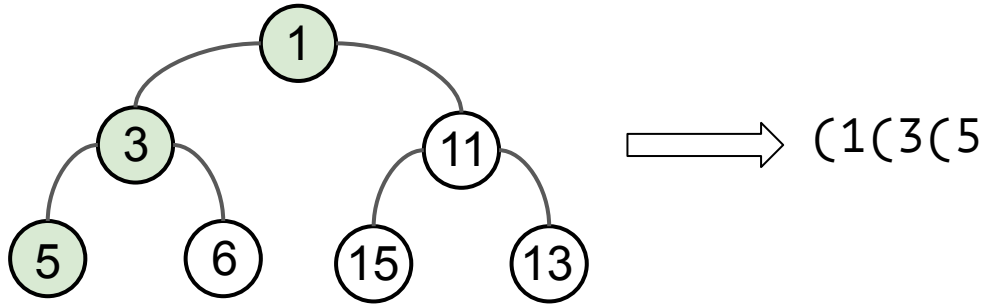
Бинарные деревья

Задача: сериализовать и десериализовать бинарное дерево.



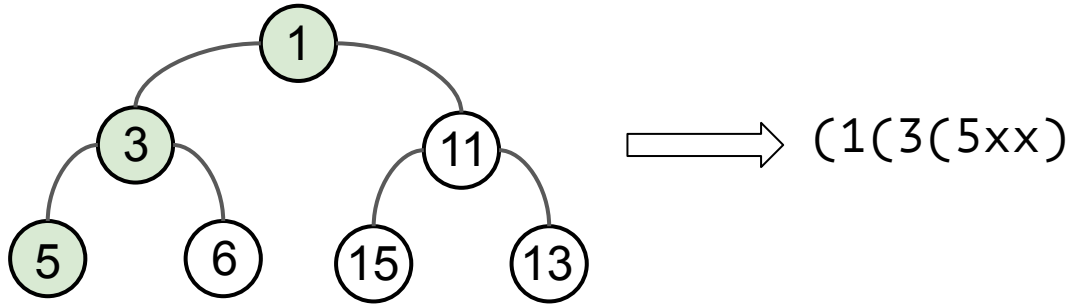
Бинарные деревья

Задача: сериализовать и десериализовать бинарное дерево.



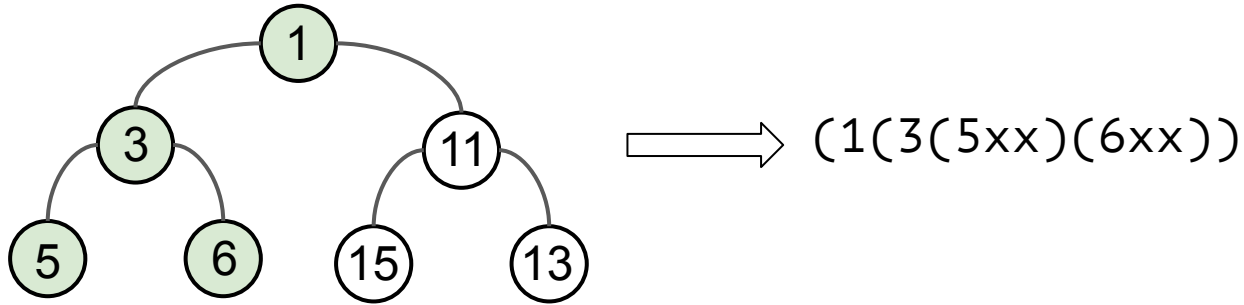
Бинарные деревья

Задача: сериализовать и десериализовать бинарное дерево.



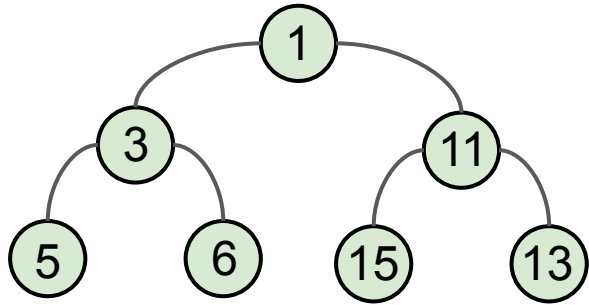
Бинарные деревья

Задача: сериализовать и десериализовать бинарное дерево.



Бинарные деревья

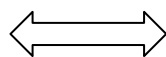
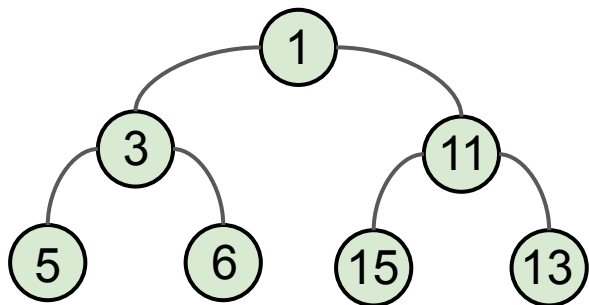
Задача: сериализовать и десериализовать бинарное дерево.



⇒ (1(3(5xx)(6xx))(11(15xx)(13xx)))

Бинарные деревья

Задача: сериализовать и десериализовать бинарное дерево.

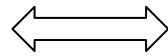
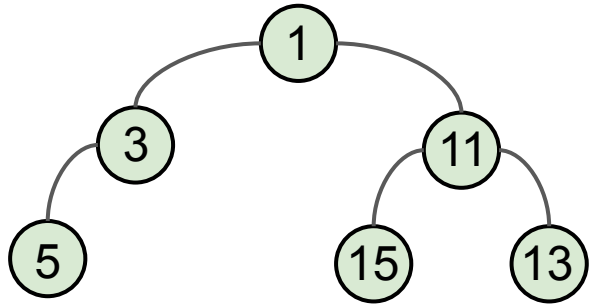


(1(3(5xx)(6xx))(11(15xx)(13xx)))

Скобочная запись дерева, построение - обход в глубину.

Бинарные деревья

Задача: сериализовать и десериализовать бинарное дерево.

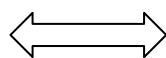
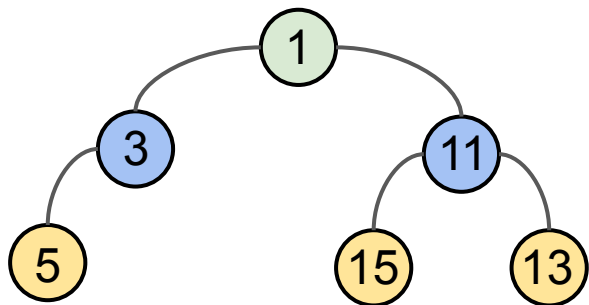


$(1(3(5xx)x)(11(15xx)(13xx)))$

Скобочная запись дерева, построение - обход в глубину.

Бинарные деревья

Задача: сериализовать и десериализовать бинарное дерево.



`(1(3(5xx)x)(11(15xx)(13xx)))`

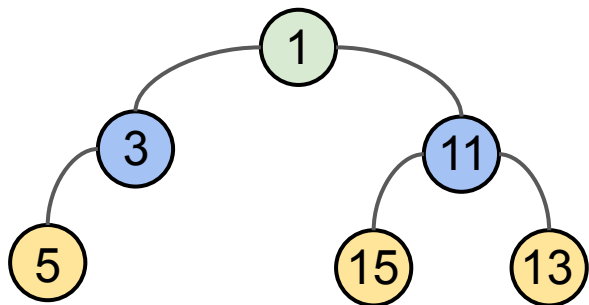
Скобочная запись дерева, построение - обход в глубину.

Альтернатива, принятая на литкоде:

`[1, 3, 11, 5, null, 15, 13]`

Бинарные деревья

Задача: сериализовать и десериализовать бинарное дерево.



\longleftrightarrow (1(3(5xx)x)(11(15xx)(13xx)))

Скобочная запись дерева, построение - обход в глубину.

Альтернатива, принятая на литкоде:

[1, 3, 11, 5, null, 15, 13]

Построение - обход в ширину

Мини-задача #22 (1 балл)

Реализовать сериализацию и десериализацию для бинарных деревьев.

Можно выбрать любую форму, какая вам больше нравится, лишь бы процесс был однозначным.

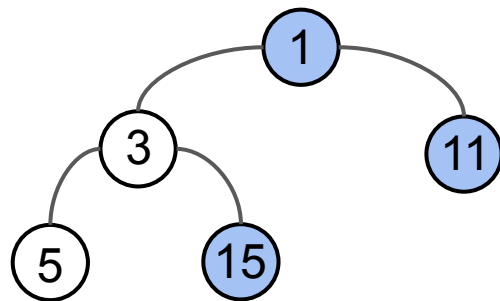
Необходимо проверить свое решение здесь:

<https://leetcode.com/problems/serialize-and-deserialize-binary-tree/>

Мини-задача #23 (1 балл)

Вы стоите справа от бинарного дерева. При этом вы можете видеть только по самой правой вершине в каждом из уровней этого дерева.

Напишите программу, которая напечатает все видимые для вас вершины в порядке от корня к листьям.



<https://leetcode.com/problems/binary-tree-right-side-view>

Бинарные деревья

Задача: найти элемент с заданным значением в бинарном дереве

Бинарные деревья

Задача: найти элемент с заданным значением в бинарном дереве

```
def find(root: TreeNode, val: int) -> TreeNode:  
    if not root: return None
```

Бинарные деревья

Задача: найти элемент с заданным значением в бинарном дереве

```
def find(root: TreeNode, val: int) -> TreeNode:  
    if not root: return None  
  
    if root.val == val: return root
```

Бинарные деревья

Задача: найти элемент с заданным значением в бинарном дереве

```
def find(root: TreeNode, val: int) -> TreeNode:  
    if not root: return None  
  
    if root.val == val: return root  
  
    left = find(root.left, val)  
    if left: return left
```

Бинарные деревья

Задача: найти элемент с заданным значением в бинарном дереве

```
def find(root: TreeNode, val: int) -> TreeNode:  
    if not root: return None  
  
    if root.val == val: return root  
  
    left = find(root.left, val)  
    if left: return left  
  
    return find(root.right, val)
```

Бинарные деревья

Задача: найти элемент с заданным значением в бинарном дереве

```
def find(root: TreeNode, val: int) -> TreeNode:  
    if not root: return None  
  
    if root.val == val: return root  
  
    left = find(root.left, val)  
    if left: return left  
  
    return find(root.right, val)
```

Сложность?

Бинарные деревья

Задача: найти элемент с заданным значением в бинарном дереве

```
def find(root: TreeNode, val: int) -> TreeNode:  
    if not root: return None  
  
    if root.val == val: return root  
  
    left = find(root.left, val)  
    if left: return left  
  
    return find(root.right, val)
```

Сложность?

$O(N)$,

что очень
слабо

Бинарные деревья (альтернативное определение)

Бинарное дерево – $\begin{cases} \emptyset \\ \{\text{value}, \text{left}, \text{right}\} \end{cases}$

где **value** – значение в вершине, а **left** и **right** также являются бинарными деревьями

Бинарные деревья поиска

Бинарное дерево **поиска** $\left\{ \begin{array}{l} \emptyset \\ \{value, left, right\} \end{array} \right.$

где **value** — значение в вершине, **left** и **right** также являются бинарными деревьями **поиска**, и каждое значение из **left** меньше **value**, а каждое значение из **right** больше **value**.

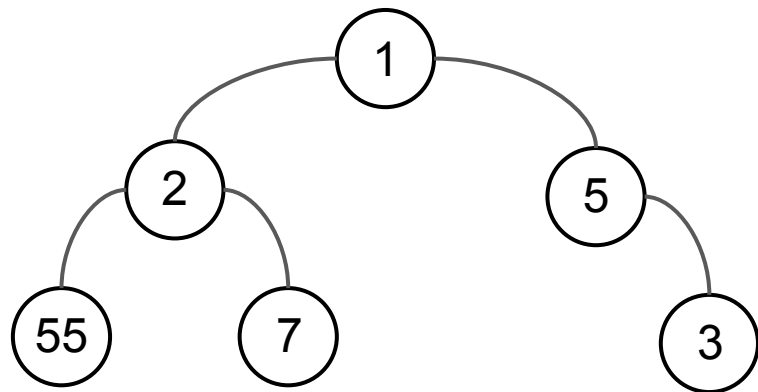
Бинарные деревья поиска

Бинарное дерево **поиска**
BST

$\left\{ \begin{array}{l} \emptyset \\ \{value, left, right\} \end{array} \right.$

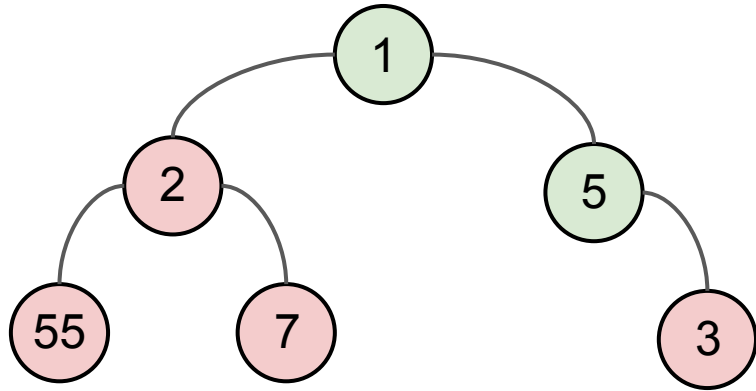
где **value** — значение в вершине, **left** и **right** также являются бинарными деревьями **поиска**, и каждое значение из **left** меньше value, а каждое значение из **right** больше value.

Бинарные деревья поиска



Бинарное дерево поиска?

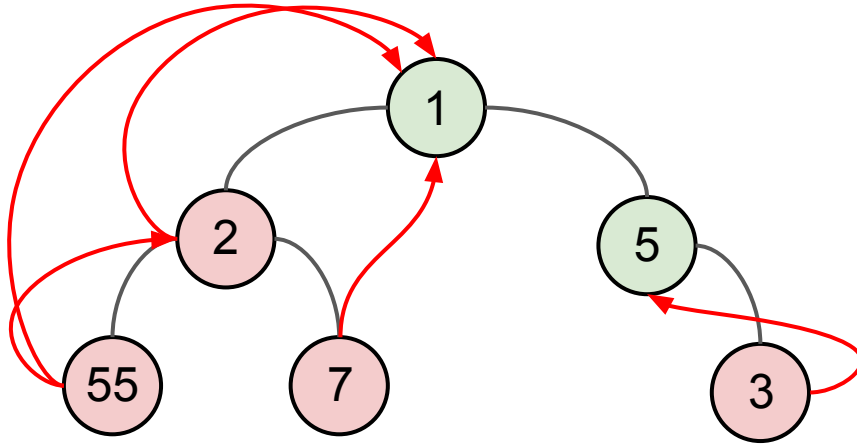
Бинарные деревья поиска



Бинарное дерево поиска?

Нет!

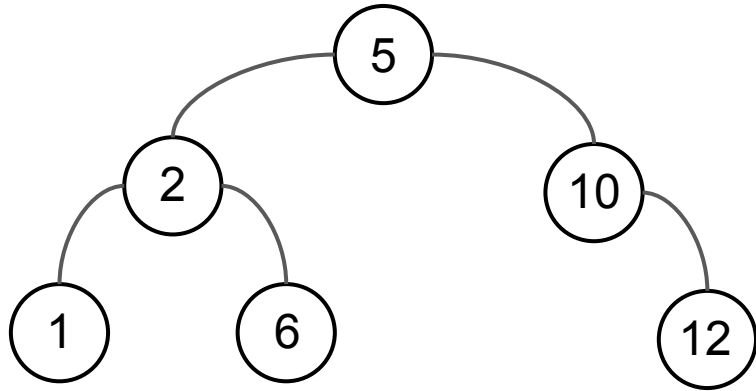
Бинарные деревья поиска



Бинарное дерево поиска?

Нет!

Бинарные деревья поиска

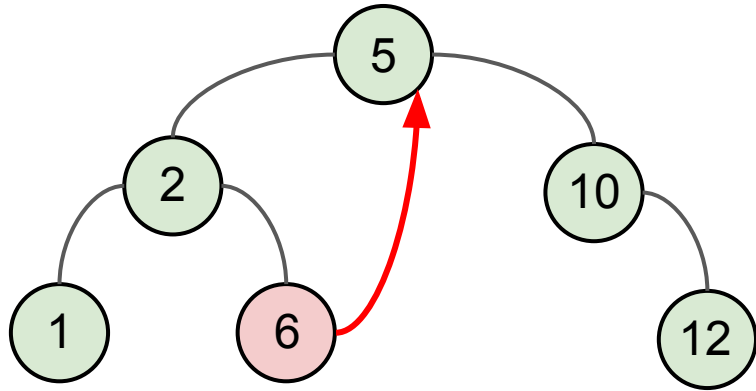


Бинарное дерево поиска?

Нет!

А теперь?

Бинарные деревья поиска



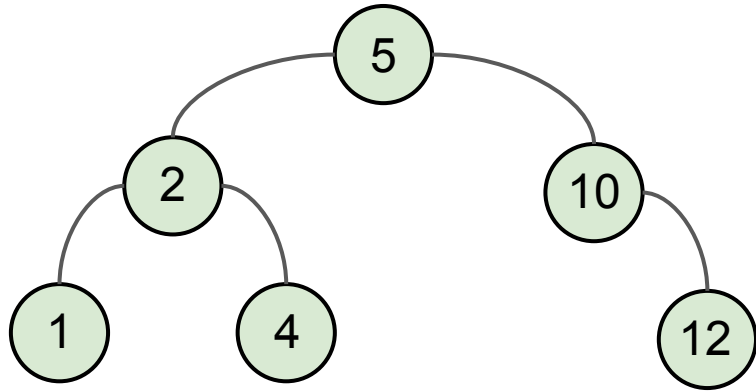
Бинарное дерево поиска?

Нет!

А теперь?

Все еще нет, **все** слева от вершины должны быть меньше.

Бинарные деревья поиска



Бинарное дерево поиска?

Нет!

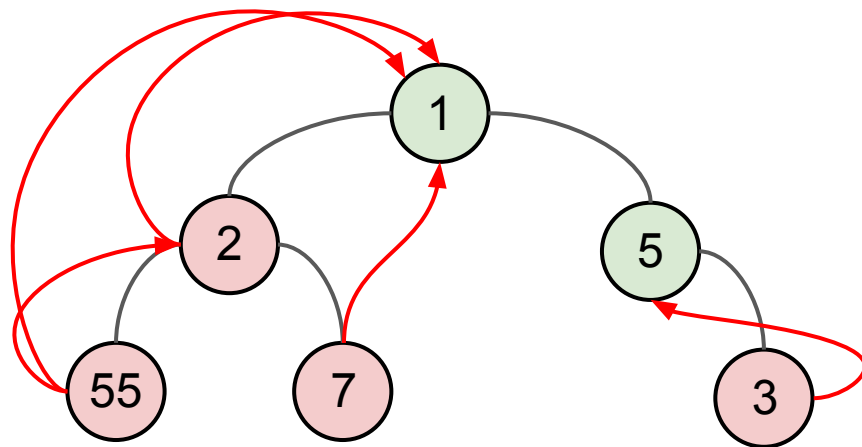
А теперь?

Все еще нет, **все** слева от вершины должны быть меньше.

Вот теперь – да.

Мини-задача #24 (1 балл)

Проверить, является ли данное вам бинарное дерево бинарным деревом **поиска**.

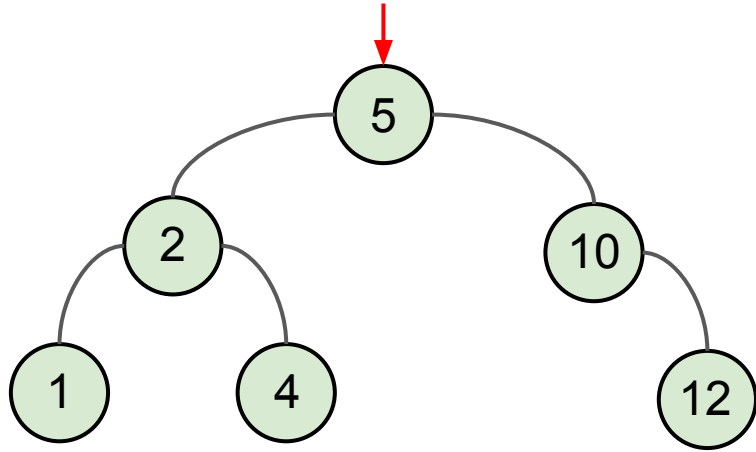


Бинарные деревья поиска

Задача: найти элемент с заданным значением в бинарном дереве **поиска**.

Бинарные деревья поиска

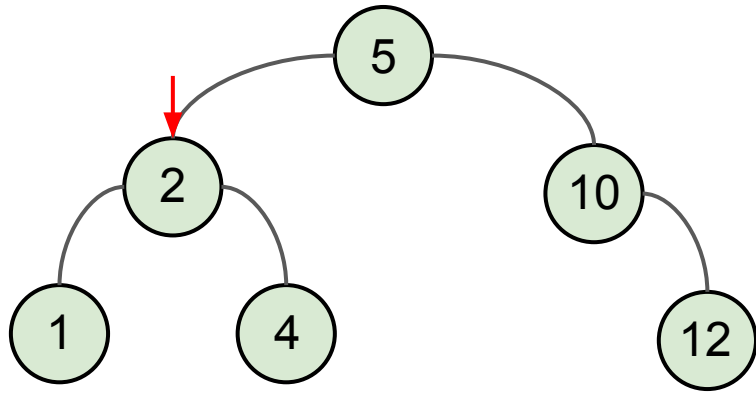
Задача: найти элемент с заданным значением в бинарном дереве **поиска**.



find(4)?

Бинарные деревья поиска

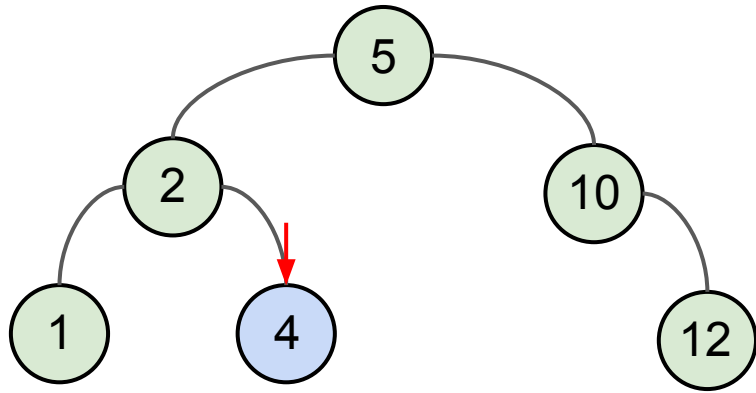
Задача: найти элемент с заданным значением в бинарном дереве **поиска**.



find(4)?

Бинарные деревья поиска

Задача: найти элемент с заданным значением в бинарном дереве **поиска**.



find(4)?

Бинарные деревья поиска

Задача: найти элемент с заданным значением в **BST**.

```
def find(root: TreeNode, val: int) -> TreeNode:  
    if not root: return None  
  
    if root.val == val: return root
```

Бинарные деревья поиска

Задача: найти элемент с заданным значением в **BST**.

```
def find(root: TreeNode, val: int) -> TreeNode:  
    if not root: return None  
  
    if root.val == val: return root  
  
    if root.val > val:  
        return find(root.left, val)
```

Бинарные деревья поиска

Задача: найти элемент с заданным значением в **BST**.

```
def find(root: TreeNode, val: int) -> TreeNode:  
    if not root: return None  
  
    if root.val == val: return root  
  
    if root.val > val:  
        return find(root.left, val)  
  
    return find(root.right, val)
```


Бинарные деревья поиска

Задача: найти элемент с заданным значением в бинарном дереве.

```
def find(root: TreeNode, val: int) -> TreeNode:  
    if not root: return None  
  
    if root.val == val: return root  
  
    left = find(root.left, val)  
    if left: return left  
  
    return find(root.right, val)
```

Бинарные деревья поиска

Задача: найти элемент с заданным значением в **BST**.

```
def find(root: TreeNode, val: int) -> TreeNode:  
    if not root: return None  
  
    if root.val == val: return root  
  
    if root.val > val:  
        return find(root.left, val)  
  
    return find(root.right, val)
```

Сложность?

Бинарные деревья поиска

Задача: найти элемент с заданным значением в **BST**.

```
def find(root: TreeNode, val: int) -> TreeNode:  
    if not root: return None  
  
    if root.val == val: return root  
  
    if root.val > val:  
        return find(root.left, val)  
  
    return find(root.right, val)
```

Сложность?

Хочется сказать
 $O(\log N)$

Бинарные деревья поиска

Задача: найти элемент с заданным значением в **BST**.

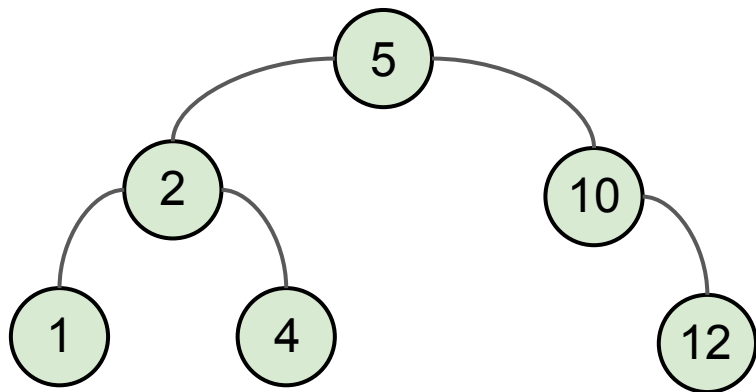
```
def find(root: TreeNode, val: int) -> TreeNode:  
    if not root: return None  
  
    if root.val == val: return root  
  
    if root.val > val:  
        return find(root.left, val)  
  
    return find(root.right, val)
```

Сложность?

Хочется сказать
 $O(\log N)$

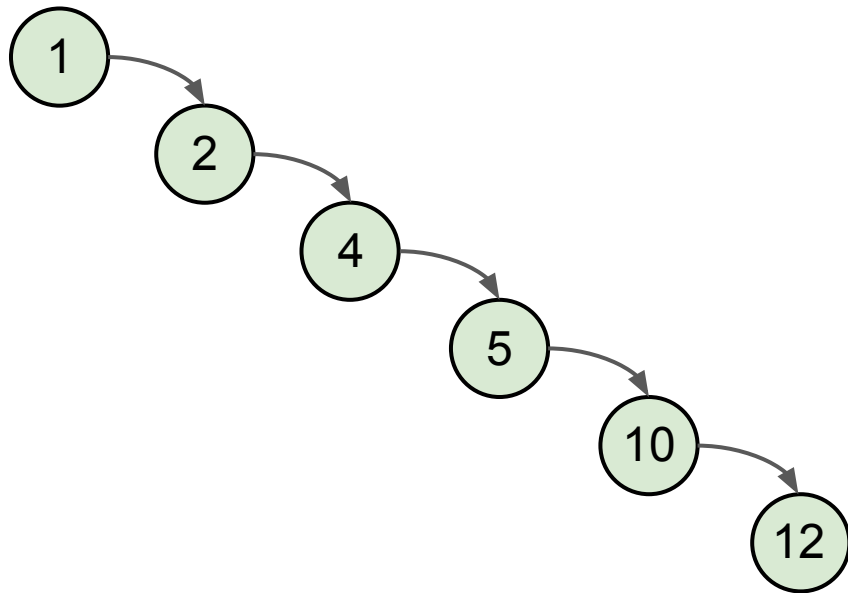
Но какие бывают
BST?

Бинарные деревья поиска



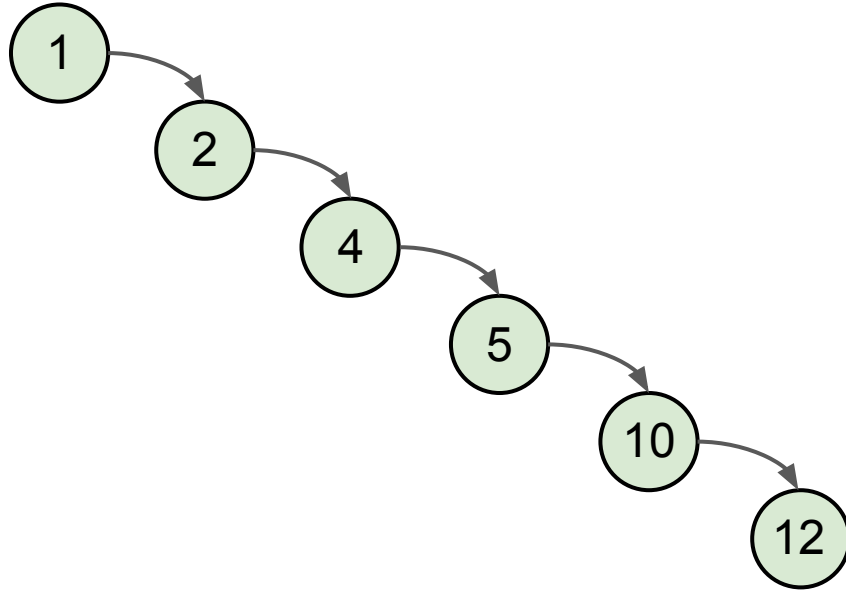
Это хорошее BST

Бинарные деревья поиска



Это всё ещё BST

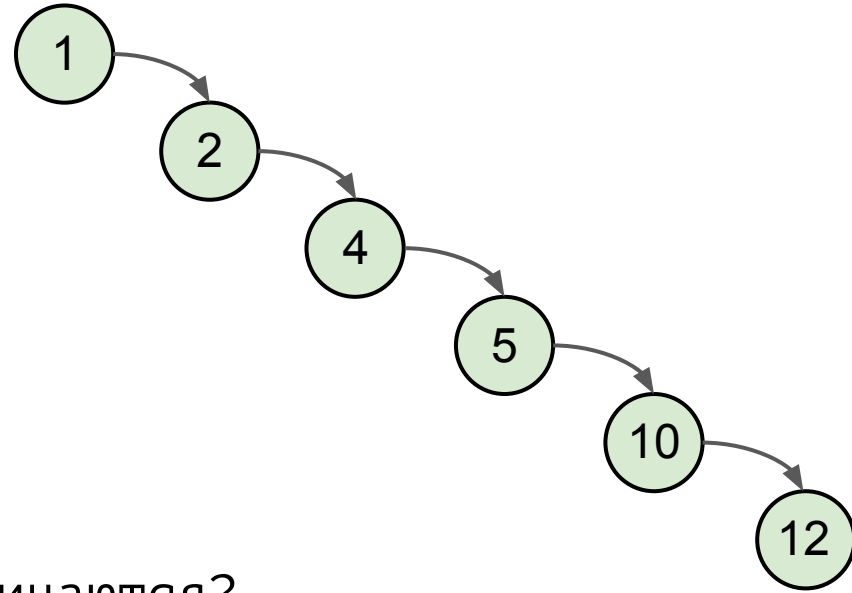
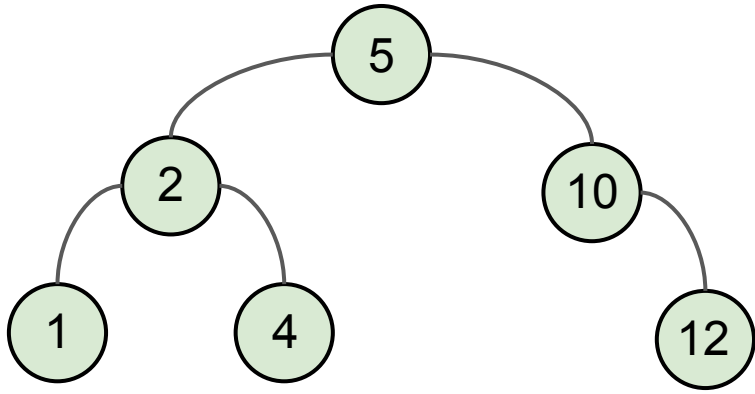
Бинарные деревья поиска



Это всё ещё BST

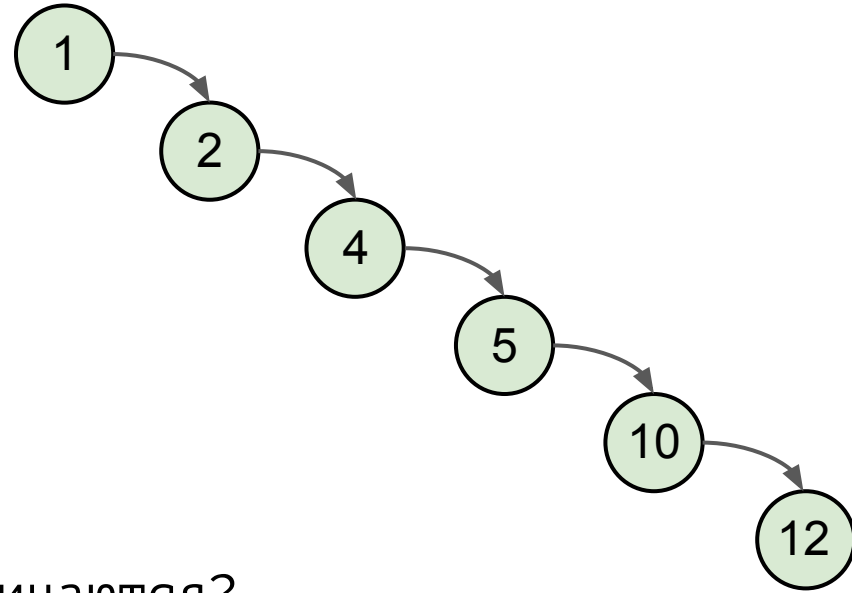
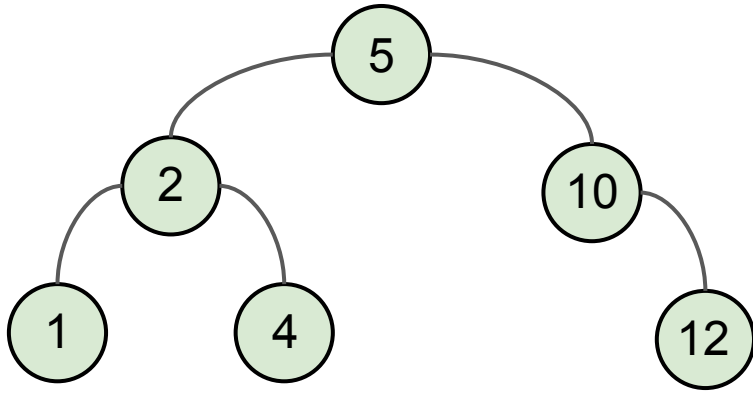
Но уже совсем
не хорошее

Бинарные деревья поиска



Чем эти деревья поиска отличаются?

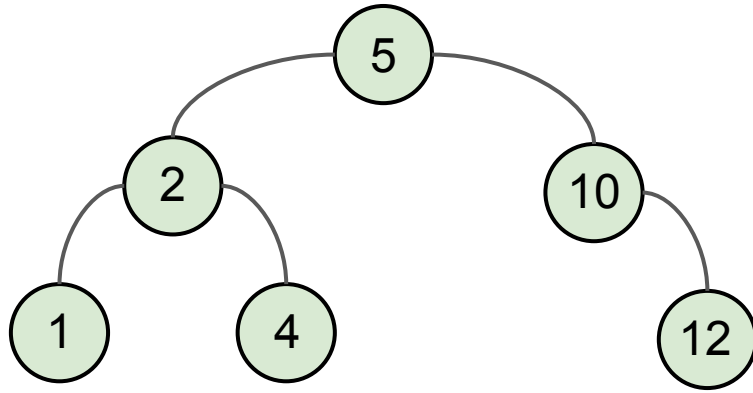
Бинарные деревья поиска



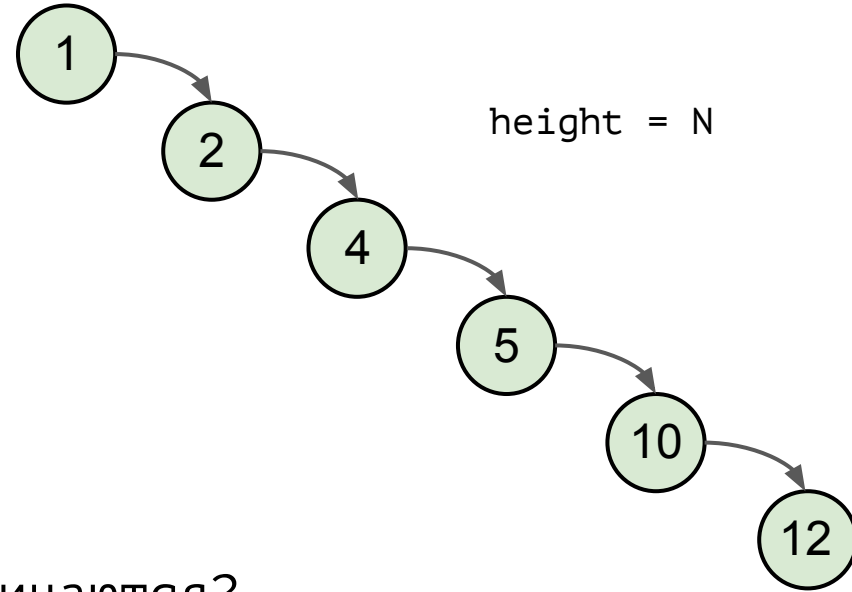
Чем эти деревья поиска отличаются?

Высотой! Второе намного выше.

Бинарные деревья поиска



height = $\log N$



height = N

Чем эти деревья поиска отличаются?

Высотой! Второе намного выше.

Бинарные деревья поиска

Задача: найти элемент с заданным значением в **BST**.

```
def find(root: TreeNode, val: int) -> TreeNode:  
    if not root: return None
```

Сложность?

```
    if root.val == val: return root
```

Хочется сказать
 $O(\log N)$

```
    if root.val > val:  
        return find(root.left, val)
```

Но какие бывают
BST?

```
    return find(root.right, val)
```

Бинарные деревья поиска

Задача: найти элемент с заданным значением в **BST**.

```
def find(root: TreeNode, val: int) -> TreeNode:
    if not root: return None
    if root.val == val: return root
    if root.val > val:
        return find(root.left, val)
    return find(root.right, val)
```

Сложность?
 $O(\text{height})$

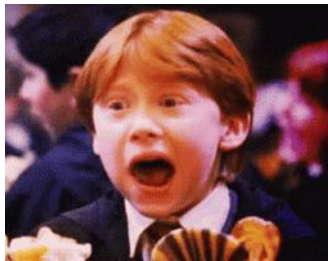
Бинарные деревья поиска

Задача: найти элемент с заданным значением в **BST**.

```
def find(root: TreeNode, val: int) -> TreeNode:  
    if not root: return None  
  
    if root.val == val: return root  
  
    if root.val > val:  
        return find(root.left, val)  
  
    return find(root.right, val)
```

Сложность?

$O(\text{height})$ - т.е.
в худшем случае
опять $O(N)$



Бинарные деревья поиска

Операции:

1. `find(value)` $\rightarrow O(???)$
2. `select(i)` $\rightarrow O(???)$
3. `min/max` $\rightarrow O(???)$
4. `pred/succ(ptr)` $\rightarrow O(???)$
5. `rank(value)` $\rightarrow O(???)$
6. вывод в пор. $\rightarrow O(???)$
 возрастания

-
7. `insert(value)` $\rightarrow O(???)$
 8. `remove(value)` $\rightarrow O(???)$

Бинарные деревья поиска

Операции:

1. `find(value)` \rightarrow `O(height)`
2. `select(i)` \rightarrow `O(???)`
3. `min/max` \rightarrow `O(???)`
4. `pred/succ(ptr)` \rightarrow `O(???)`
5. `rank(value)` \rightarrow `O(???)`
6. вывод в пор. \rightarrow `O(???)`
 возрастания

-
7. `insert(value)` \rightarrow `O(???)`
 8. `remove(value)` \rightarrow `O(???)`

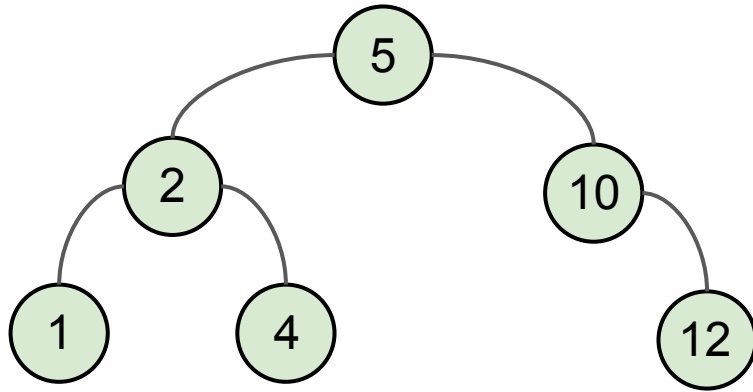
Бинарные деревья поиска

Операции:

1. `find(value)` \rightarrow `O(height)`
2. `select(i)` \rightarrow `O(???)`
3. `min/max` \rightarrow `O(???)`
4. `pred/succ(ptr)` \rightarrow `O(???)`
5. `rank(value)` \rightarrow `O(???)`
6. вывод в пор. \rightarrow `O(???)`
 возрастания

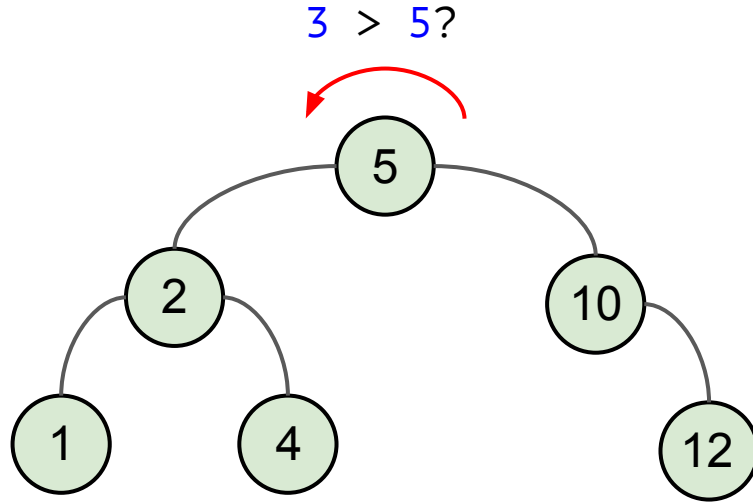
-
7. `insert(value)` \rightarrow `O(???)`
 8. `remove(value)` \rightarrow `O(???)`

Бинарные деревья поиска: вставка



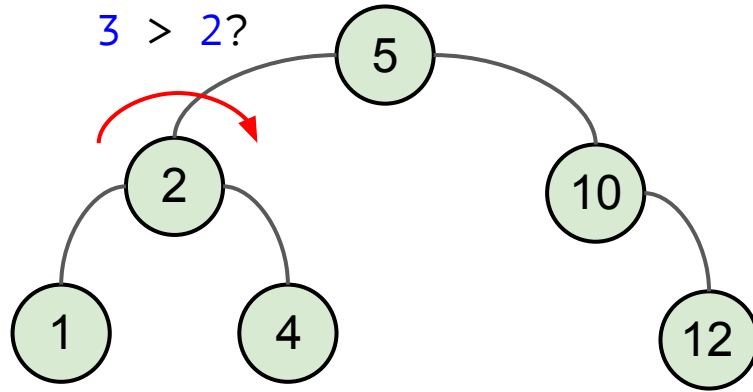
insert(3)

Бинарные деревья поиска: вставка



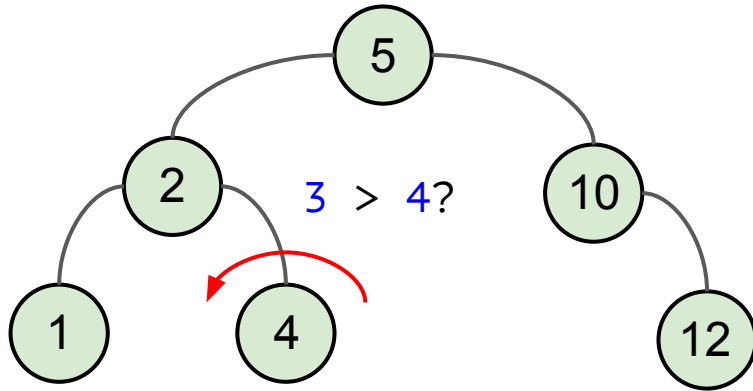
insert(3)

Бинарные деревья поиска: вставка



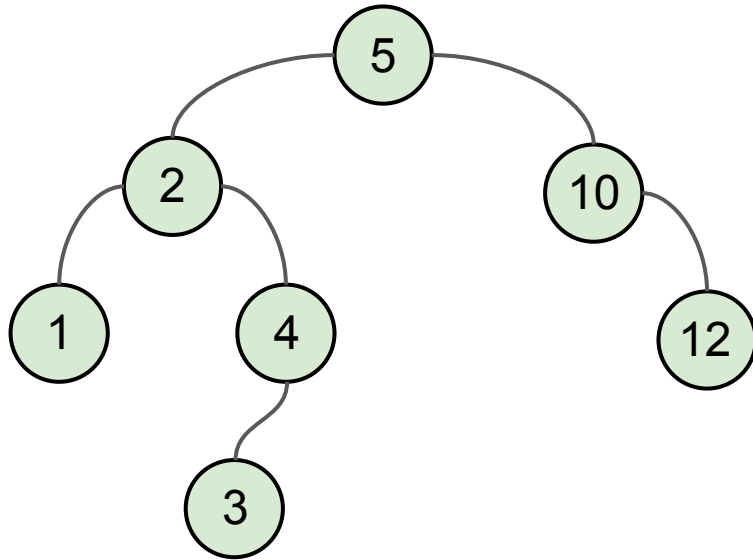
insert(3)

Бинарные деревья поиска: вставка



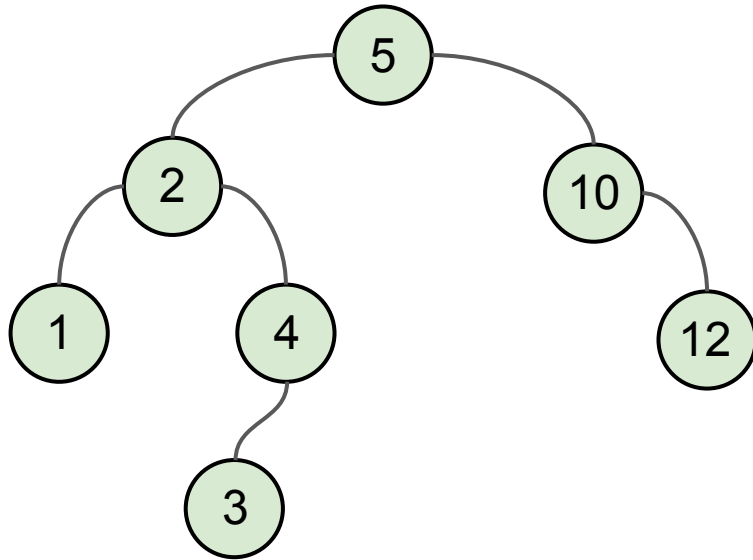
insert(3)

Бинарные деревья поиска: вставка



insert(3)

Бинарные деревья поиска: вставка



`insert(3)`

Проход до места вставки
за $O(\text{height})$, сама
вставка за константу.

Бинарные деревья поиска

Операции:

1. `find(value)` -> 0(`height`)
2. `select(i)` -> 0(???)
3. `min/max` -> 0(???)
4. `pred/succ(ptr)` -> 0(???)
5. `rank(value)` -> 0(???)
6. вывод в пор. -> 0(???)
возрастания

-
7. `insert(value)` -> 0(`height`)
 8. `remove(value)` -> 0(???)

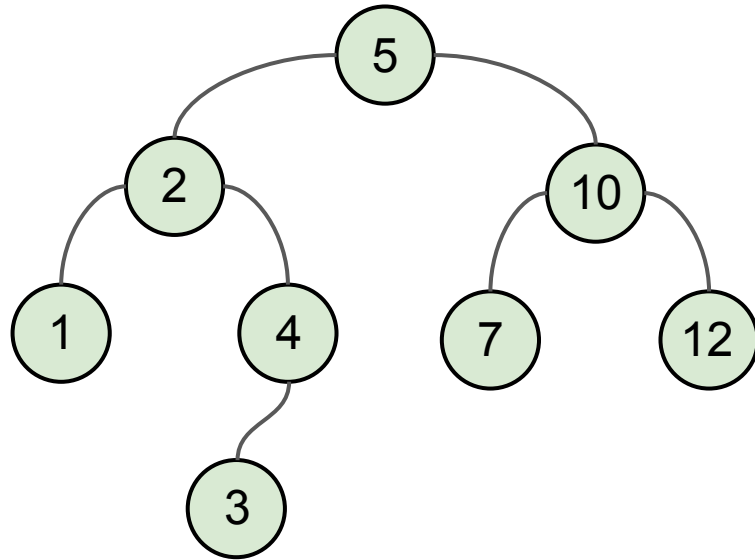
Бинарные деревья поиска

Операции:

1. `find(value)` -> 0(**height**)
 2. `select(i)` -> 0(???)
 3. `min/max` -> 0(???)
 4. `pred/succ(ptr)` -> 0(???)
 5. `rank(value)` -> 0(???)
 6. вывод в пор. -> 0(???)
возрастания
-

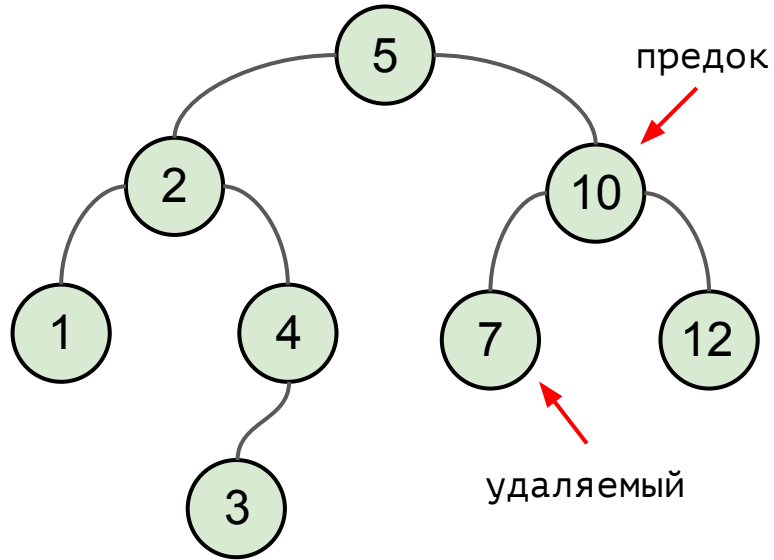
7. `insert(value)` -> 0(**height**)
8. **remove**(value) -> 0(???)

Бинарные деревья поиска: удаление



remove(7)

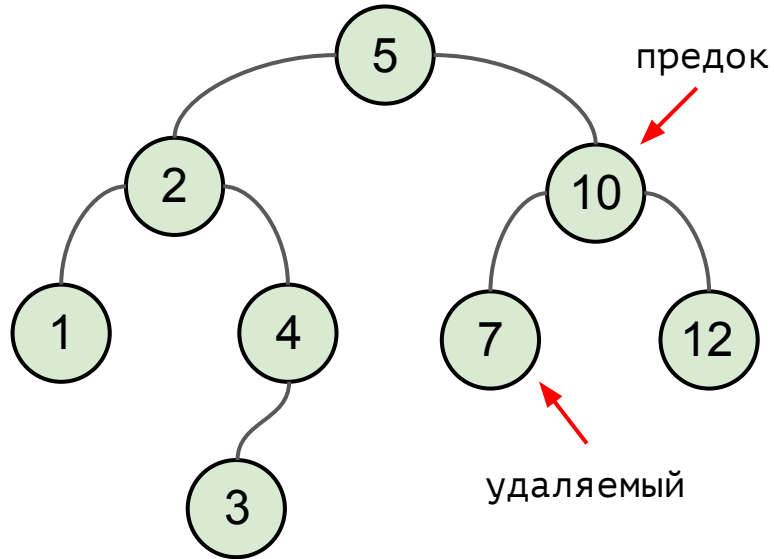
Бинарные деревья поиска: удаление



`remove(7)`

все время нужно будет
смотреть на **предка**,
иногда удобно его
хранить в вершине

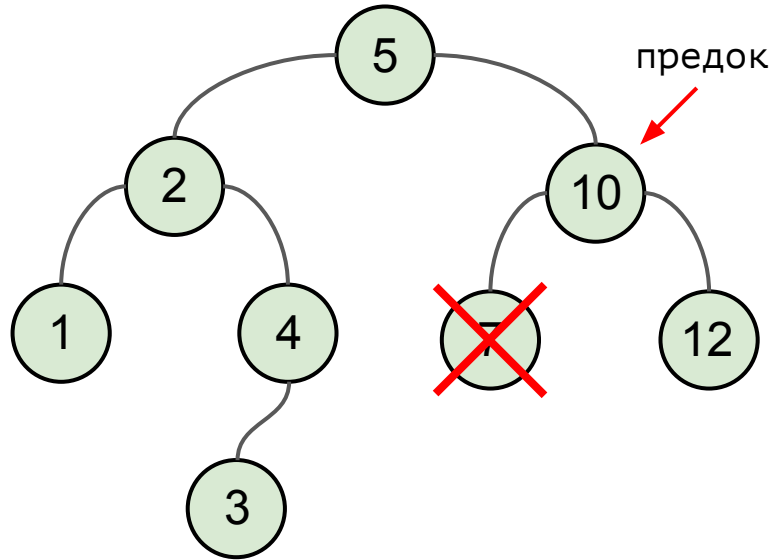
Бинарные деревья поиска: удаление



`remove(7)`

1) случай - удаляем лист

Бинарные деревья поиска: удаление

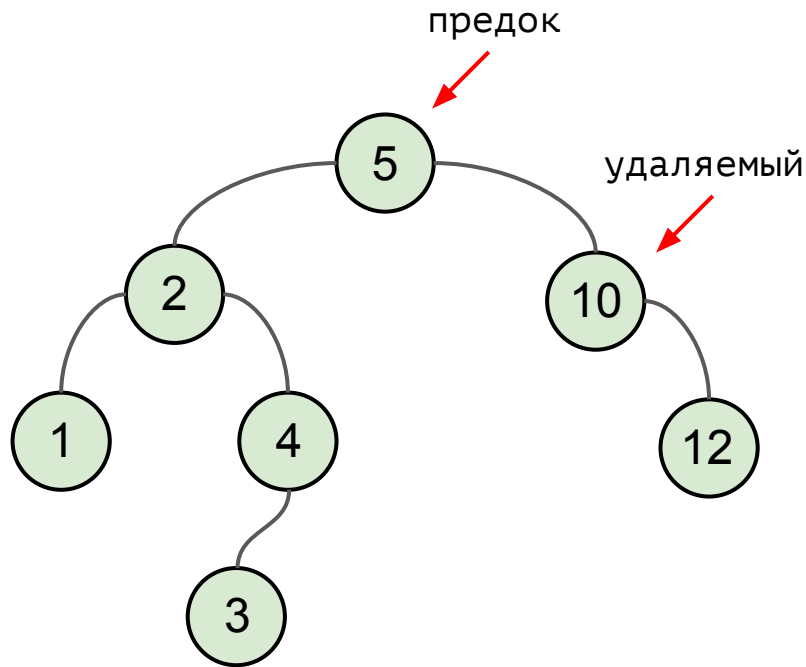


`remove(7)`

1) случай - удаляем лист

просто зануляем предку
поле (left или right)

Бинарные деревья поиска: удаление

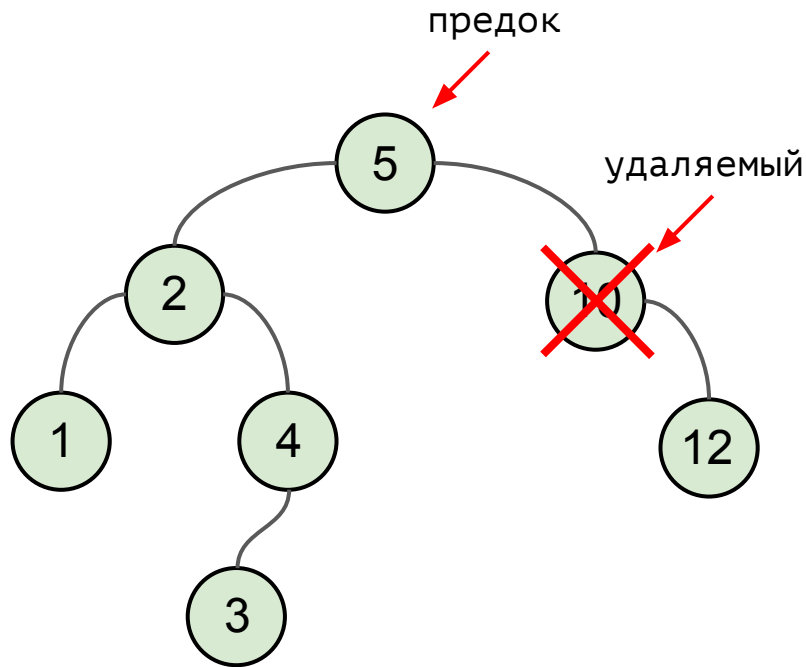


remove(10)

1) случай - удаляем лист

2) удаляем вершину с одним наследником

Бинарные деревья поиска: удаление

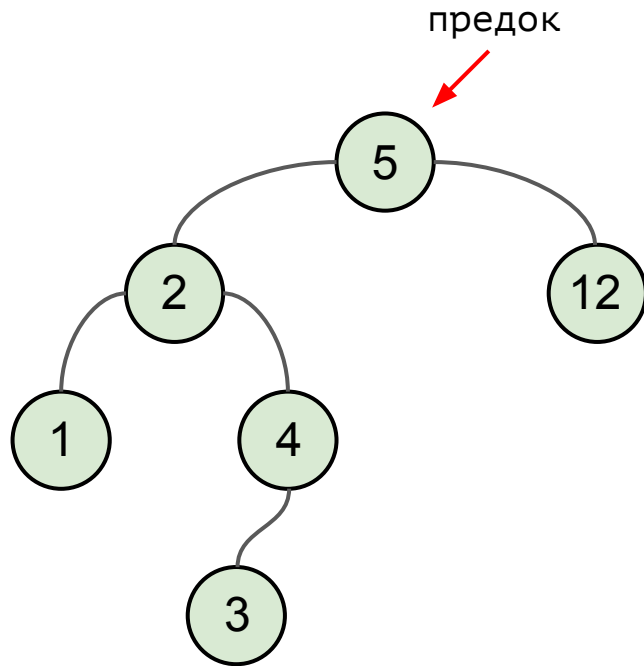


remove(10)

1) случай - удаляем лист

2) удаляем вершину с
одним наследником

Бинарные деревья поиска: удаление



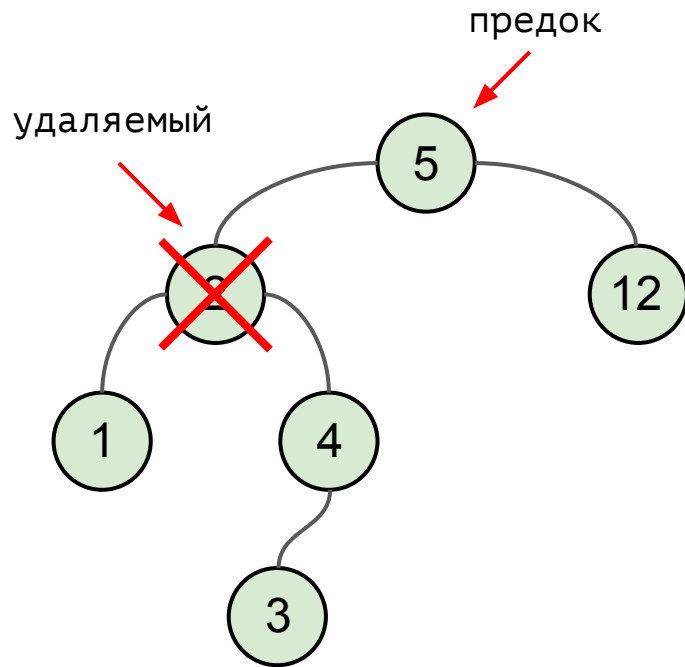
`remove(10)`

1) случай - удаляем лист

2) удаляем вершину с
одним наследником

подтягиваем наследника к
деду

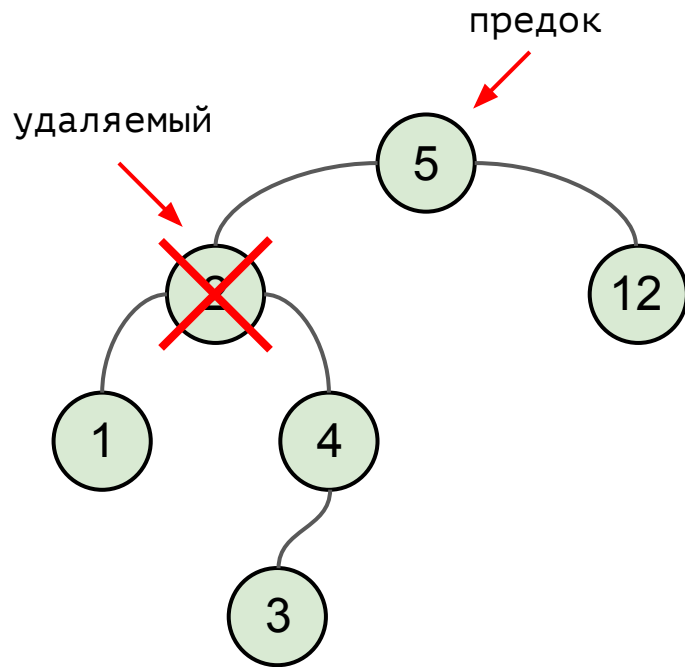
Бинарные деревья поиска: удаление



remove(2)

- 1) случай - удаляем лист
- 2) удаляем вершину с одним наследником
- 3) удаляем вершину с двумя наследниками

Бинарные деревья поиска: удаление



remove(2)

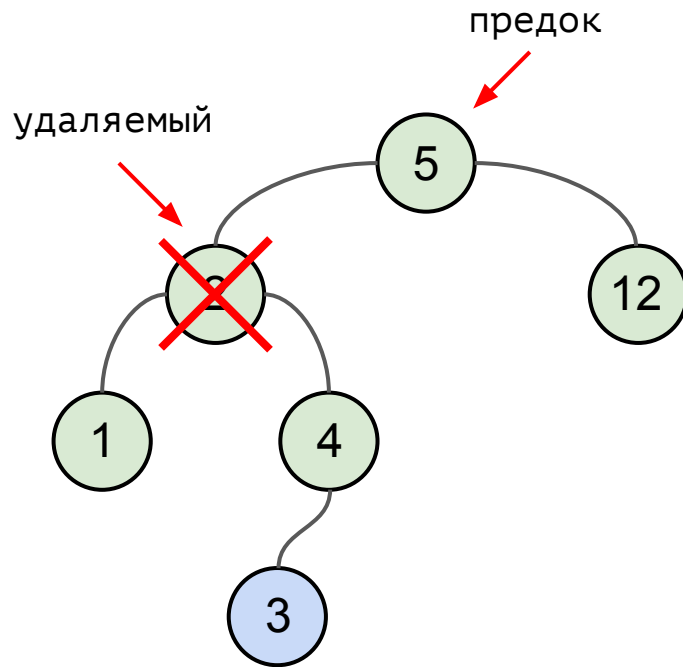
1) случай - удаляем лист

2) удаляем вершину с одним наследником

3) удаляем вершину с двумя наследниками

Кем его заменить?

Бинарные деревья поиска: удаление

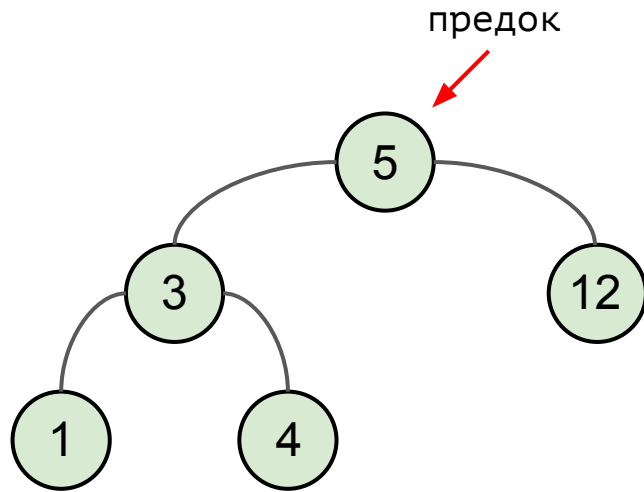


`remove(2)`

- 1) случай - удаляем лист
- 2) удаляем вершину с одним наследником
- 3) удаляем вершину с двумя наследниками

Заменяем самым левым из
правого поддерева

Бинарные деревья поиска: удаление



`remove(2)`

1) случай - удаляем лист

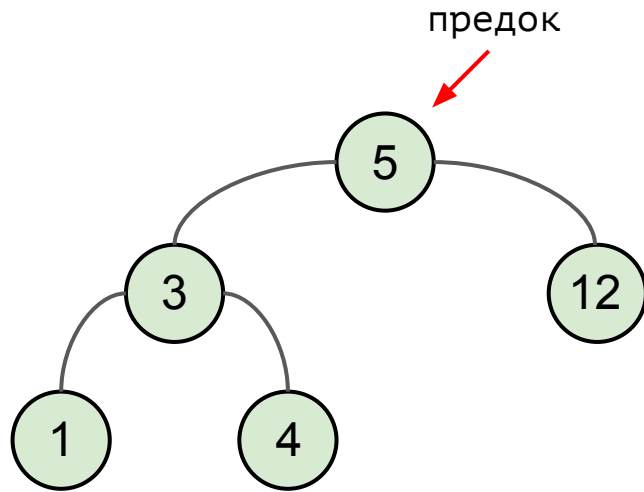
2) удаляем вершину с одним наследником

3) удаляем вершину с двумя наследниками

(или самым правым из левого)

Заменяем самым левым из
правого поддерева

Бинарные деревья поиска: удаление



Сложность?

`remove(2)`

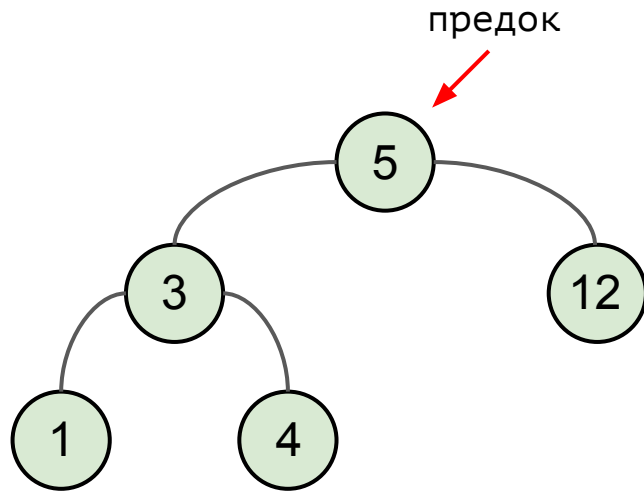
1) случай - удаляем лист

2) удаляем вершину с одним наследником

3) удаляем вершину с двумя наследниками

Заменяем самым левым из
правого поддерева

Бинарные деревья поиска: удаление



Сложность?

Нужно дойти до элемента, потом еще, возможно до конца. $O(\text{height})$

`remove(2)`

1) случай - удаляем лист

2) удаляем вершину с одним наследником

3) удаляем вершину с двумя наследниками

Заменяем самым левым из правого поддерева

Бинарные деревья поиска

Операции:

1. `find(value)` \rightarrow 0(*height*)
2. `select(i)` \rightarrow 0(???)
3. `min/max` \rightarrow 0(???)
4. `pred/succ(ptr)` \rightarrow 0(???)
5. `rank(value)` \rightarrow 0(???)
6. вывод в пор. \rightarrow 0(???)
 возрастания

-
7. `insert(value)` \rightarrow 0(*height*)
 8. *remove*(value) \rightarrow 0(*height*)

Мини-задача #25 (1 балл)

Реализовать процедуру `trim(root, left, right)`, которая принимает BST и **удаляет** из него все вершины, значения которых не попадают в интервал `[left, right]`.

При этом относительный порядок оставшихся вершин должен совпадать с изначальным, т.е. если какая-то вершина была наследником другой выжившей вершины, то так и должно остаться.

Утверждается, что решение существует единственное.

<https://leetcode.com/problems/trim-a-binary-search-tree/>

Бинарные деревья поиска

Операции:

1. `find(value)` -> `O(height)`
 2. `select(i)` -> `O(???)`
 3. `min/max` -> `O(???)`
 4. `pred/succ(ptr)` -> `O(???)`
 5. `rank(value)` -> `O(???)`
 6. вывод в пор. возрастания -> `O(???)`
-

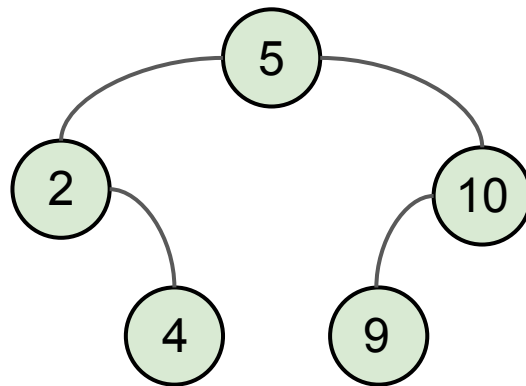
7. `insert(value)` -> `O(height)`
8. `remove(value)` -> `O(height)`

Бинарные деревья поиска

Операции:

1. `find(value)` \rightarrow $O(\text{height})$
 2. `select(i)` \rightarrow $O(???)$
 3. `min/max` \rightarrow $O(???)$
 4. `pred/succ(ptr)` \rightarrow $O(???)$
 5. `rank(value)` \rightarrow $O(???)$
 6. вывод в пор.
возрастания \rightarrow $O(???)$
-

7. `insert(value)` \rightarrow $O(\text{height})$
8. `remove(value)` \rightarrow $O(\text{height})$



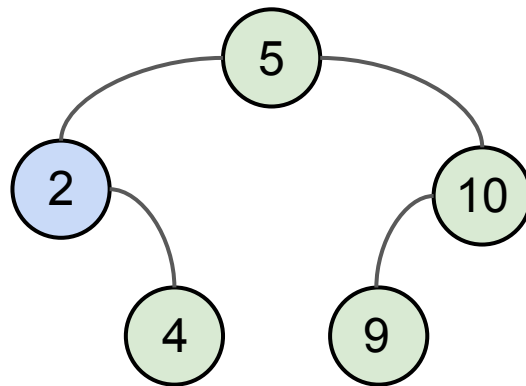
Бинарные деревья поиска

Операции:

1. find(value) -> O(height)
 2. select(i) -> O(???)
 3. min/max -> O(???)
 4. pred/succ(ptr) -> O(???)
 5. rank(value) -> O(???)
 6. вывод в пор.
возрастания -> O(???)
-

7. insert(value) -> O(height)
8. remove(value) -> O(height)

Минимум - встаем в корень, идем влево до упора (пока левого сына не окажется)



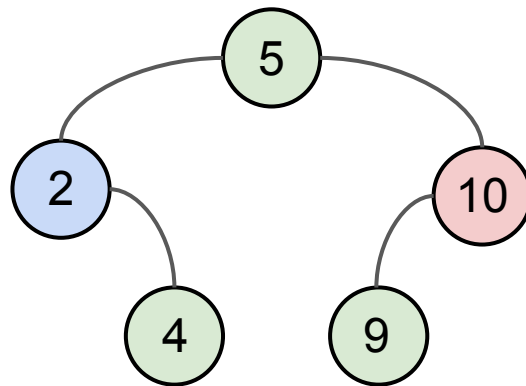
Бинарные деревья поиска

Операции:

1. `find(value)` \rightarrow 0(**height**)
 2. `select(i)` \rightarrow 0(???)
 3. **min/max** \rightarrow 0(???)
 4. `pred/succ(ptr)` \rightarrow 0(???)
 5. `rank(value)` \rightarrow 0(???)
 6. вывод в пор. возрастания \rightarrow 0(???)
-

7. `insert(value)` \rightarrow 0(**height**)
8. `remove(value)` \rightarrow 0(**height**)

Максимум - встаем в корень, идем вправо до упора (пока правого сына не окажется)



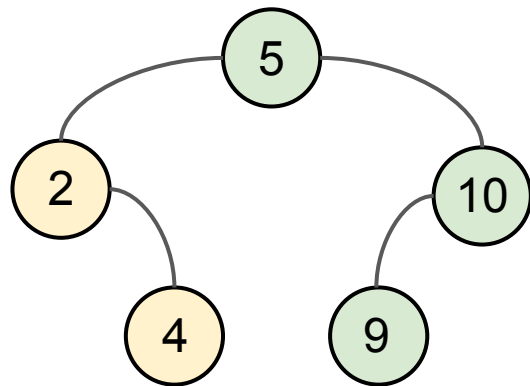
Бинарные деревья поиска

`pred(4) == 2`

Операции:

1. `find(value)` \rightarrow `O(height)`
 2. `select(i)` \rightarrow `O(???)`
 3. `min/max` \rightarrow `O(height)`
 4. `pred/succ(ptr)` \rightarrow `O(???)`
 5. `rank(value)` \rightarrow `O(???)`
 6. вывод в пор. возрастания \rightarrow `O(???)`
-

7. `insert(value)` \rightarrow `O(height)`
8. `remove(value)` \rightarrow `O(height)`



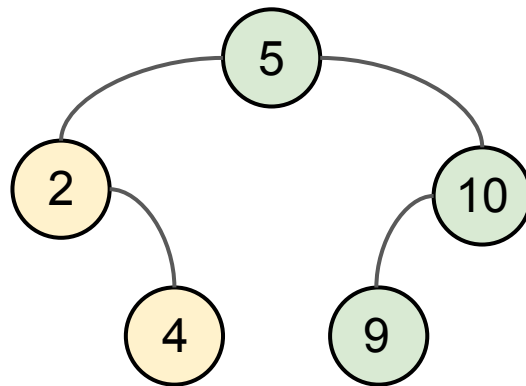
Бинарные деревья поиска

Операции:

1. `find(value)` \rightarrow `O(height)`
 2. `select(i)` \rightarrow `O(???)`
 3. `min/max` \rightarrow `O(height)`
 4. `pred/succ(ptr)` \rightarrow `O(???)`
 5. `rank(value)` \rightarrow `O(???)`
 6. вывод в пор. возрастания \rightarrow `O(???)`
-

7. `insert(value)` \rightarrow `O(height)`
8. `remove(value)` \rightarrow `O(height)`

`pred(4) == 2`
`pred(9) == ?`



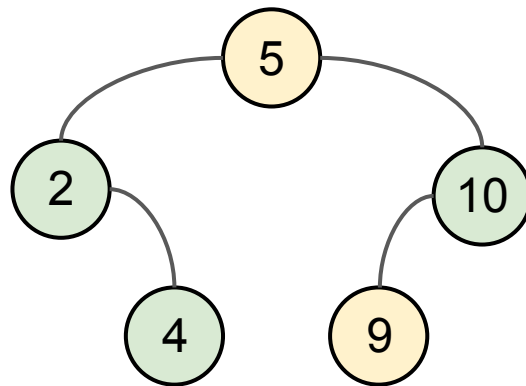
Бинарные деревья поиска

Операции:

1. `find(value)` \rightarrow `O(height)`
 2. `select(i)` \rightarrow `O(???)`
 3. `min/max` \rightarrow `O(height)`
 4. `pred/succ(ptr)` \rightarrow `O(???)`
 5. `rank(value)` \rightarrow `O(???)`
 6. вывод в пор. возрастания \rightarrow `O(???)`
-

7. `insert(value)` \rightarrow `O(height)`
8. `remove(value)` \rightarrow `O(height)`

`pred(4) == 2`
`pred(9) == 5`



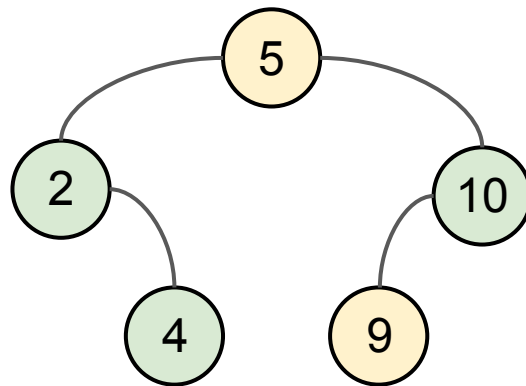
Бинарные деревья поиска

Операции:

1. `find(value)` \rightarrow $O(\text{height})$
 2. `select(i)` \rightarrow $O(???)$
 3. `min/max` \rightarrow $O(\text{height})$
 4. `pred/succ(ptr)` \rightarrow $O(???)$
 5. `rank(value)` \rightarrow $O(???)$
 6. вывод в пор.
возрастания \rightarrow $O(???)$
-

7. `insert(value)` \rightarrow $O(\text{height})$
8. `remove(value)` \rightarrow $O(\text{height})$

Чтобы найти `pred`, идем по предкам, пока не найдем кого-то меньшего, чем мы.

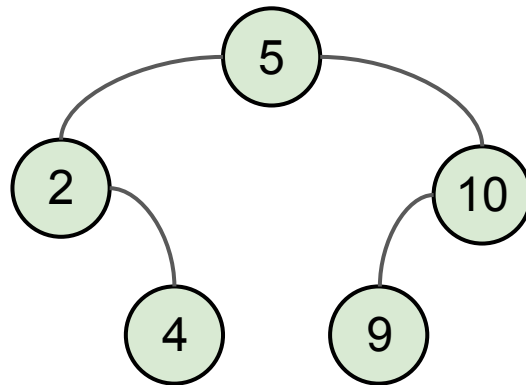


Бинарные деревья поиска

Операции:

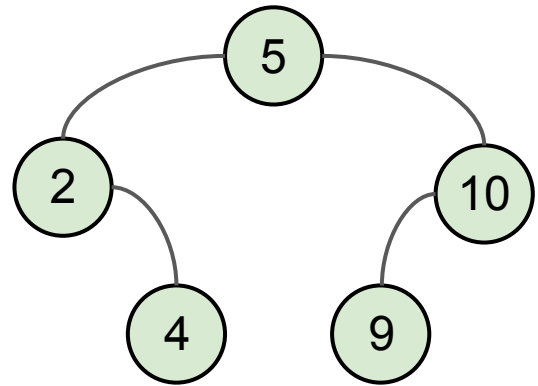
1. `find(value)` $\rightarrow O(\text{height})$
 2. `select(i)` $\rightarrow O(???)$
 3. `min/max` $\rightarrow O(\text{height})$
 4. `pred/succ(ptr)` $\rightarrow O(\text{height})$
 5. `rank(value)` $\rightarrow O(???)$
 6. **вывод в пор.** $\rightarrow O(???)$
возрастания
-

7. `insert(value)` $\rightarrow O(\text{height})$
8. `remove(value)` $\rightarrow O(\text{height})$



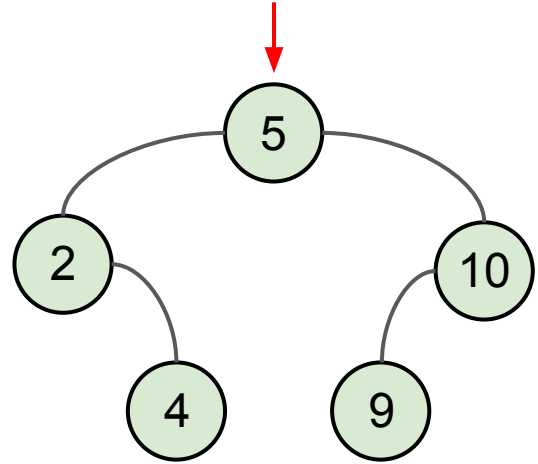
Бинарные деревья поиска

```
def print_asc(root: TreeNode):  
    if not root: return
```



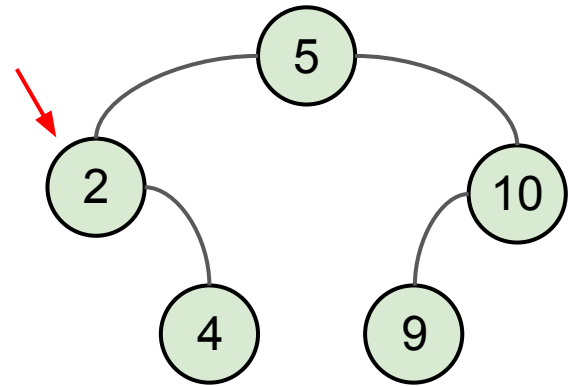
Бинарные деревья поиска

```
def print_asc(root: TreeNode):  
    if not root: return  
  
    print_asc(root.left)
```



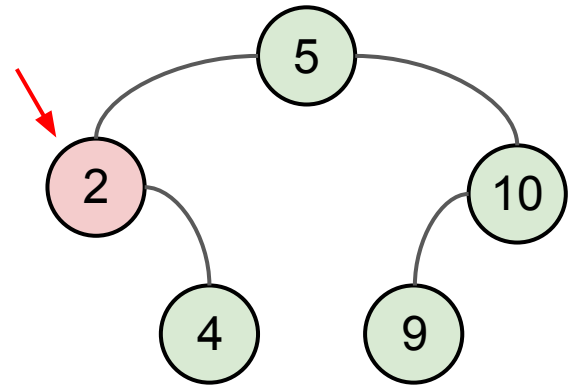
Бинарные деревья поиска

```
def print_asc(root: TreeNode):  
    if not root: return  
  
    print_asc(root.left)
```



Бинарные деревья поиска

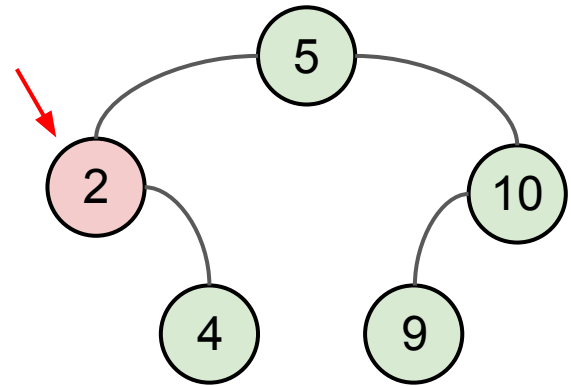
```
def print_asc(root: TreeNode):  
    if not root: return  
  
    print_asc(root.left)  
    print(root.val)
```



2

Бинарные деревья поиска

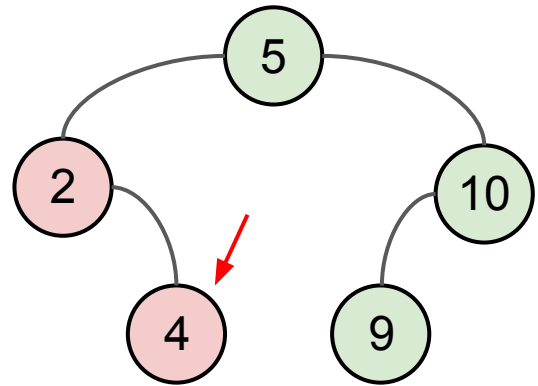
```
def print_asc(root: TreeNode):  
    if not root: return  
  
    print_asc(root.left)  
    print(root.val)  
    print_asc(root.right)
```



2

Бинарные деревья поиска

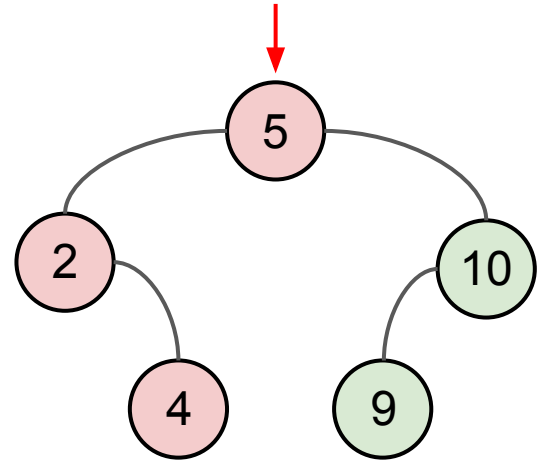
```
def print_asc(root: TreeNode):  
    if not root: return  
  
    print_asc(root.left)  
    print(root.val)  
    print_asc(root.right)
```



2 4

Бинарные деревья поиска

```
def print_asc(root: TreeNode):  
    if not root: return  
  
    print_asc(root.left)  
    print(root.val)  
    print_asc(root.right)
```



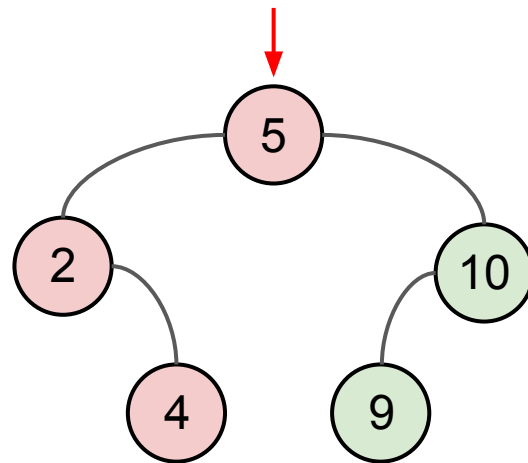
2 4 5

Бинарные деревья поиска

```
def print_asc(root: TreeNode):  
    if not root: return  
  
    print_asc(root.left)  
    print(root.val)  
    print_asc(root.right)
```

Это называется **инфиксный** обход дерева, т.к. обработка вершины находится между рекурсивными вызовами.

Еще бывают **префиксный** и **постфиксный**.



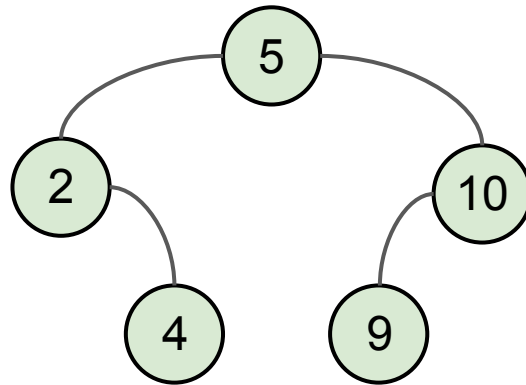
2 4 5

Бинарные деревья поиска

Операции:

1. `find(value)` \rightarrow $O(\text{height})$
 2. `select(i)` \rightarrow $O(???)$
 3. `min/max` \rightarrow $O(\text{height})$
 4. `pred/succ(ptr)` \rightarrow $O(\text{height})$
 5. `rank(value)` \rightarrow $O(???)$
 6. **вывод в пор.** \rightarrow $O(???)$
возрастания
-

7. `insert(value)` \rightarrow $O(\text{height})$
8. `remove(value)` \rightarrow $O(\text{height})$

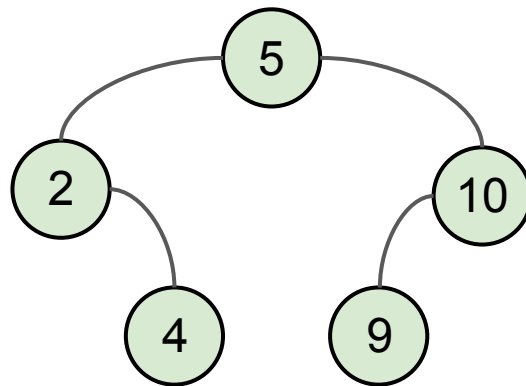


Бинарные деревья поиска

Операции:

1. `find(value)` $\rightarrow O(\text{height})$
 2. `select(i)` $\rightarrow O(???)$
 3. `min/max` $\rightarrow O(\text{height})$
 4. `pred/succ(ptr)` $\rightarrow O(\text{height})$
 5. `rank(value)` $\rightarrow O(???)$
 6. вывод в пор.
возрастания $\rightarrow O(N)$
-

7. `insert(value)` $\rightarrow O(\text{height})$
8. `remove(value)` $\rightarrow O(\text{height})$

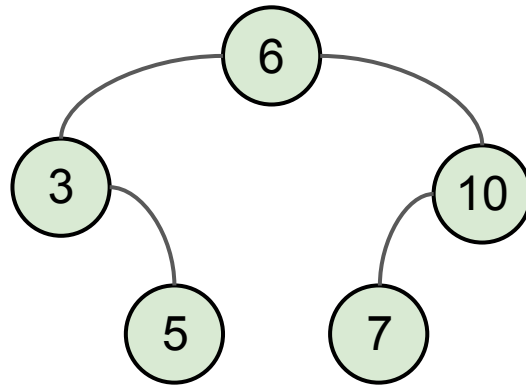


Бинарные деревья поиска

Операции:

1. `find(value)` $\rightarrow O(\text{height})$
 2. `select(i)` $\rightarrow O(???)$
 3. `min/max` $\rightarrow O(\text{height})$
 4. `pred/succ(ptr)` $\rightarrow O(\text{height})$
 5. `rank(value)` $\rightarrow O(???)$
 6. вывод в пор. возрастания $\rightarrow O(N)$
-

7. `insert(value)` $\rightarrow O(\text{height})$
8. `remove(value)` $\rightarrow O(\text{height})$



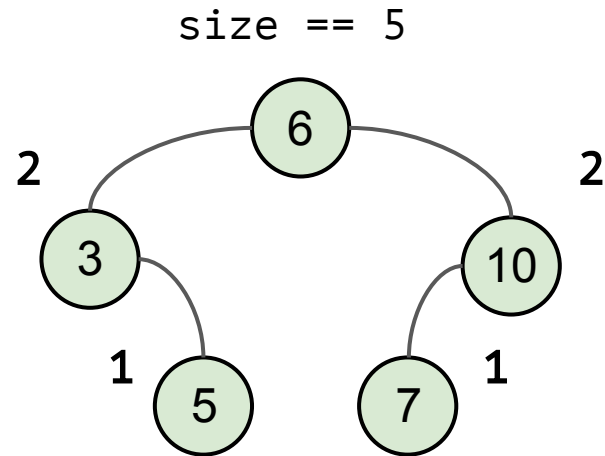
Бинарные деревья поиска

Давайте в каждой вершине хранить
доп. инфу: размер поддерева

Операции:

1. `find(value)` $\rightarrow O(\text{height})$
 2. `select(i)` $\rightarrow O(???)$
 3. `min/max` $\rightarrow O(\text{height})$
 4. `pred/succ(ptr)` $\rightarrow O(\text{height})$
 5. `rank(value)` $\rightarrow O(???)$
 6. вывод в пор. возрастания $\rightarrow O(N)$
-

7. `insert(value)` $\rightarrow O(\text{height})$
8. `remove(value)` $\rightarrow O(\text{height})$



Бинарные деревья поиска

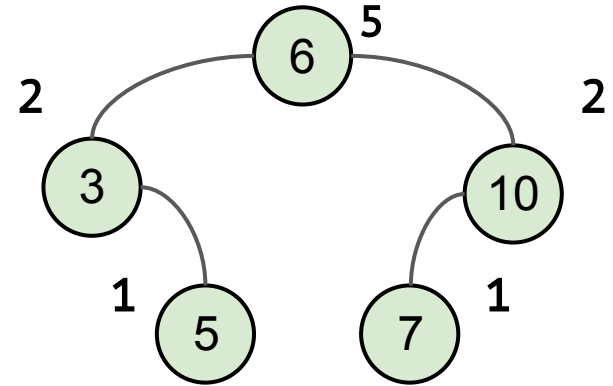
Операции:

1. `find(value)` $\rightarrow O(\text{height})$
 2. `select(i)` $\rightarrow O(???)$
 3. `min/max` $\rightarrow O(\text{height})$
 4. `pred/succ(ptr)` $\rightarrow O(\text{height})$
 5. `rank(value)` $\rightarrow O(???)$
 6. вывод в пор. возрастания $\rightarrow O(N)$
-

7. `insert(value)` $\rightarrow O(\text{height})$
8. `remove(value)` $\rightarrow O(\text{height})$

Давайте в каждой вершине хранить доп. инфу: размер поддерева

`rank` - запускаем поиск от дерева, при поворотах направо увеличиваем счет на размер левого поддерева



Бинарные деревья поиска

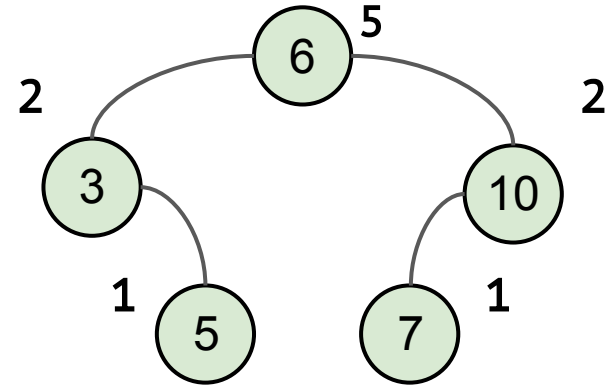
Операции:

1. `find(value)` $\rightarrow O(\text{height})$
 2. `select(i)` $\rightarrow O(???)$
 3. `min/max` $\rightarrow O(\text{height})$
 4. `pred/succ(ptr)` $\rightarrow O(\text{height})$
 5. `rank(value)` $\rightarrow O(\text{height})$
 6. вывод в пор. возрастания $\rightarrow O(N)$
-

7. `insert(value)` $\rightarrow O(\text{height})$
8. `remove(value)` $\rightarrow O(\text{height})$

Давайте в каждой вершине хранить доп. инфу: размер поддерева

`rank` - запускаем поиск от дерева, при поворотах направо увеличиваем счет на размер левого поддерева



Бинарные деревья поиска

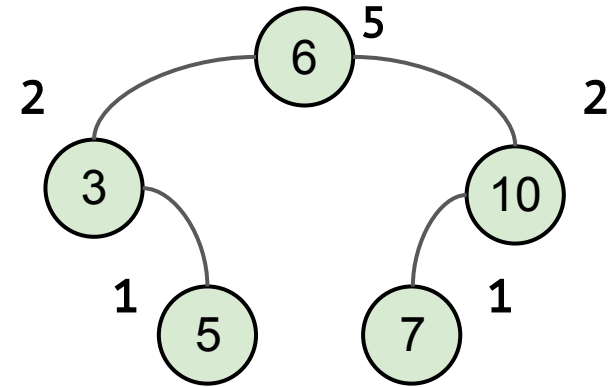
Операции:

- 1. `find(value)` $\rightarrow O(\text{height})$
 - 2. `select(i)` $\rightarrow O(???)$
 - 3. `min/max` $\rightarrow O(\text{height})$
 - 4. `pred/succ(ptr)` $\rightarrow O(\text{height})$
 - 5. `rank(value)` $\rightarrow O(\text{height})$
 - 6. вывод в пор. возрастания $\rightarrow O(N)$
-

- 7. `insert(value)` $\rightarrow O(\text{height})$
- 8. `remove(value)` $\rightarrow O(\text{height})$

i -ая статистика ищется модификацией поиска:

- 1) если попали в вершину с `size-ом` левого поддерева $== i - 1 \Rightarrow$ нашли i -ую статистику



Бинарные деревья поиска

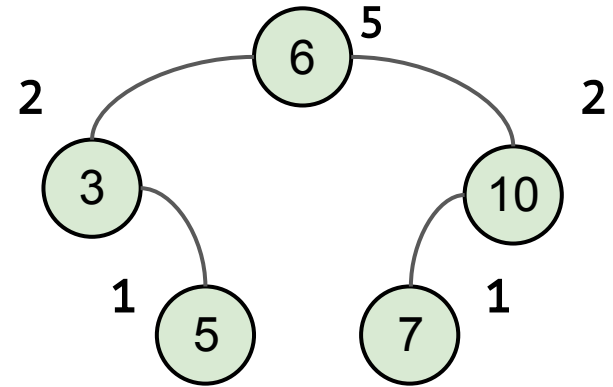
Операции:

- 1. `find(value)` $\rightarrow O(\text{height})$
 - 2. `select(i)` $\rightarrow O(???)$
 - 3. `min/max` $\rightarrow O(\text{height})$
 - 4. `pred/succ(ptr)` $\rightarrow O(\text{height})$
 - 5. `rank(value)` $\rightarrow O(\text{height})$
 - 6. вывод в пор. возрастания $\rightarrow O(N)$
-

- 7. `insert(value)` $\rightarrow O(\text{height})$
- 8. `remove(value)` $\rightarrow O(\text{height})$

`i`-ая статистика ищется модификацией поиска:

- 2) иначе, продолжаем поиск, если в правом поддереве, то ищем там статистику `i - root.val - 1`



Бинарные деревья поиска

Операции:

1. `find(value)` $\rightarrow O(\text{height})$
 2. `select(i)` $\rightarrow O(\text{height})$
 3. `min/max` $\rightarrow O(\text{height})$
 4. `pred/succ(ptr)` $\rightarrow O(\text{height})$
 5. `rank(value)` $\rightarrow O(\text{height})$
 6. вывод в пор.
возрастания $\rightarrow O(N)$
-

7. `insert(value)` $\rightarrow O(\text{height})$
8. `remove(value)` $\rightarrow O(\text{height})$

Бинарные деревья поиска

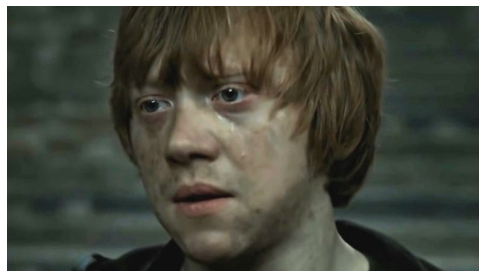
Операции:

1. `find(value)` $\rightarrow O(\text{height})$
 2. `select(i)` $\rightarrow O(\text{height})$
 3. `min/max` $\rightarrow O(\text{height})$
 4. `pred/succ(ptr)` $\rightarrow O(\text{height})$
 5. `rank(value)` $\rightarrow O(\text{height})$
 6. вывод в пор. $\rightarrow O(N)$
возрастания
-

7. `insert(value)` $\rightarrow O(\text{height})$
8. `remove(value)` $\rightarrow O(\text{height})$

Назревает проблема!

Обещали логарифм, а
дают какой-то $O(\text{height})$



Бинарные деревья поиска

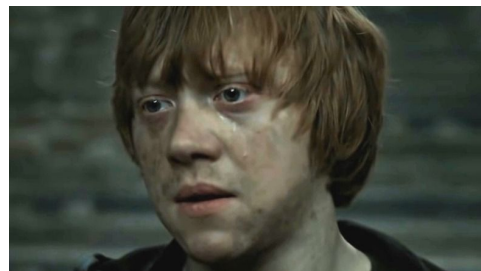
Операции:

1. `find(value)` $\rightarrow O(\text{height})$
 2. `select(i)` $\rightarrow O(\text{height})$
 3. `min/max` $\rightarrow O(\text{height})$
 4. `pred/succ(ptr)` $\rightarrow O(\text{height})$
 5. `rank(value)` $\rightarrow O(\text{height})$
 6. вывод в пор. возрастания $\rightarrow O(N)$
-

7. `insert(value)` $\rightarrow O(\text{height})$
8. `remove(value)` $\rightarrow O(\text{height})$

Назревает проблема!

Обещали логарифм, а дают какой-то $O(\text{height})$



Значит нам нужны такие BST, чтобы высота у дерева всегда была $\log N$

AVL-деревья

АВЛ-деревья

Авторы: Адельсон-Вельский Г.М. и Ландис Е. М., 1962 год

АВЛ-деревья

Авторы: Адельсон-Вельский Г.М. и Ландис Е. М., 1962 год

Идея: возьмем обычное BST и добавим ограничение

АВЛ-деревья

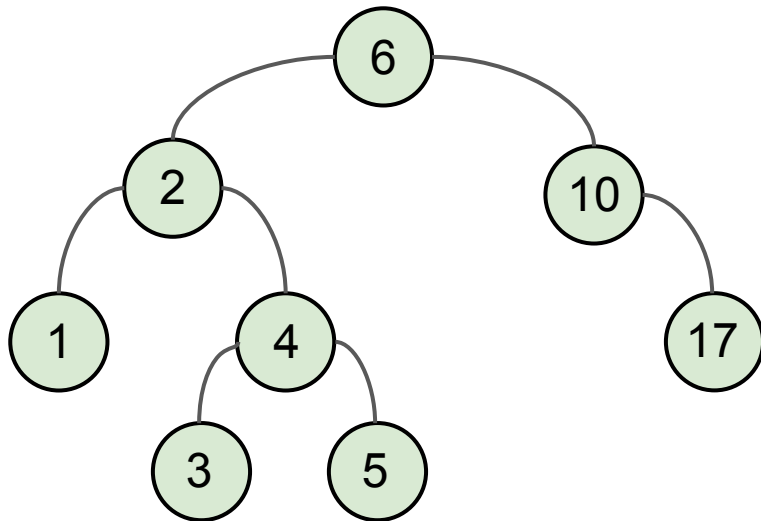
Авторы: Адельсон-Вельский Г.М. и Ландис Е. М., 1962 год

Идея: возьмем обычное BST и добавим ограничение, что для любой вершины **высоты** левого и правого поддеревьев отличаются не более, чем на **1**.

АВЛ-деревья

Авторы: Адельсон-Вельский Г.М. и Ландис Е. М., 1962 год

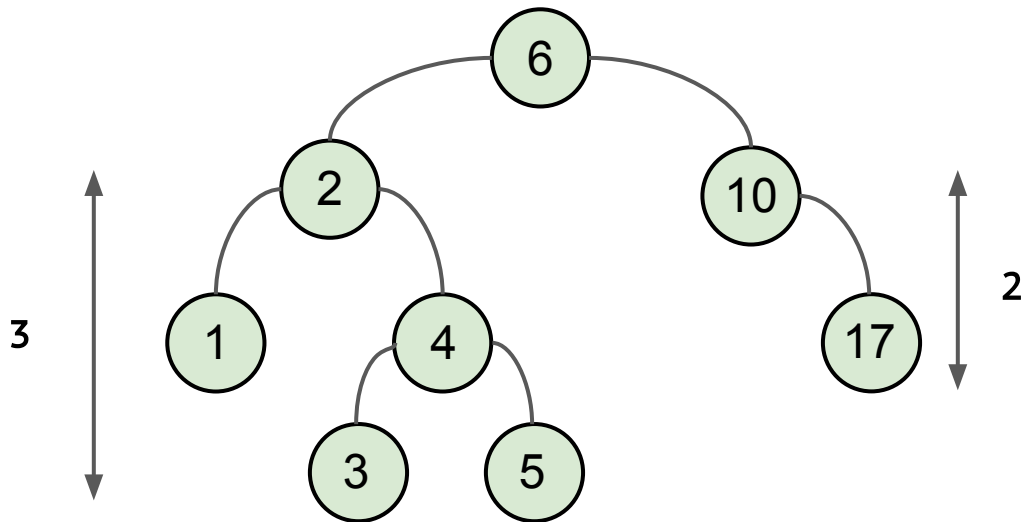
Идея: возьмем обычное BST и добавим ограничение, что для любой вершины **высоты** левого и правого поддеревьев отличаются не более, чем на **1**.



АВЛ-деревья

Авторы: Адельсон-Вельский Г.М. и Ландис Е. М., 1962 год

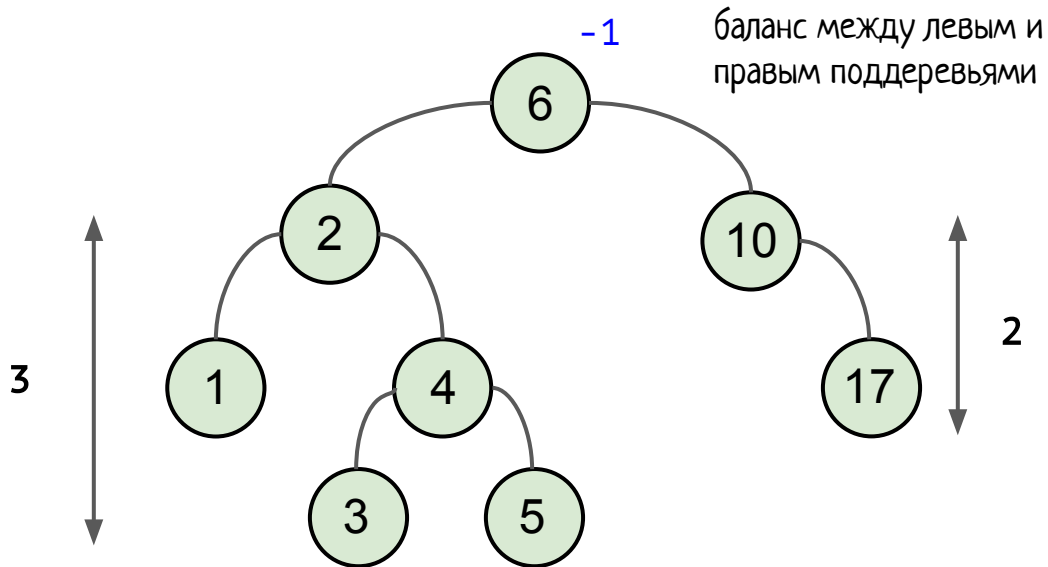
Идея: возьмем обычное BST и добавим ограничение, что для любой вершины **высоты** левого и правого поддеревьев отличаются не более, чем на **1**.



АВЛ-деревья

Авторы: Адельсон-Вельский Г.М. и Ландис Е. М., 1962 год

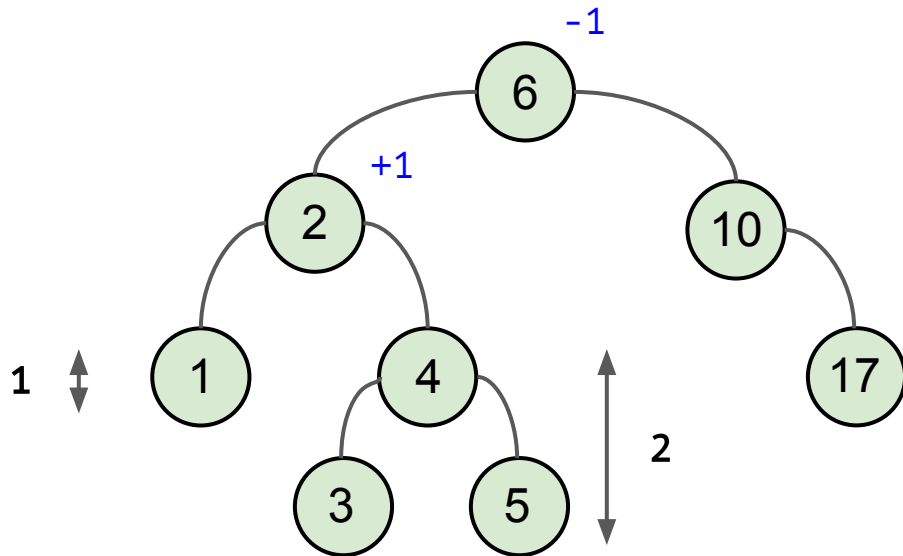
Идея: возьмем обычное BST и добавим ограничение, что для любой вершины **высоты** левого и правого поддеревьев отличаются не более, чем на **1**.



АВЛ-деревья

Авторы: Адельсон-Вельский Г.М. и Ландис Е. М., 1962 год

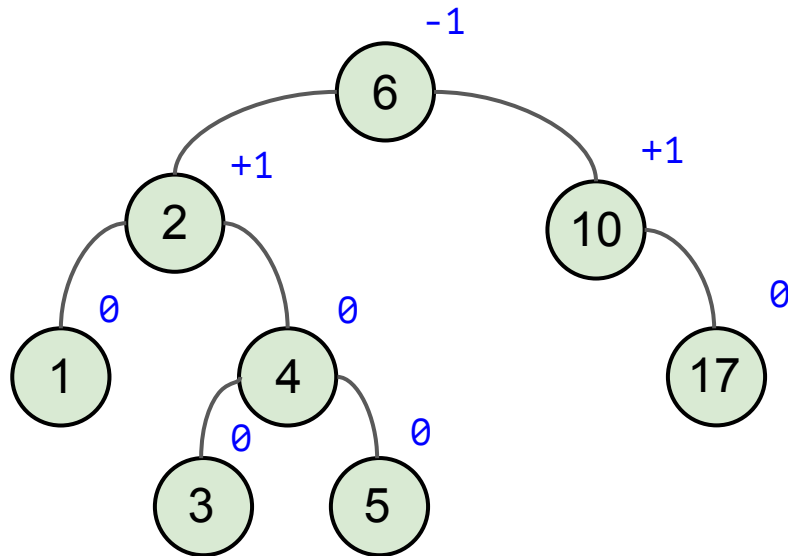
Идея: возьмем обычное BST и добавим ограничение, что для любой вершины **высоты** левого и правого поддеревьев отличаются не более, чем на **1**.



АВЛ-деревья

Авторы: Адельсон-Вельский Г.М. и Ландис Е. М., 1962 год

Идея: возьмем обычное BST и добавим ограничение, что для любой вершины **высоты** левого и правого поддеревьев отличаются не более, чем на **1**.

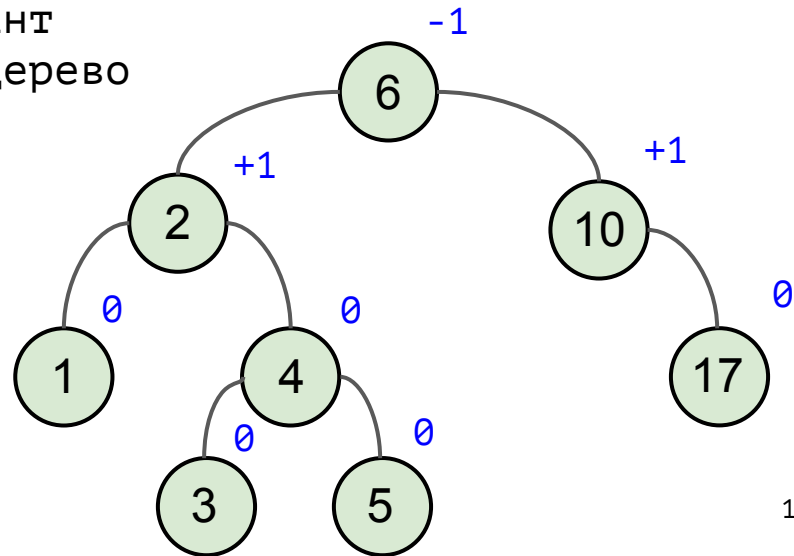


АВЛ-деревья

Авторы: Адельсон-Вельский Г.М. и Ландис Е. М., 1962 год

Идея: возьмем обычное BST и добавим ограничение, что для любой вершины **высоты** левого и правого поддеревьев отличаются не более, чем на **1**.

Далее поддерживаем этот инвариант во всех операциях, изменяющих дерево (т.е. insert и remove)



АВЛ-деревья

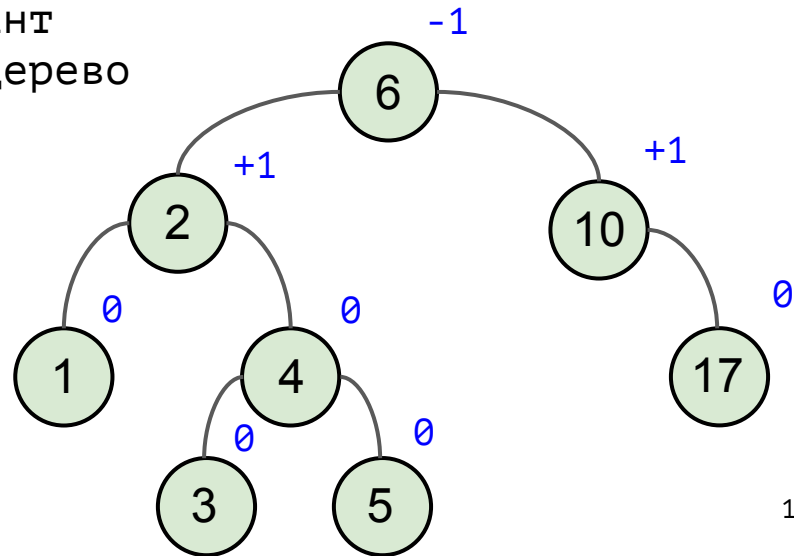
Авторы: Адельсон-Вельский Г.М. и Ландис Е. М., 1962 год

Идея: возьмем обычное BST и добавим ограничение, что для любой вершины **высоты** левого и правого поддеревьев отличаются не более, чем на **1**.

Далее поддерживаем этот инвариант во всех операциях, изменяющих дерево (т.е. insert и remove)

А почему это вообще работает?

Какая высота?




AVL-деревья

Утверждение: пусть $S(k)$ - минимальное количество вершин в AVL-дереве высоты k .

АВЛ-деревья

Утверждение: пусть $S(k)$ - минимальное количество вершин в AVL-дереве высоты k .

$$\text{тогда } S(k) = F(k + 1) - 1$$



$(k + 1)$ -ое число Фибоначчи

АВЛ-деревья

Утверждение: пусть $S(k)$ - минимальное количество вершин в AVL-дереве высоты k .

$$\text{тогда } S(k) = F(k + 1) - 1$$

Доказательство: индукция по k

$(k + 1)$ -ое число Фибоначчи



АВЛ-деревья

Утверждение: пусть $S(k)$ - минимальное количество вершин в AVL-дереве высоты k .

$$\text{тогда } S(k) = F(k + 1) - 1$$

Доказательство: индукция по k

$(k + 1)$ -ое число Фибоначчи

height = 1



$$S(k) = 1 = F(2) - 1$$

АВЛ-деревья

Утверждение: пусть $S(k)$ - минимальное количество вершин в AVL-дереве высоты k .

$$\text{тогда } S(k) = F(k + 1) - 1$$

Доказательство: индукция по k

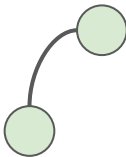
$(k + 1)$ -ое число Фибоначчи

height = 1



$$S(k) = 1 = F(2) - 1$$

height = 2



$$S(k) = 2 = F(3) - 1$$

АВЛ-деревья

Утверждение: пусть $S(k)$ - минимальное количество вершин в AVL-дереве высоты k .

$$\text{тогда } S(k) = F(k + 1) - 1$$

Доказательство: индукция по k

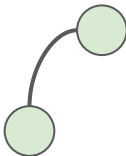
$(k + 1)$ -ое число Фибоначчи

height = 1



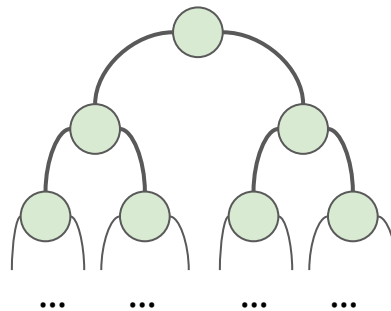
$$S(k) = 1 = F(2) - 1$$

height = 2



$$S(k) = 2 = F(3) - 1$$

height = k



АВЛ-деревья

Утверждение: пусть $S(k)$ - минимальное количество вершин в AVL-дереве высоты k .

$$\text{тогда } S(k) = F(k + 1) - 1$$

Доказательство: индукция по k

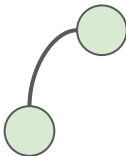
$(k + 1)$ -ое число Фибоначчи

height = 1



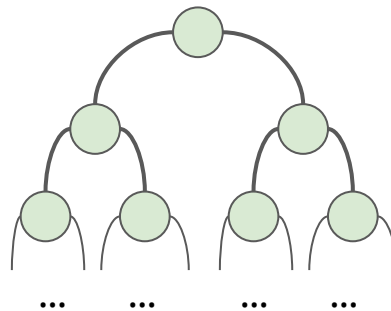
$$S(k) = 1 = F(2) - 1$$

height = 2



$$S(k) = 2 = F(3) - 1$$

height = k



Тогда хотя бы у одного из детей должна быть высота $k - 1$

АВЛ-деревья

Утверждение: пусть $S(k)$ - минимальное количество вершин в AVL-дереве высоты k .

$$\text{тогда } S(k) = F(k + 1) - 1$$

Доказательство: индукция по k

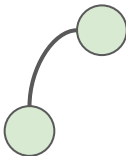
$(k + 1)$ -ое число Фибоначчи

height = 1



$$S(k) = 1 = F(2) - 1$$

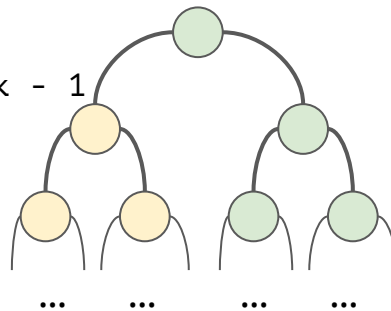
height = 2



$$S(k) = 2 = F(3) - 1$$

height = k

$h = k - 1$




Тогда хотя бы у одного из детей должна быть высота $k - 1$

АВЛ-деревья

Утверждение: пусть $S(k)$ - минимальное количество вершин в AVL-дереве высоты k .

$$\text{тогда } S(k) = F(k + 1) - 1$$

 $(k + 1)$ -ое число Фибоначчи

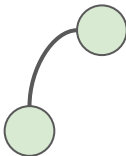
Доказательство: индукция по k

height = 1



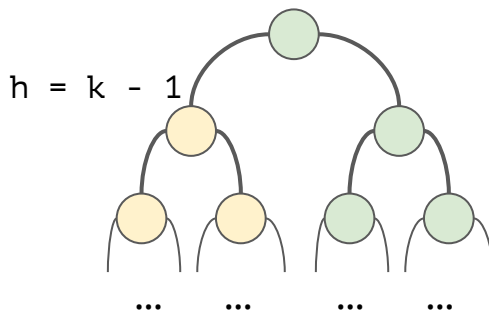
$$S(k) = 1 = F(2) - 1$$

height = 2



$$S(k) = 2 = F(3) - 1$$

height = k



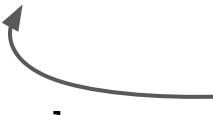
Тогда хотя бы у одного из детей должна быть высота $k - 1$

А какие тогда варианты у правого сына?

АВЛ-деревья

Утверждение: пусть $S(k)$ - минимальное количество вершин в AVL-дереве высоты k .

$$\text{тогда } S(k) = F(k + 1) - 1$$

 $(k + 1)$ -ое число Фибоначчи

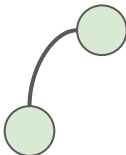
Доказательство: индукция по k

height = 1



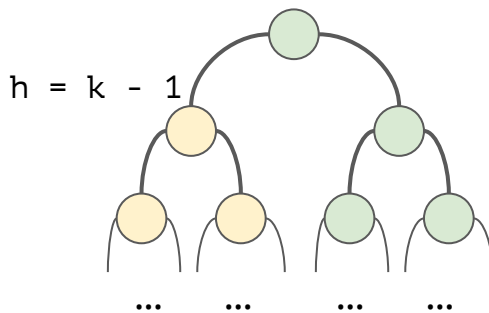
$$S(k) = 1 = F(2) - 1$$

height = 2



$$S(k) = 2 = F(3) - 1$$

height = k



Тогда хотя бы у одного из детей должна быть высота $k - 1$

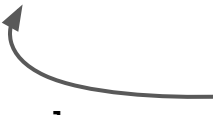
А какие тогда варианты у правого сына?

$\{-1, 0, +1\}$

АВЛ-деревья

Утверждение: пусть $S(k)$ - минимальное количество вершин в AVL-дереве высоты k .

$$\text{тогда } S(k) = F(k + 1) - 1$$

 $(k + 1)$ -ое число Фибоначчи

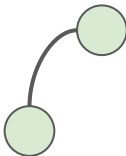
Доказательство: индукция по k

height = 1



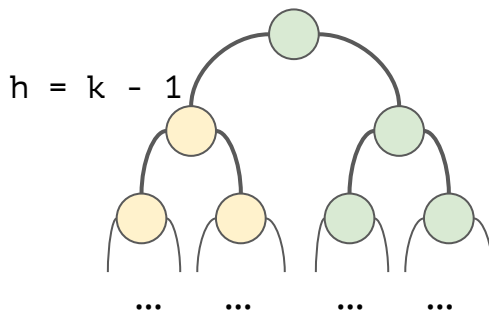
$$S(k) = 1 = F(2) - 1$$

height = 2



$$S(k) = 2 = F(3) - 1$$

height = k



Тогда хотя бы у одного из детей должна быть высота $k - 1$

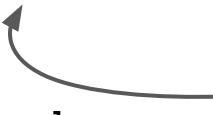
А какие тогда варианты у правого сына?

$\{k-2, k-1, k\}$

АВЛ-деревья

Утверждение: пусть $S(k)$ - минимальное количество вершин в AVL-дереве высоты k .

$$\text{тогда } S(k) = F(k + 1) - 1$$

 $(k + 1)$ -ое число Фибоначчи

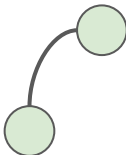
Доказательство: индукция по k

height = 1



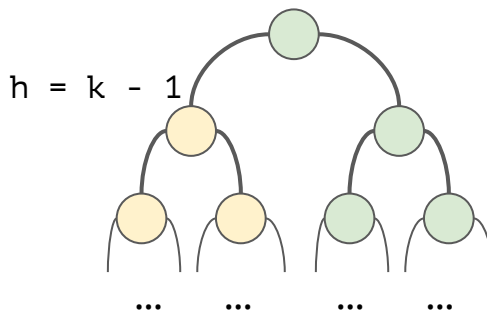
$$S(k) = 1 = F(2) - 1$$

height = 2



$$S(k) = 2 = F(3) - 1$$

height = k



Тогда хотя бы у одного из детей должна быть высота $k - 1$

А какие тогда варианты у правого сына?

$\{k-2, k-1, \text{X}\}$

АВЛ-деревья

Утверждение: пусть $S(k)$ - **минимальное** количество вершин в AVL-дереве высоты k .

$$\text{тогда } S(k) = F(k + 1) - 1$$

Доказательство: индукция по k

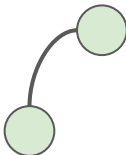
$(k + 1)$ -ое число Фибоначчи

height = 1



$$S(k) = 1 = F(2) - 1$$

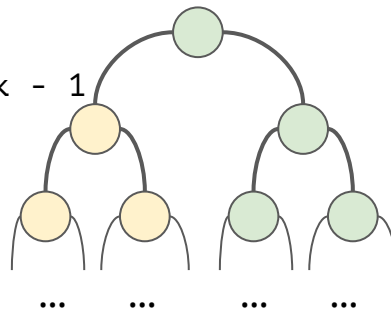
height = 2



$$S(k) = 2 = F(3) - 1$$

height = k

$h = k - 1$



Тогда хотя бы у одного из детей должна быть высота $k - 1$

В **минимальном** дереве у правого сына высота $k - 2$

АВЛ-деревья

Утверждение: пусть $S(k)$ - минимальное количество вершин в AVL-дереве высоты k .

$$\text{тогда } S(k) = F(k + 1) - 1$$

Доказательство: индукция по k

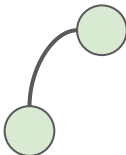
$(k + 1)$ -ое число Фибоначчи

height = 1



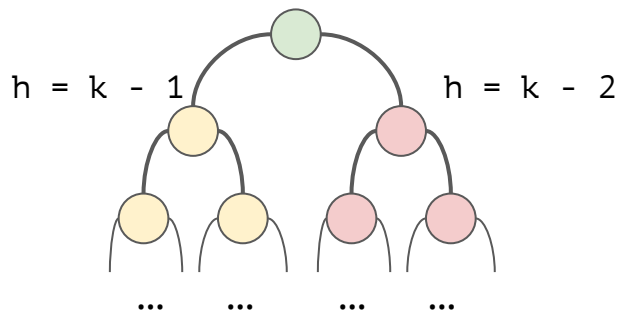
$$S(k) = 1 = F(2) - 1$$

height = 2



$$S(k) = 2 = F(3) - 1$$

height = k



АВЛ-деревья

Утверждение: пусть $S(k)$ - минимальное количество вершин в AVL-дереве высоты k .

$$\text{тогда } S(k) = F(k + 1) - 1$$

Доказательство: индукция по k

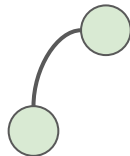
$(k + 1)$ -ое число Фибоначчи

height = 1



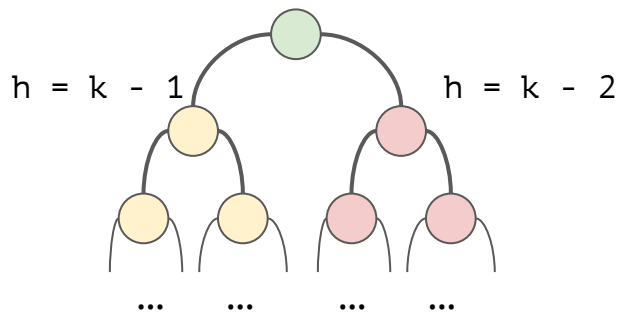
$$S(k) = 1 = F(2) - 1$$

height = 2



$$S(k) = 2 = F(3) - 1$$

height = k



$$S(k) = 1 + S(k - 1) + S(k - 2)$$

АВЛ-деревья

Утверждение: пусть $S(k)$ - минимальное количество вершин в AVL-дереве высоты k .

$$\text{тогда } S(k) = F(k + 1) - 1$$

Доказательство: индукция по k

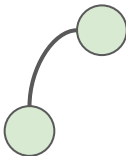
$(k + 1)$ -ое число Фибоначчи

height = 1



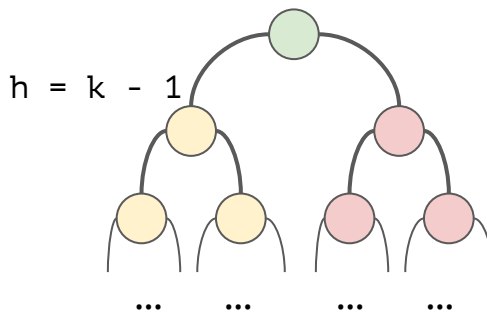
$$S(k) = 1 = F(2) - 1$$

height = 2



$$S(k) = 2 = F(3) - 1$$

height = k



$$S(k) = 1 + S(k - 1) + S(k - 2)$$

$$= 1 + F(k) - 1 + F(k - 1) - 1$$

АВЛ-деревья

Утверждение: пусть $S(k)$ - минимальное количество вершин в AVL-дереве высоты k .

$$\text{тогда } S(k) = F(k + 1) - 1$$

Доказательство: индукция по k

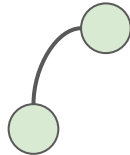
$(k + 1)$ -ое число Фибоначчи

height = 1



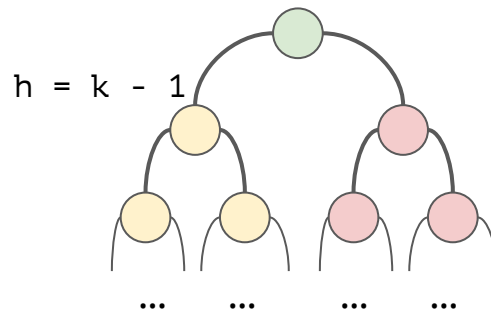
$$S(k) = 1 = F(2) - 1$$

height = 2



$$S(k) = 2 = F(3) - 1$$

height = k



$$S(k) = 1 + S(k - 1) + S(k - 2)$$

$$= 1 + F(k) - 1 + F(k - 1) - 1$$

$$= F(k + 1) - 1$$



АВЛ-деревья

Утверждение: пусть $S(k)$ - минимальное количество вершин в AVL-дереве высоты k .

$$\text{тогда } S(k) = F(k + 1) - 1$$

Утверждение: высота AVL-дерева из N вершин - это $O(\log N)$

АВЛ-деревья

Утверждение: пусть $S(k)$ - минимальное количество вершин в AVL-дереве высоты k .

$$\text{тогда } S(k) = F(k + 1) - 1$$

Утверждение: высота AVL-дерева из N вершин - это $O(\log N)$

Действительно, как мы узнали на прошлой лекции:

$$F_{k+2} \geq \phi^k$$

АВЛ-деревья

Утверждение: пусть $S(k)$ - минимальное количество вершин в AVL-дереве высоты k .

$$\text{тогда } S(k) = F(k + 1) - 1$$

Утверждение: высота AVL-дерева из N вершин - это $O(\log N)$

Действительно, как мы узнали на прошлой лекции:

$$F_{k+2} \geq \phi^k \quad \text{Тогда: } S(k) = F_{k+1} - 1 \geq \phi^{k-1} - 1$$

АВЛ-деревья

Утверждение: пусть $S(k)$ - минимальное количество вершин в AVL-дереве высоты k .

$$\text{тогда } S(k) = F(k + 1) - 1$$

Утверждение: высота AVL-деревя из N вершин - это $O(\log N)$

Действительно, как мы узнали на прошлой лекции:

$$F_{k+2} \geq \phi^k \quad \text{Тогда: } S(k) = F_{k+1} - 1 \geq \phi^{k-1} - 1 \Rightarrow N \geq S(k) \geq \phi^{k-1} - 1$$

АВЛ-деревья

Утверждение: пусть $S(k)$ - минимальное количество вершин в AVL-дереве высоты k .

$$\text{тогда } S(k) = F(k + 1) - 1$$

Утверждение: высота AVL-деревя из N вершин - это $O(\log N)$

Действительно, как мы узнали на прошлой лекции:

$$F_{k+2} \geq \phi^k \quad \text{Тогда: } S(k) = F_{k+1} - 1 \geq \phi^{k-1} - 1 \Rightarrow N \geq S(k) \geq \phi^{k-1} - 1$$

Тогда прологарифмируем по основанию ϕ и получим: $k = O(\log N)$ □

Бинарные деревья поиска

Операции:

1. `find(value)` $\rightarrow O(\text{height})$
 2. `select(i)` $\rightarrow O(\text{height})$
 3. `min/max` $\rightarrow O(\text{height})$
 4. `pred/succ(ptr)` $\rightarrow O(\text{height})$
 5. `rank(value)` $\rightarrow O(\text{height})$
 6. вывод в пор. возрастания $\rightarrow O(N)$
-

7. `insert(value)` $\rightarrow O(\text{height})$
8. `remove(value)` $\rightarrow O(\text{height})$

Назревает проблема!

Обещали логарифм, а дают какой-то $O(\text{height})$



Значит нам нужны такие BST, чтобы высота у дерева всегда была $\log N$

АВЛ-деревья

Операции:

1. `find(value)` $\rightarrow O(\log N)$
 2. `select(i)` $\rightarrow O(\log N)$
 3. `min/max` $\rightarrow O(\log N)$
 4. `pred/succ(ptr)` $\rightarrow O(\log N)$
 5. `rank(value)` $\rightarrow O(\log N)$
 6. вывод в пор. возрастания $\rightarrow O(N)$
-

7. `insert(value)` $\rightarrow O(???)$
8. `remove(value)` $\rightarrow O(???)$

В АВЛ-дереве
действительно высота
всегда порядка $\log N$

АВЛ-деревья

Операции:

1. `find(value)` $\rightarrow O(\log N)$
 2. `select(i)` $\rightarrow O(\log N)$
 3. `min/max` $\rightarrow O(\log N)$
 4. `pred/succ(ptr)` $\rightarrow O(\log N)$
 5. `rank(value)` $\rightarrow O(\log N)$
 6. вывод в пор.
возрастания $\rightarrow O(N)$
-

7. `insert(value)` $\rightarrow O(???)$
8. `remove(value)` $\rightarrow O(???)$

В АВЛ-дереве
действительно высота
всегда порядка $\log N$



Но как поддержка
инвариантов повлияет на
добавление и удаление?

АВЛ-деревья

Операции:

1. `find(value)` $\rightarrow O(\log N)$
 2. `select(i)` $\rightarrow O(\log N)$
 3. `min/max` $\rightarrow O(\log N)$
 4. `pred/succ(ptr)` $\rightarrow O(\log N)$
 5. `rank(value)` $\rightarrow O(\log N)$
 6. вывод в пор.
возрастания $\rightarrow O(N)$
-

7. `insert(value)` $\rightarrow O(???)$
8. `remove(value)` $\rightarrow O(???)$

TO BE CONTINUED...

В АВЛ-дереве
действительно высота
всегда порядка $\log N$



Но как поддержка
инвариантов повлияет на
добавление и удаление?