ВЫСШАЯ ШКОЛА ЭКОНОМИКИ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

Департамент программной инженерии

Алгоритмы и структуры данных

Семинар №9. 2021-2022 учебный год

Нестеров Роман Александрович, *ДПИ ФКН и НУЛ ПОИС*

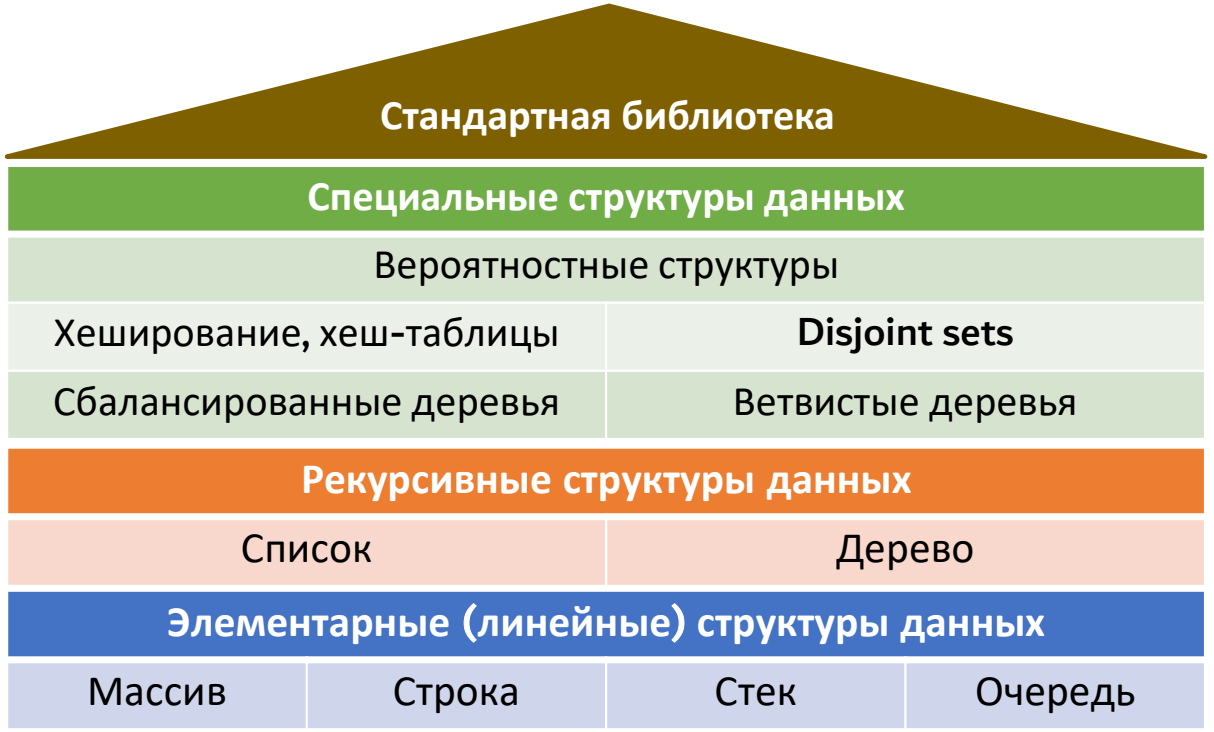
Бессмертный Александр Игоревич, *ДПИ ФКН*

Higher School of Economics

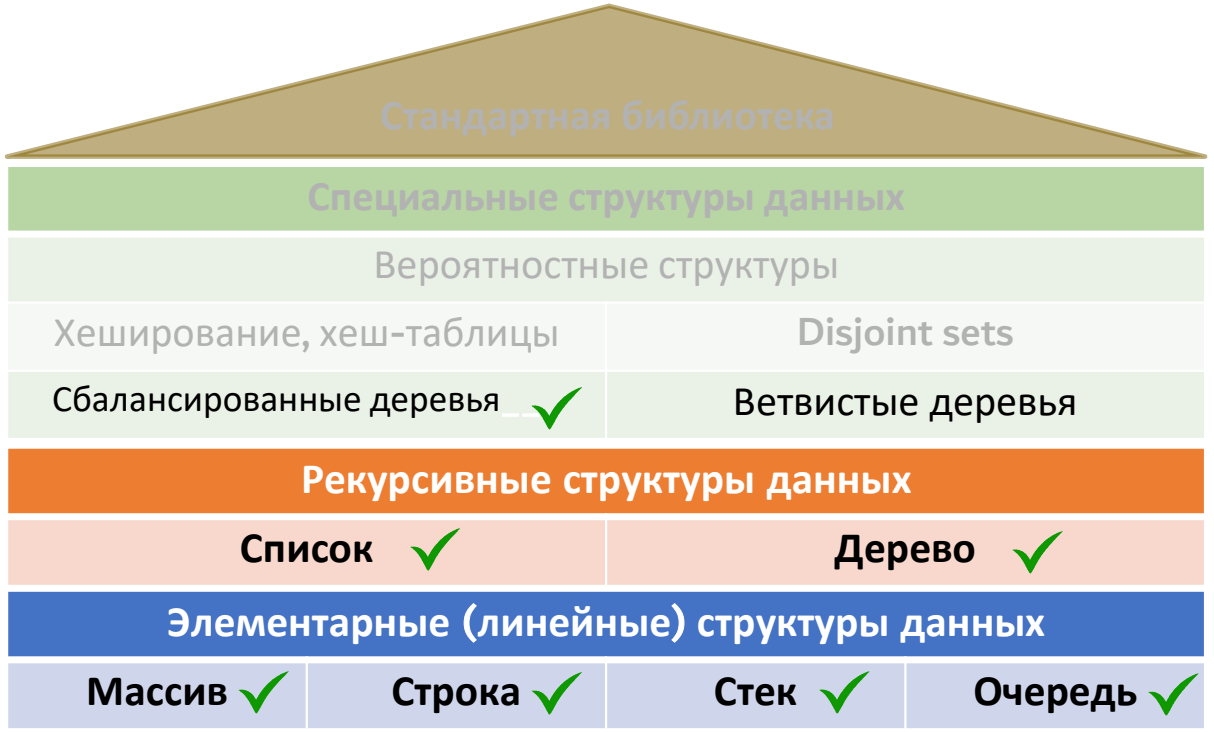
Faculty of Computer science

La práctica hace al maestro

Где мы?


Стандартная библиотека
Специальные структуры данных
Вероятностные структуры
Хеширование, хеш-таблицы
Disjoint sets
Сбалансированные деревья
Ветвистые деревья
Рекурсивные структуры данных
Список
Дерево
Элементарные (линейные) структуры данных
Массив
Строка
Стек
Очередь

Где мы?


Стандартная библиотека
Специальные структуры данных
Вероятностные структуры
Хеширование, хеш-таблицы
Disjoint sets
Сбалансированные деревья
Ветвистые деревья
Рекурсивные структуры данных
Список
Дерево
Элементарные (линейные) структуры данных
Массив
Строка
Стек
Очередь

План

- **Splay**-деревья – «выворачивающиеся» самобалансирующиеся деревья
- B-дерево – плотное хранение информации
- B+-дерево – многоуровневый индекс

Splay-дерево

Splay-деревья. Основные идеи

Уходят корнями в **1983 год** (Р. Тарьян и Д. Слейтер)

- Не являются постоянно сбалансированными и перестраиваются на основании поступающих запросов.
- Узел, к которому был недавно получен доступ, перемещается в корень дерева посредством поворотов.

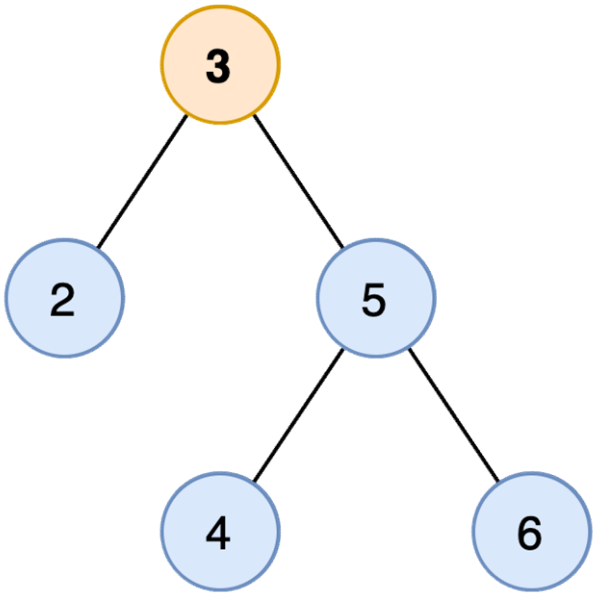
Splay-деревья. Повороты

При выполнении операции с деревом (поиск, вставка, и удаление) выполняется проталкивание (**`splay`**) узла в корень дерева.

- **`Zig`** – Правый поворот
- **`Zag`** – Левый поворот

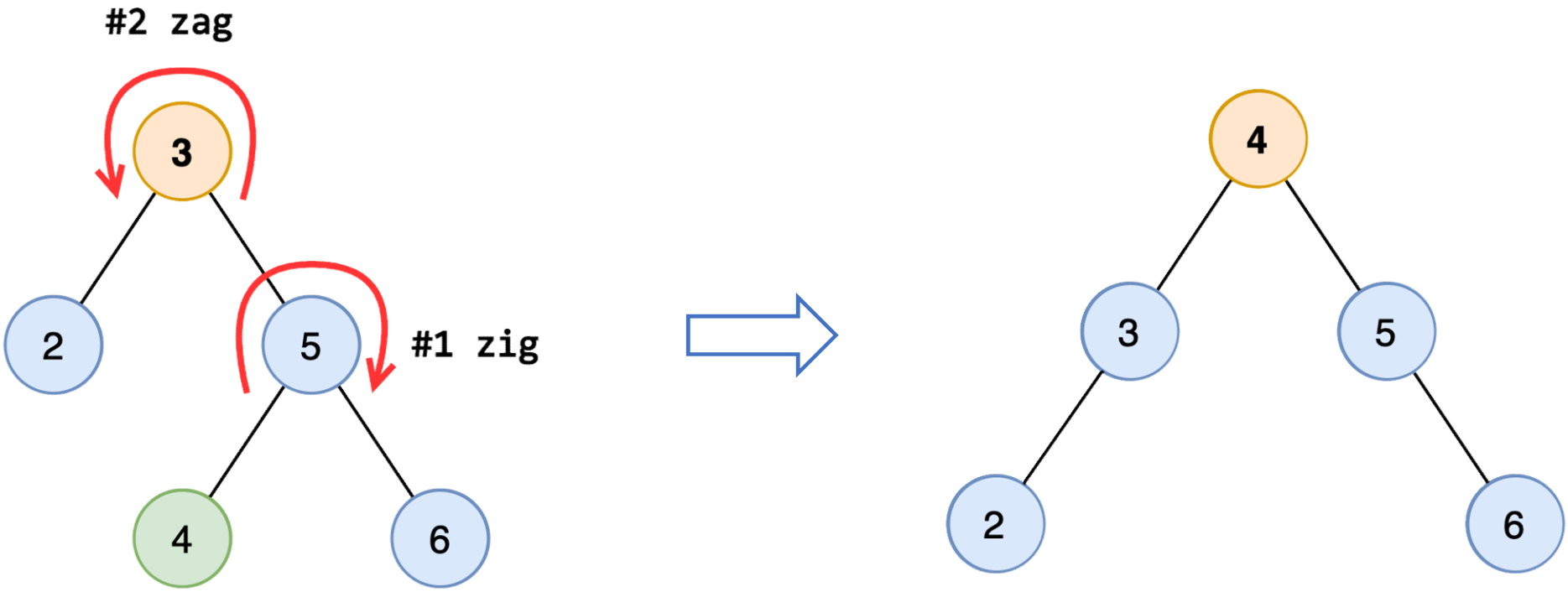
Splay-деревья. Сложные повороты

Проталкиваем вершину с ключом **4** в корень дерева.



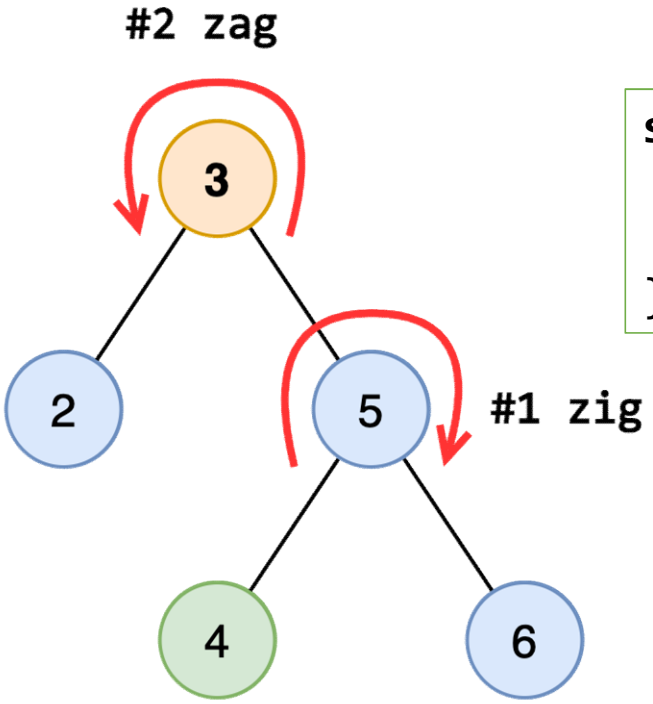
Splay-деревья. Сложные повороты

Проталкиваем вершину с ключом **4** в корень дерева.

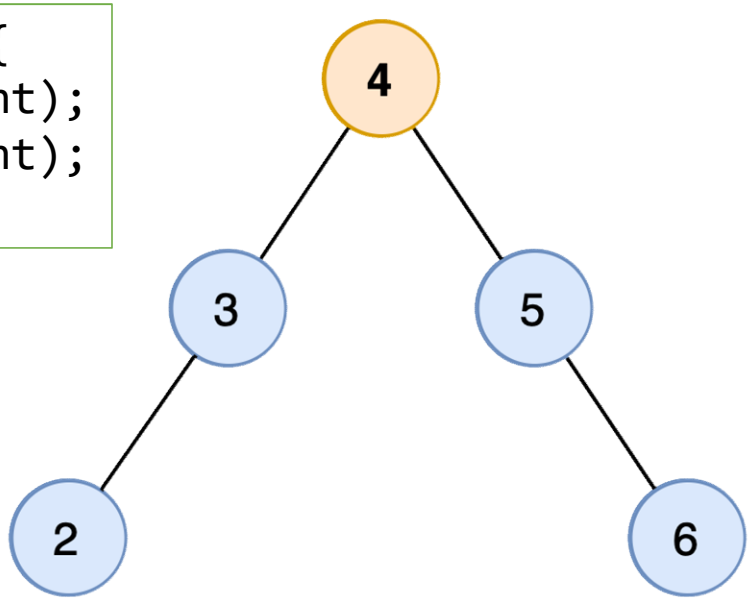


Splay-деревья. Сложные повороты

Проталкиваем вершину с ключом **4** в корень дерева.




```
splay(Node *curr) {
    zig(curr->parent);
    zag(curr->parent);
}
```



Splay-деревья. Повороты

Подобным образом определяется **6** вариантов поворотов:

1. Одинарный **`zig`** или **`zag`**
2. Двойной **`zig-zig`** или **`zag-zag`**
3. Сложный **`zig-zag`** или **`zag-zig`**

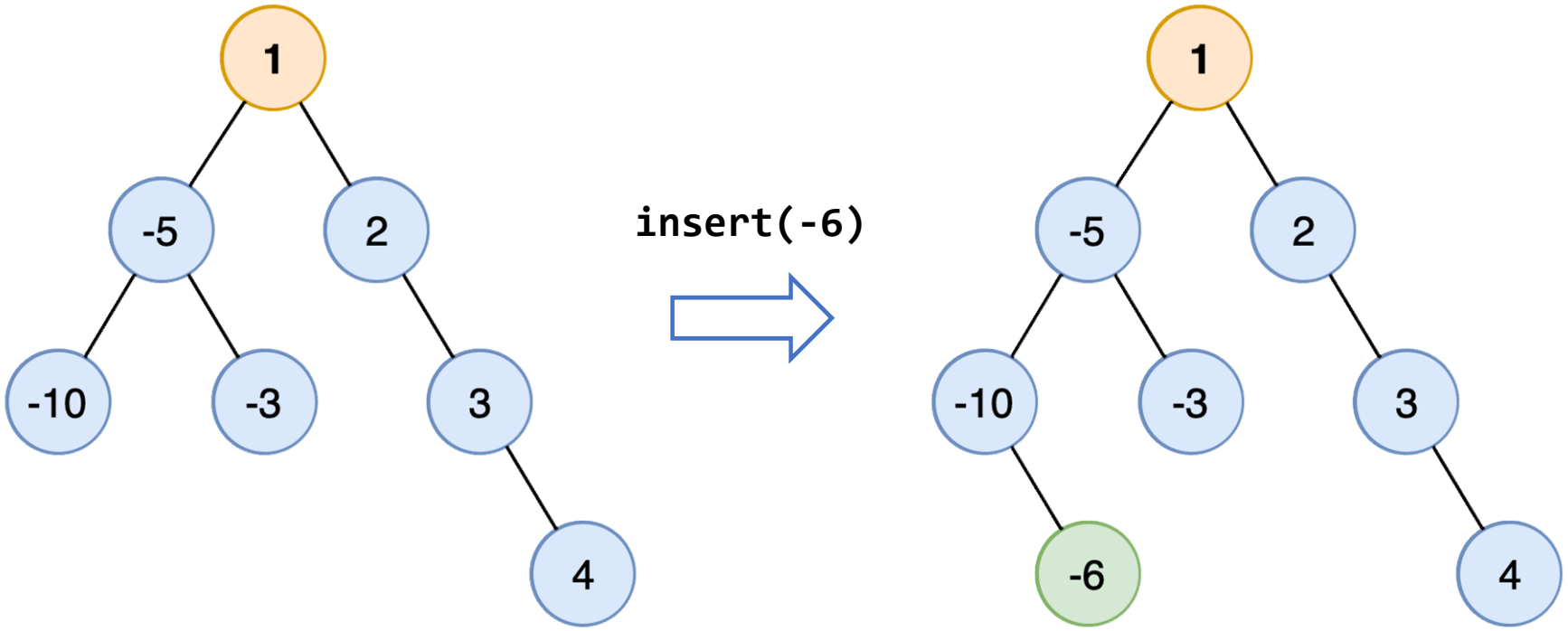
Выбор конкретного вида поворота определяется расположением проталкиваемой вершины.

Splay-деревья. Вставка узла

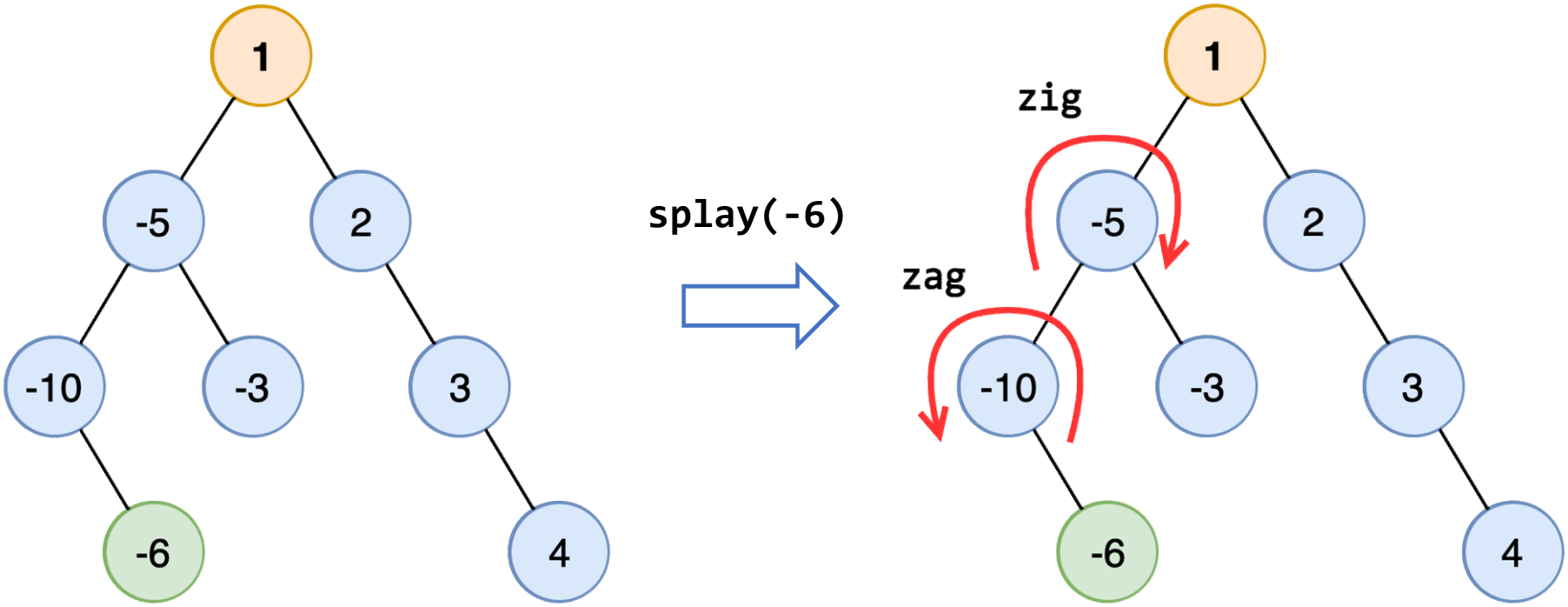
Вставка ключа **x** разбивается на две задачи:

1. Вставка ключа в подходящее место.
2. Проталкивание соответствующего узла в корень дерева.

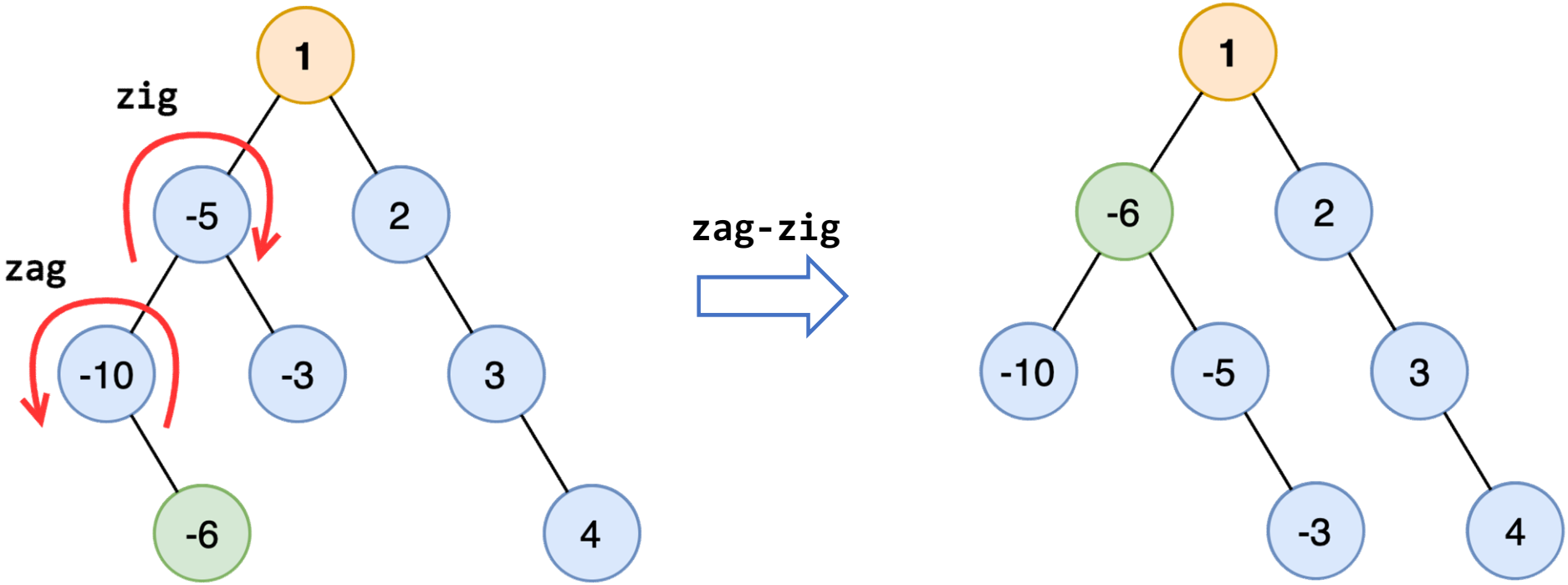
Splay-деревья. Вставка узла



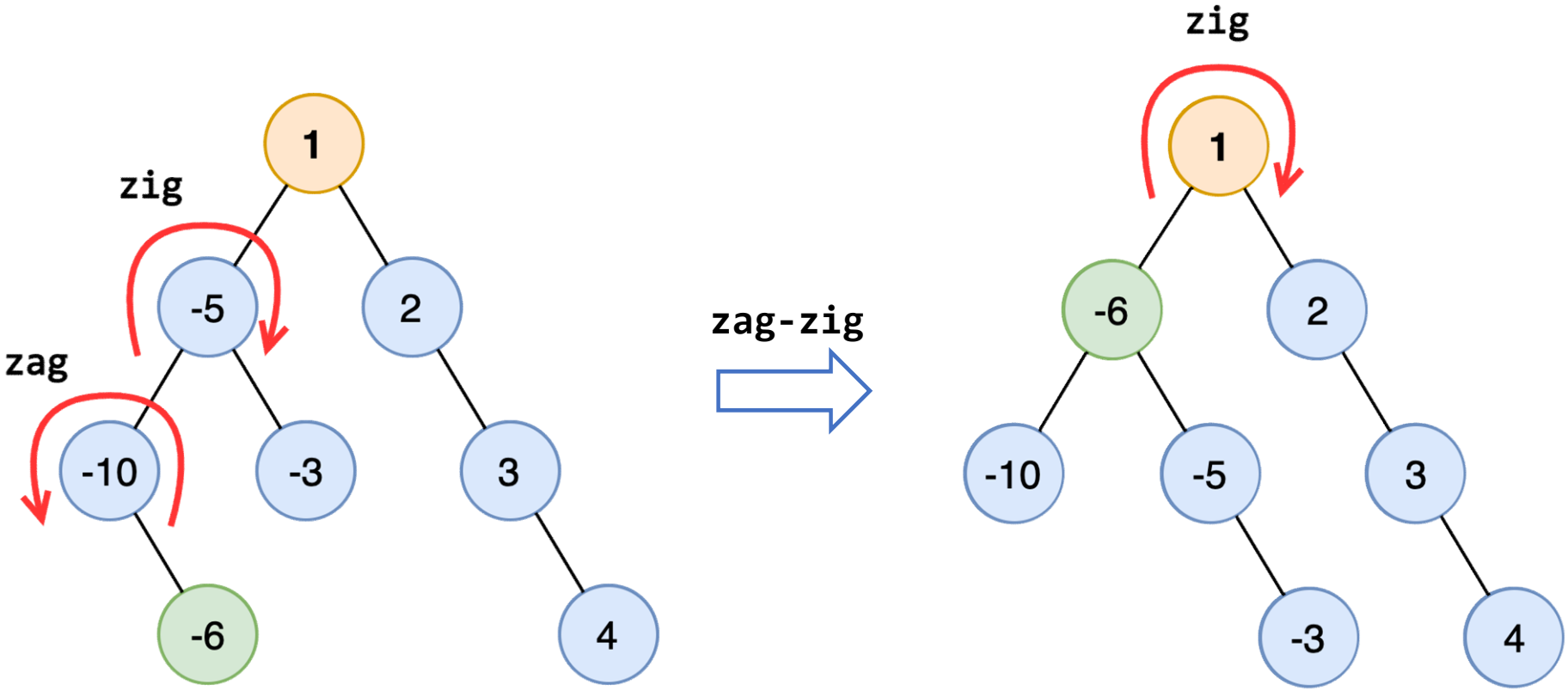
Splay-деревья. Вставка узла



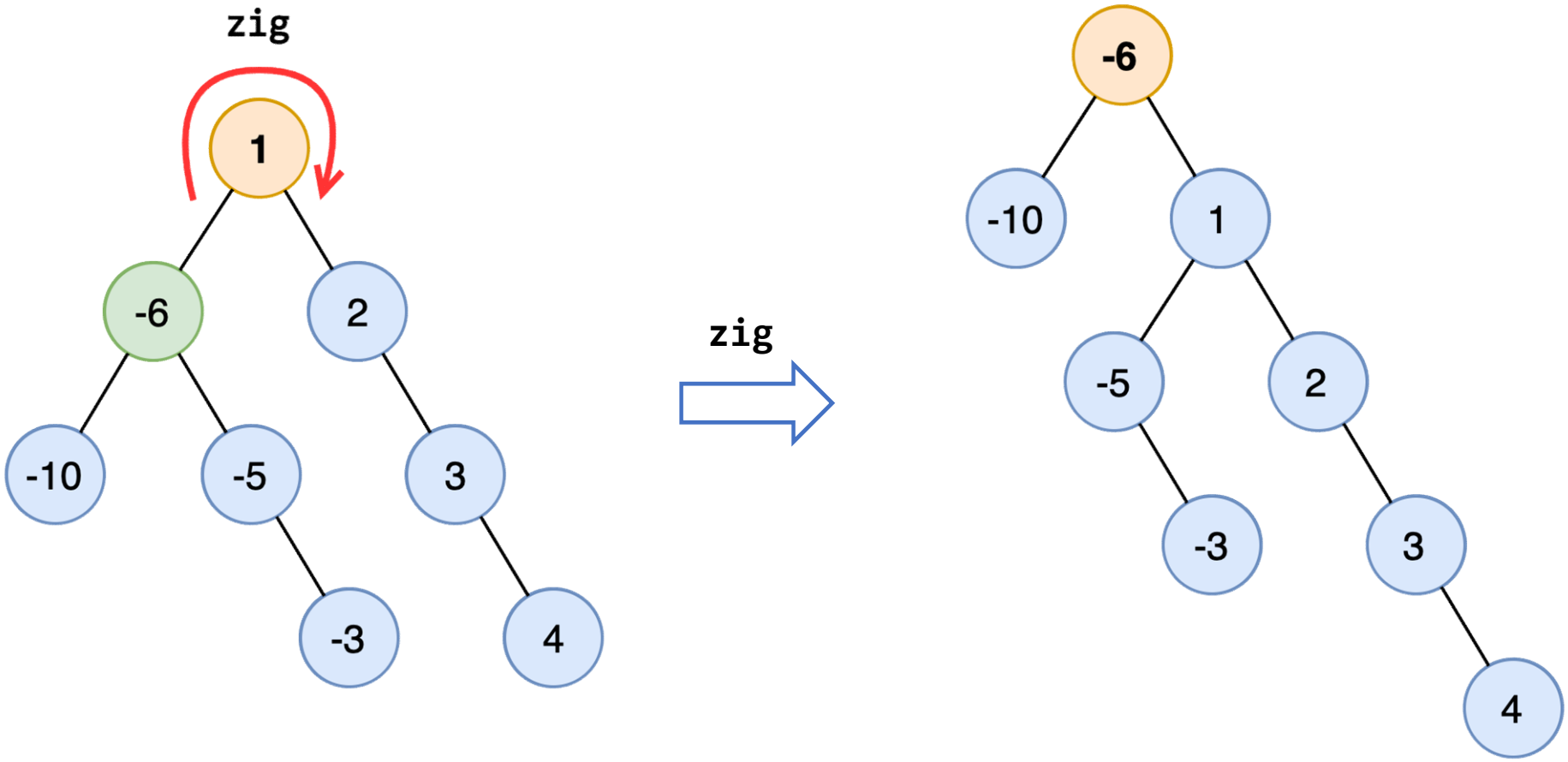
Splay-деревья. Вставка узла



Splay-деревья. Вставка узла



Splay-деревья. Вставка узла



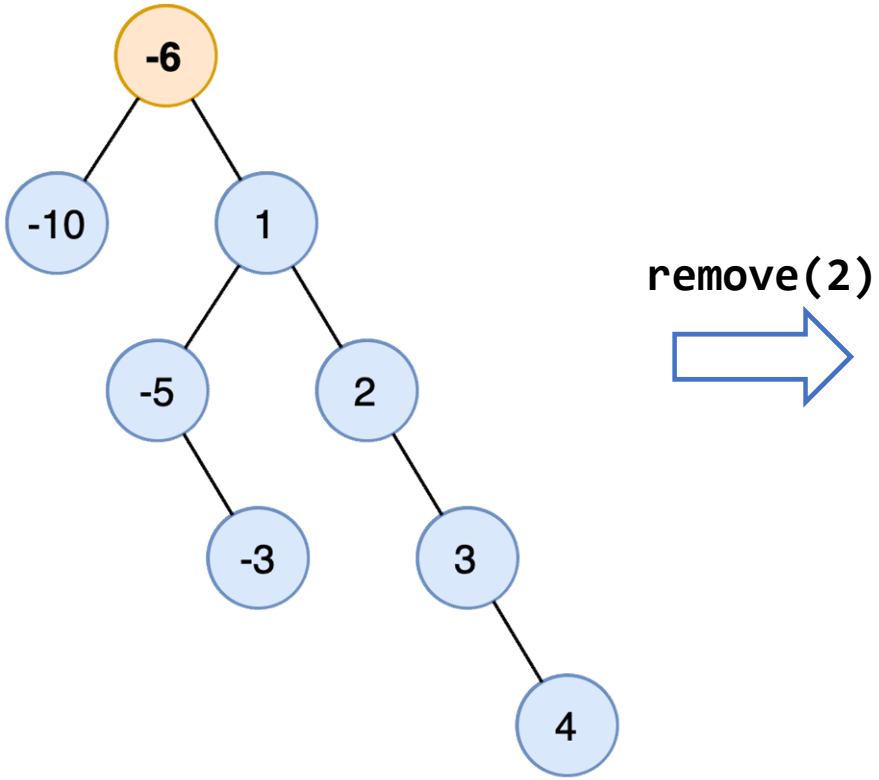
Splay-деревья. Удаление узла

Удаление ключа выполняется с помощью расщепления и слияния:

1. Проталкиваем вершину с удаляемым ключом в корень
2. Физически удаляем корень дерева
3. Ищем наибольший (наименьший) ключ в левом (правом) поддереве и проталкиваем его в корень поддереве
4. Выполняем слияние поддеревьев

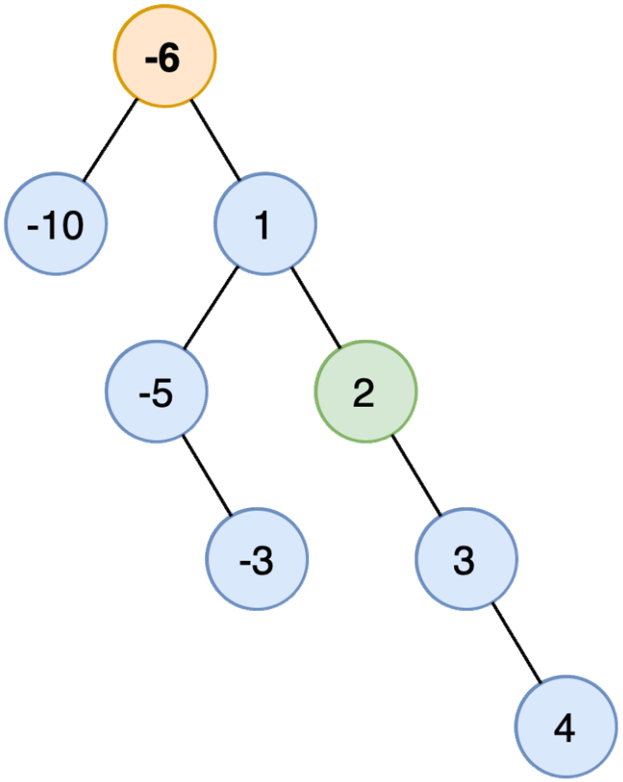
Splay-деревья. Удаление узла

Удаляем узел с ключом **2**.



Splay-деревья. Удаление узла

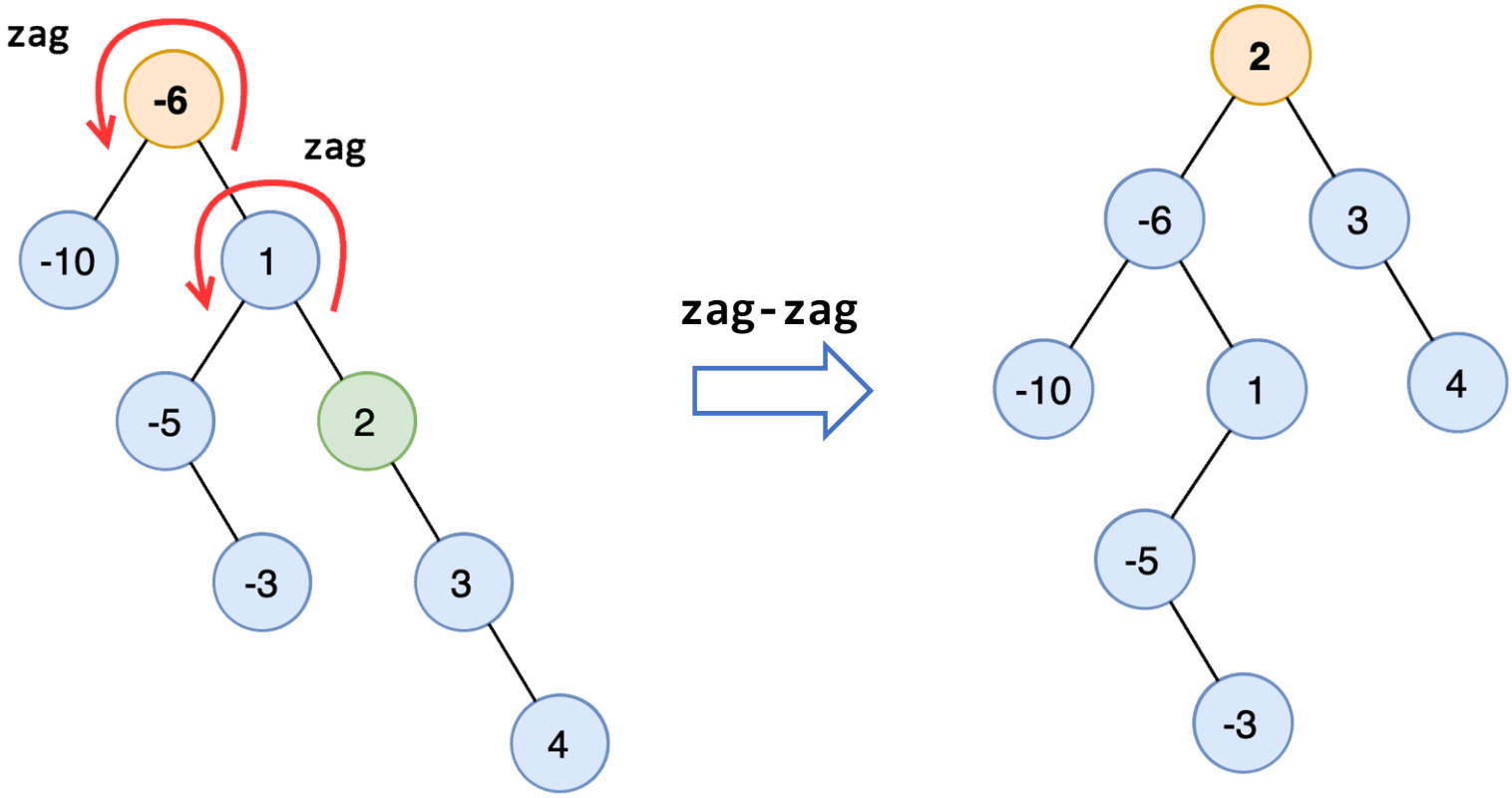
Удаляем узел с ключом **2**.



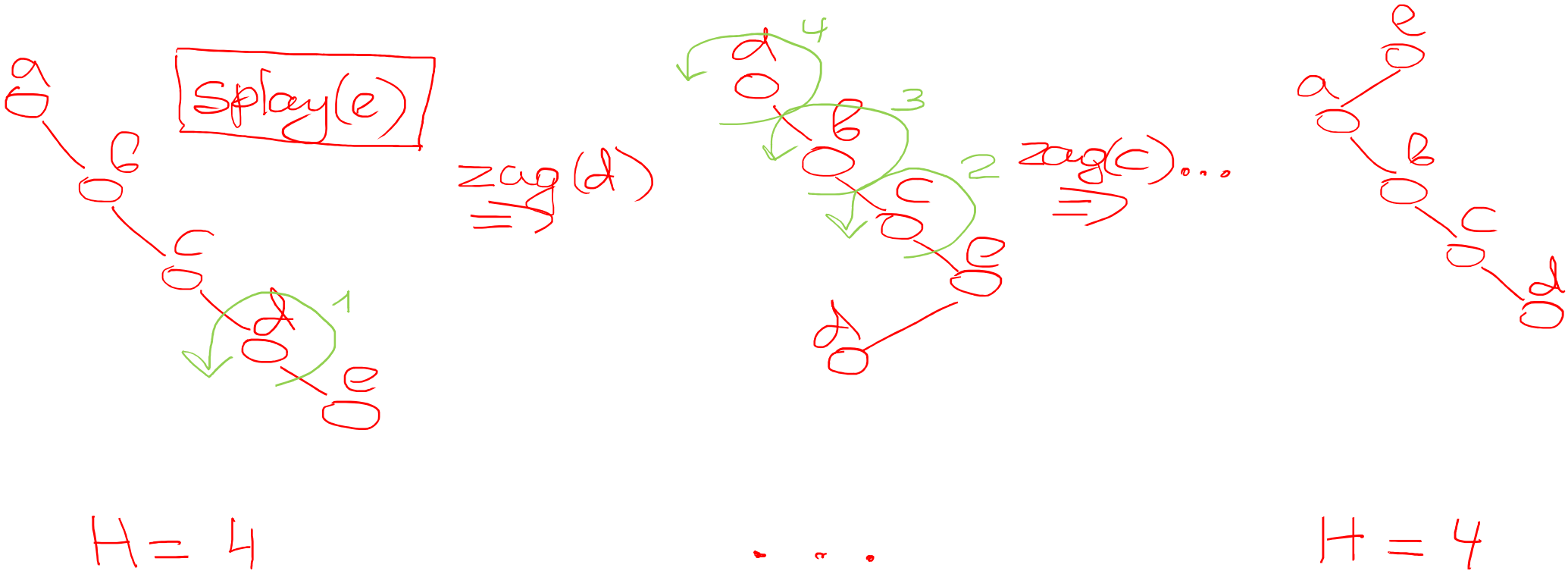
`splay(2)`

Splay-деревья. Удаление узла

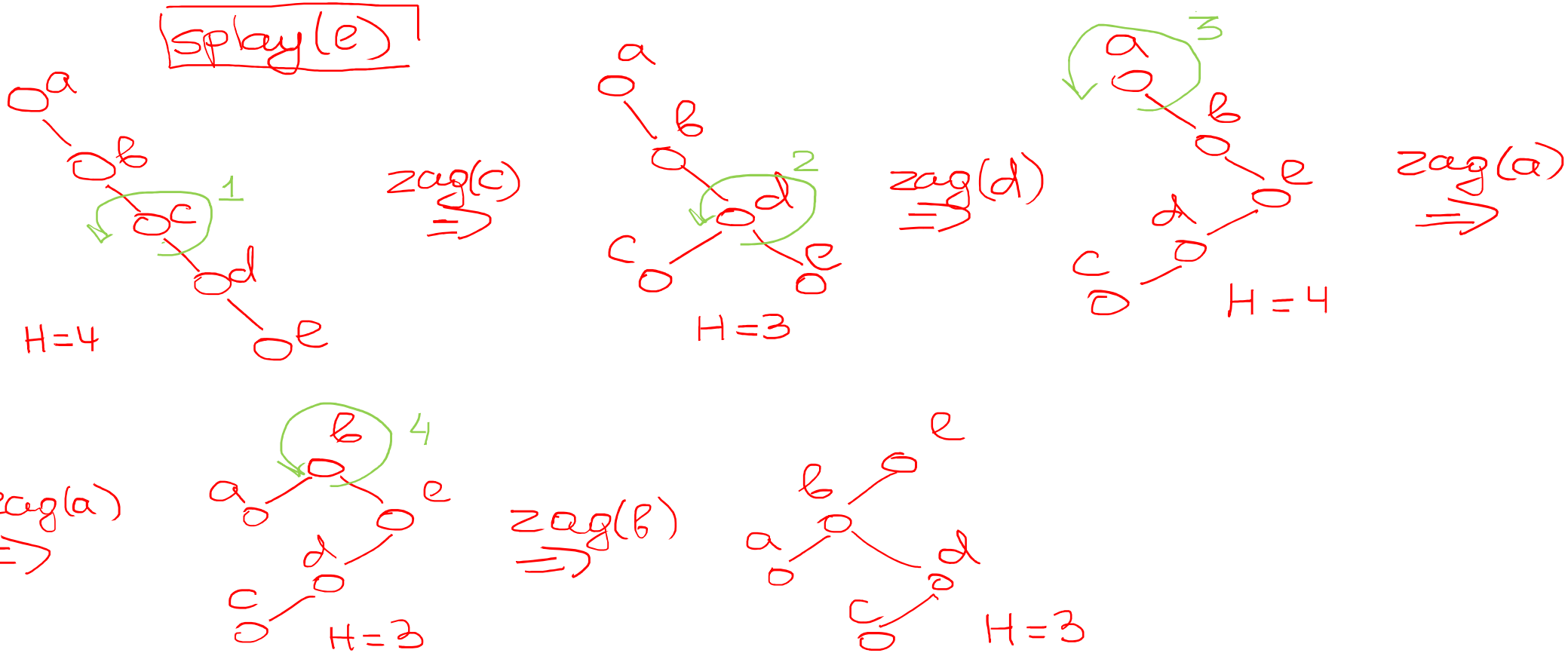
Удаляем узел с ключом **2**.



Splay-деревья. Порядок двойного поворота

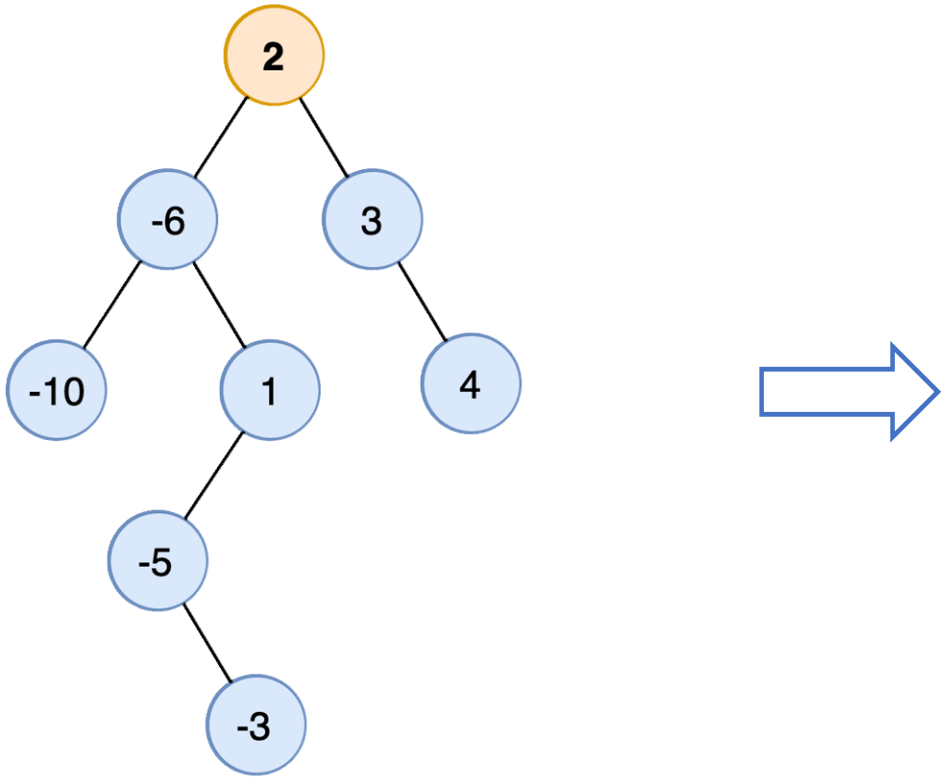


Splay-деревья. Порядок двойного поворота



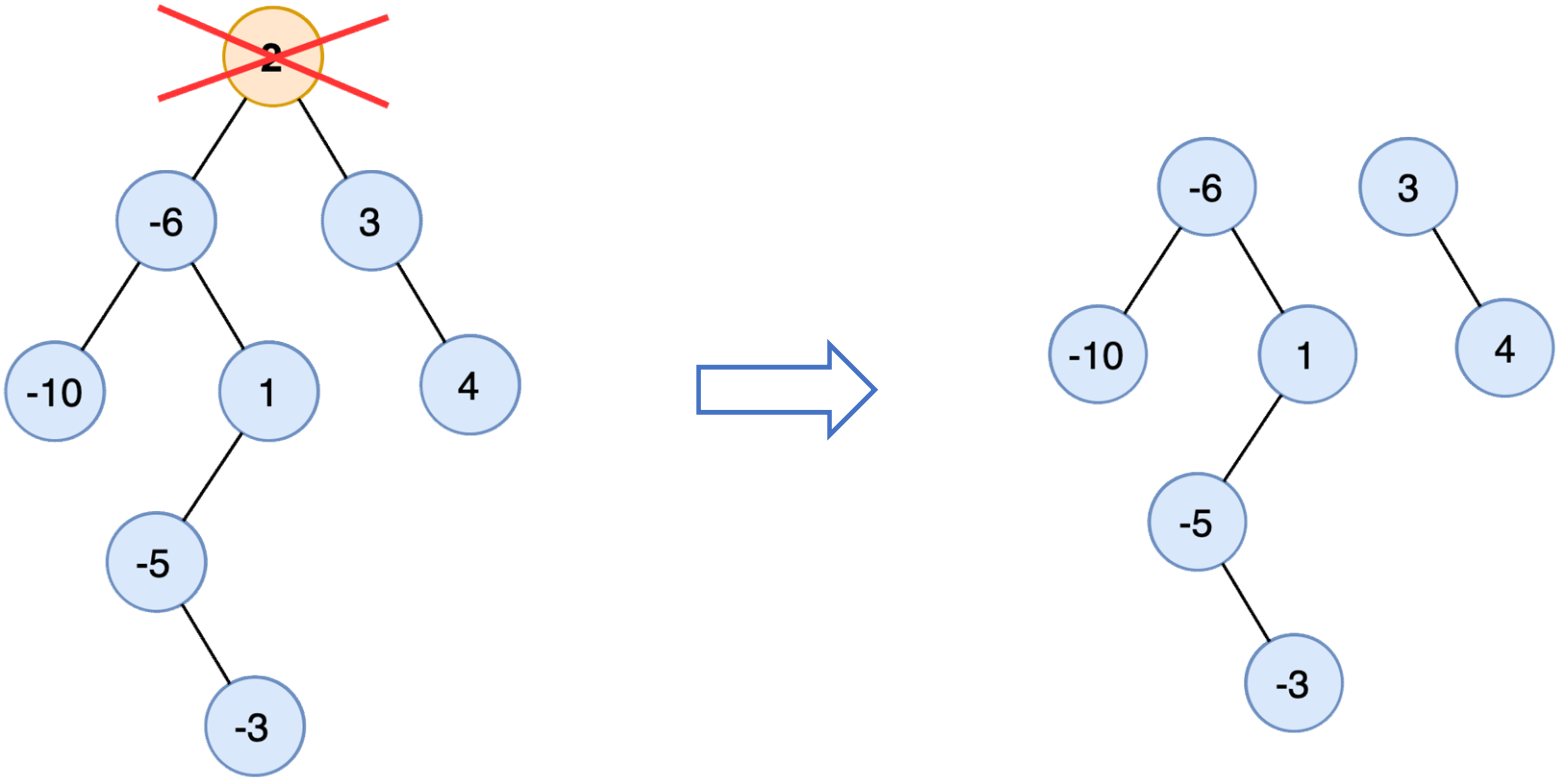
Splay-деревья. Удаление узла

Удаляем узел с ключом **2**.



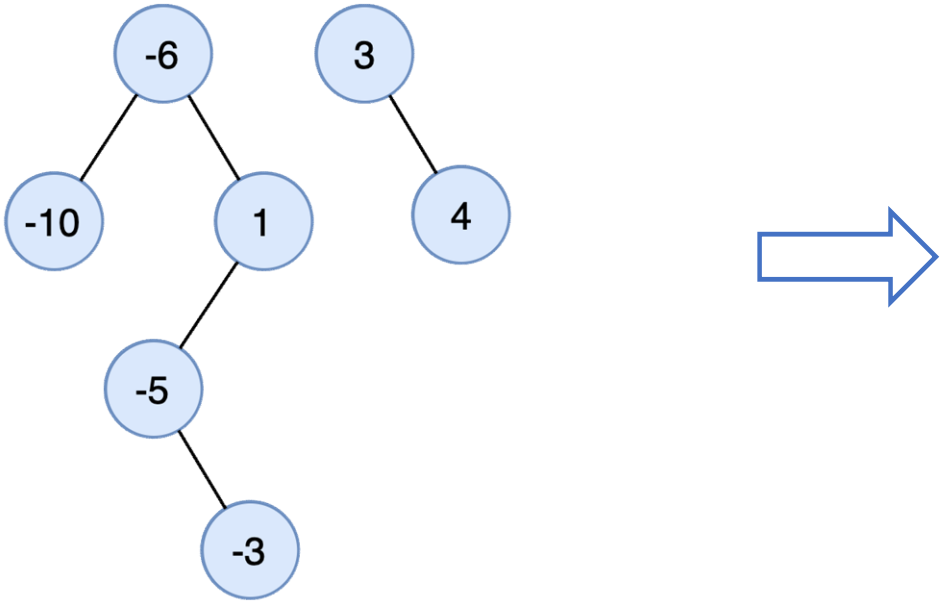
Splay-деревья. Удаление узла

Удаляем узел с ключом **2**.



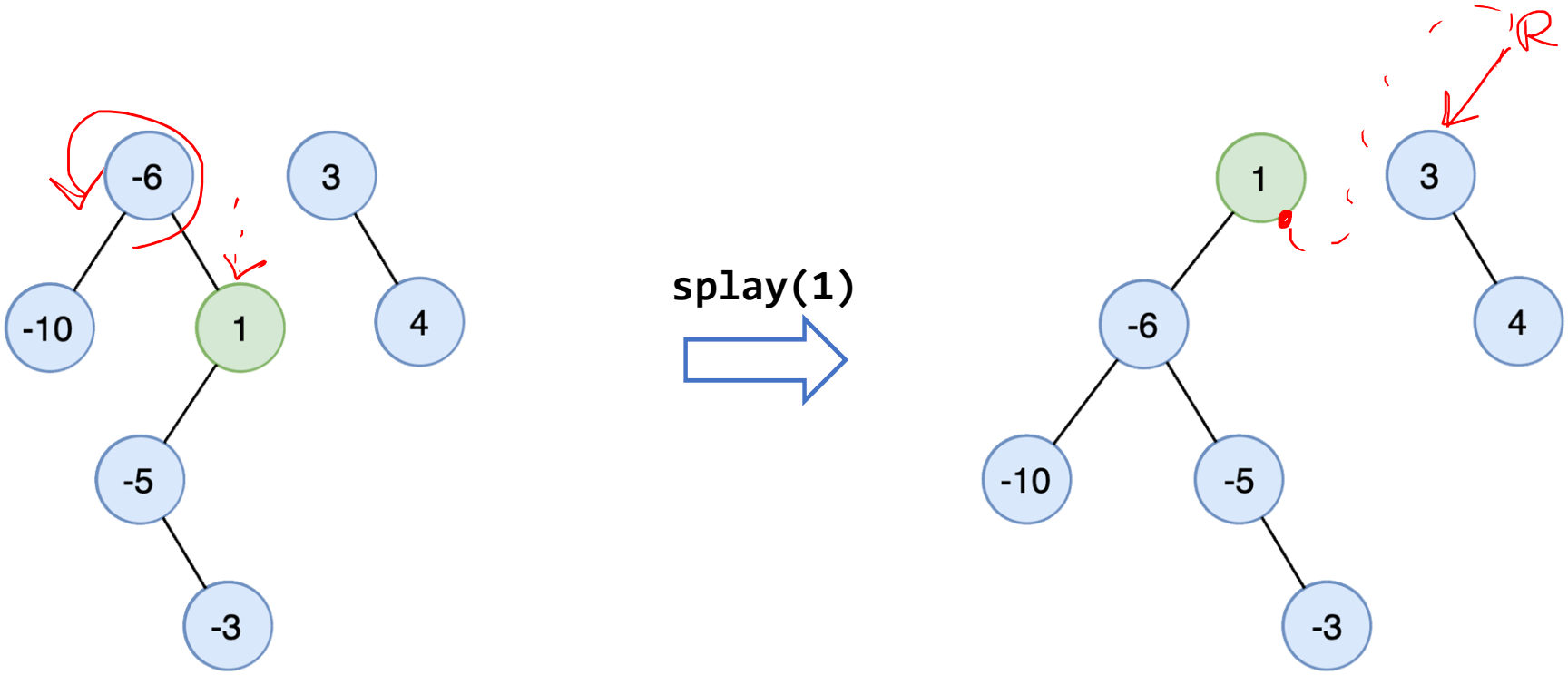
Splay-деревья. Удаление узла

Удаляем узел с ключом **2**.



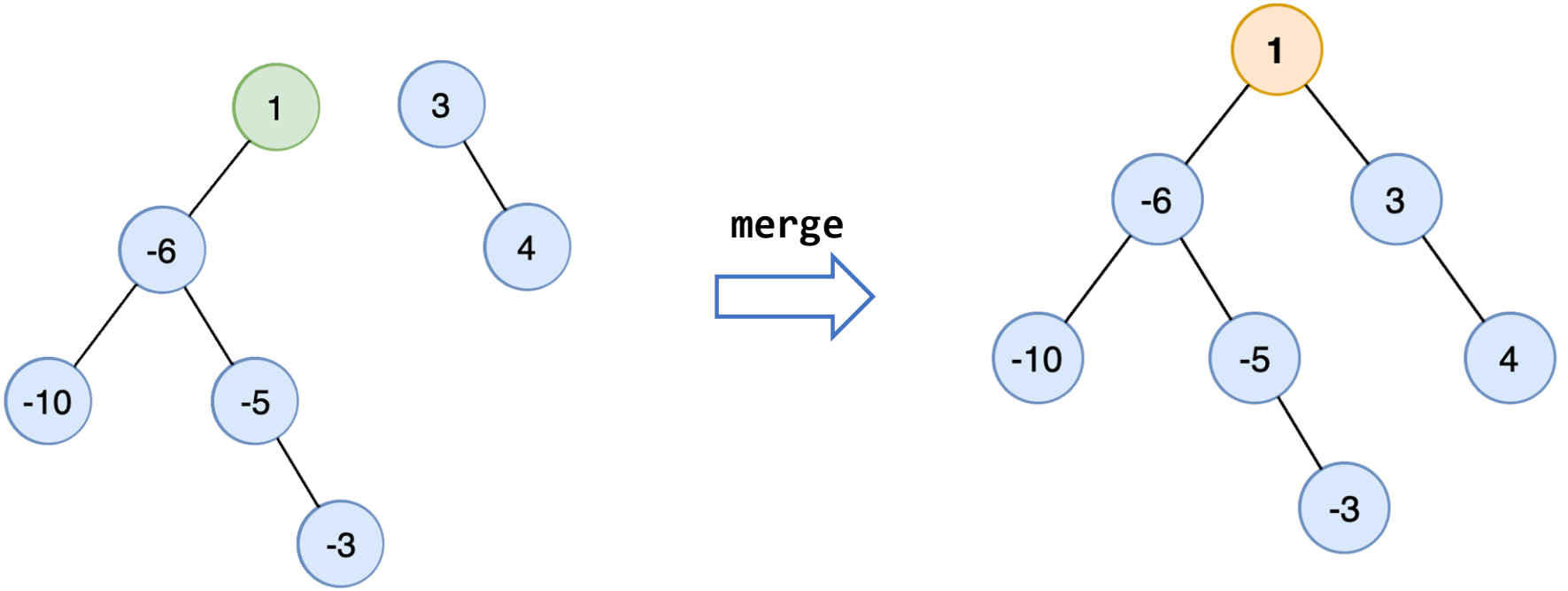
Splay-деревья. Удаление узла

Удаляем узел с ключом **2**.



Splay-деревья. Удаление узла

Удаляем узел с ключом **2**.



Splay-деревья. Замечания о структуре

Для выполнения оптимальных поворотов может потребоваться указатель на родителя.

```
class Node {
    T data;
    Node *left;
    Node *right;
    Node *parent;
    …;
}
```

Splay-деревья. Замечания о структуре

Для выполнения оптимальных поворотов может потребоваться указатель на родителя.

```
class Node {
    T data;
    Node *left;
    Node *right;
    Node *parent;
    …;
}
```

```
void splay (Node *x) {
    …;
    if (x == x->parent->right &&
        x->parent == x->parent->parent->left)
    {
        zag(x->parent);
        zig(x->parent);
    }
    …;
}
```

Splay-деревья. Замечания о структуре

Указатель на родителя можно не хранить, если действовать по обратной схеме (сверху-вниз):

1. Если искомый ключ больше корня, то вращаем дерево влево относительно корня.
2. Если искомый ключ меньше корня, то вращаем дерево вправо относительно корня.

Так, повороты происходят *одновременно* с выполнением операций.

Splay-деревья. Замечания о структуре

Указатель на родителя можно не хранить, если действовать по обратной схеме (сверху-вниз):

Splay-деревья. Замечания о сложности

Пусть **n** – это количество ключей в **splay-**дереве.

В определенные моменты времени выполнение операций в **splay-**дереве может поптребовать **O(n)** времени.

Splay-деревья. Замечания о сложности

Пусть **n** – это количество ключей в **splay-**дереве.

В определенные моменты времени выполнение операций в **splay-**дереве может поптребовать **O(n)** времени.

Амортизированные затраты (среднее время выполнения в худшем случае) каждой операции – **O(log n)**.

B-дерево

B-деревья. Чтение и запись больших блоков

Само-балансирующиеся B-деревья введены в **1970** г.
Р. Байером, Е. МакКрейтом как средство для эффективного представления **больших упорядоченных индексов.**

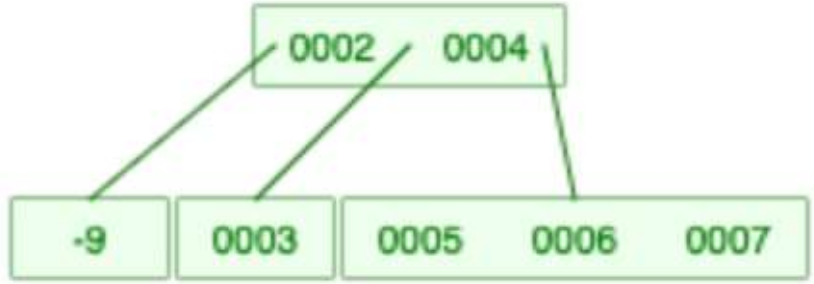
- Поддерживают упорядоченность данных и все стандартные операции
- Узел B-дерева может содержать несколько значений
- Узел B-дерева может иметь несколько детей

B-дерево определяется…

…минимальной степенью **`t>=2`** и набором правил:

1. Каждый узел B-дерева должен иметь **минимум** `t-1` ключей (кроме корня, в котором разрешается меньше).
2. Наибольшее число ключей, которое может содержать в узле, не превосходит `2*t-1`.
3. Ключи в узле B-дерева **отсортированы.**
4. Количество детей узла B-дерева всегда на **1** больше количества ключей, хранящихся в этом узле.
5. Упорядоченное расположение ключей в поддеревьях.
6. Все листья B-дерева находятся на одном уровне.

Узел B-дерева



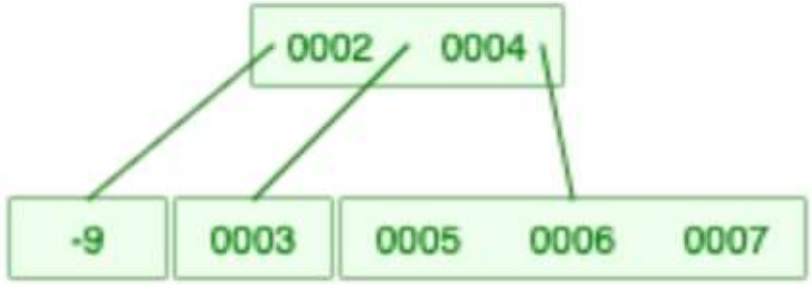

```
class Node {
    T *data;
    Node **childPtrs;

    int t;
    int size;
    bool leaf;
    …;
}
```

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

Вставка ключа в B-дерево

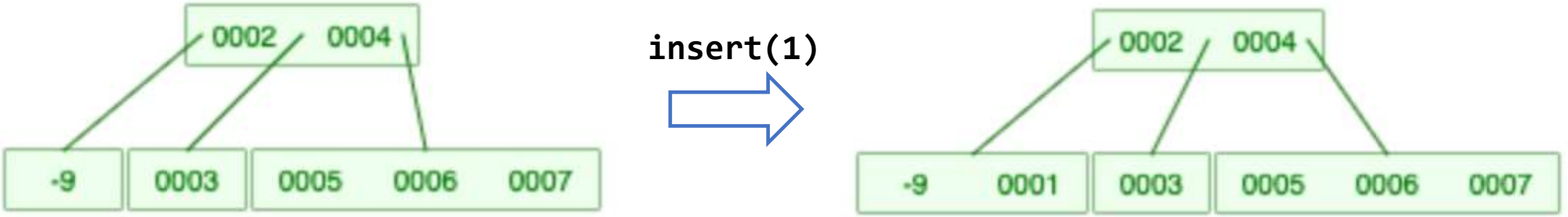
Максимальная емкость узла не достигнута




```
insert(1)
```

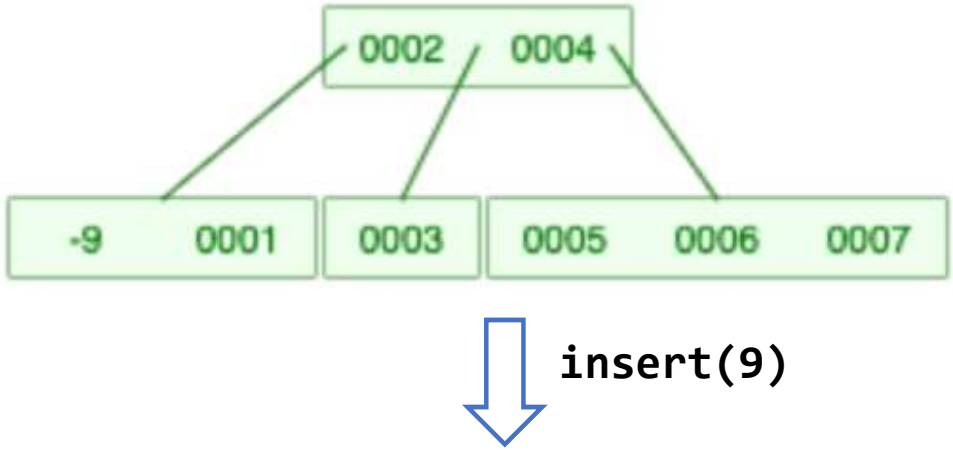
Вставка ключа в B-дерево

Максимальная емкость узла не достигнута



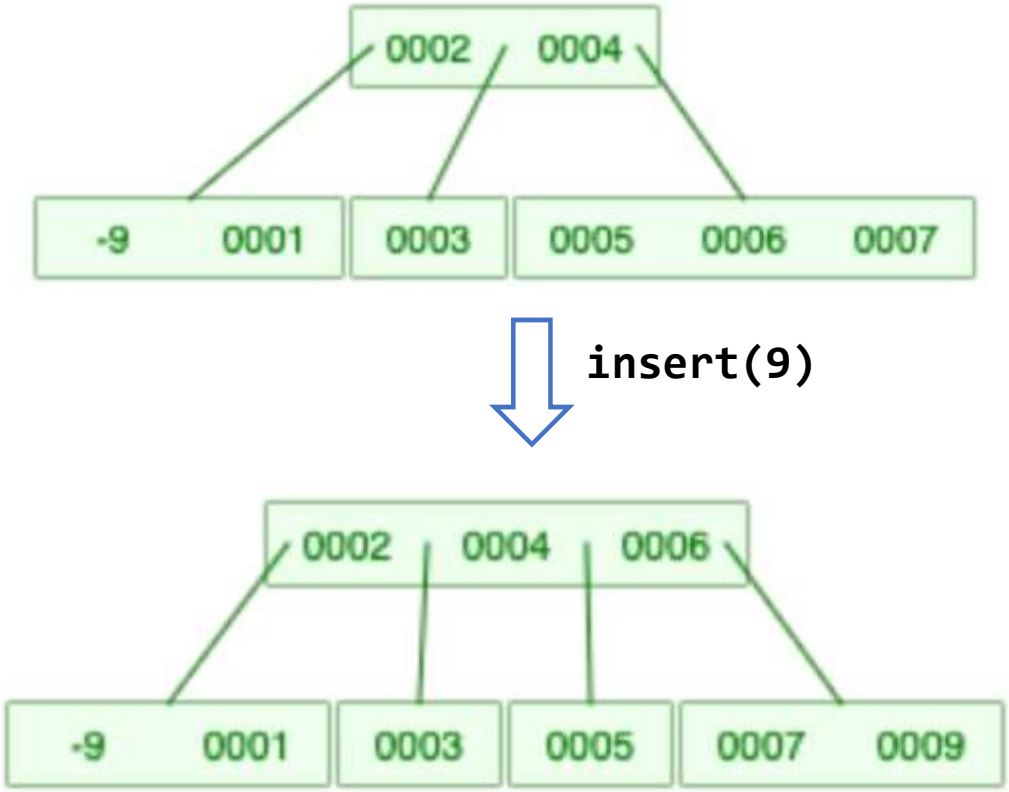
Вставка ключа в B-дерево

Максимальная емкость узла достигнута. Вариант **1**



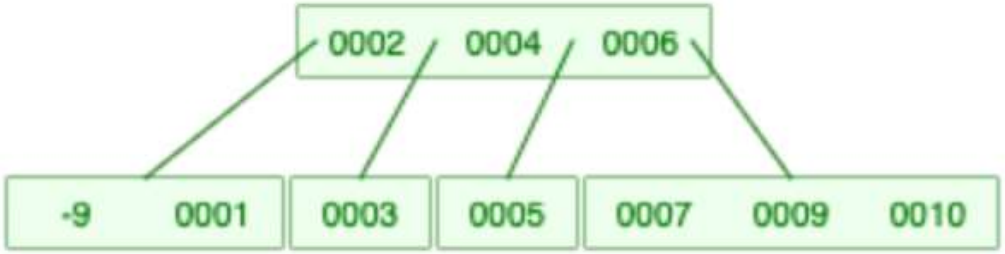
Вставка ключа в B-дерево

Максимальная емкость узла достигнута. Вариант **1**



Вставка ключа в B-дерево

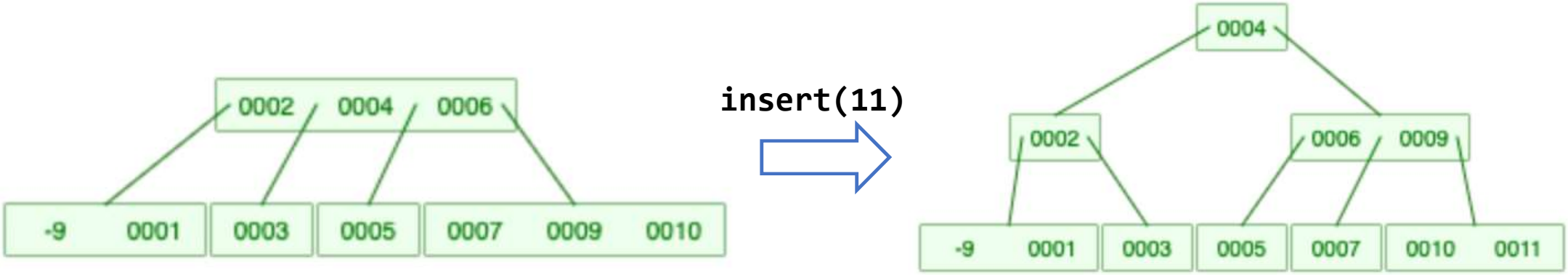
Максимальная емкость узла достигнута. Вариант **2**



`insert(11)`

Вставка ключа в B-дерево

Максимальная емкость узла достигнута. Вариант **2**



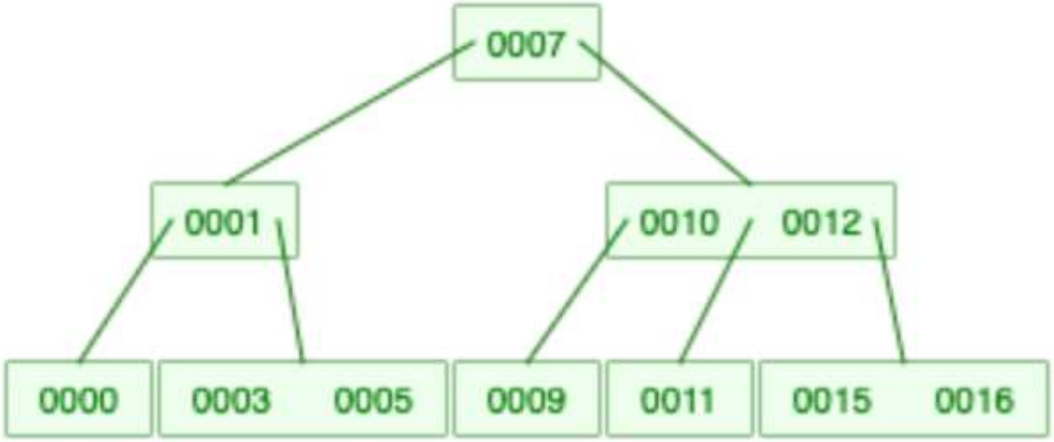
Вставка ключа в B-дерево

Дерево в результате вставки растет **вверх.**

1. Поиск подходящего узла для вставки ключа.
2. Проверка текущего размера узла на заполнение.
   - Предел не достигнут **->** запись ключа в узел
   - Предел достигнут **–>** расщепление узла по «среднему» значению, которое выталкивается наверх.

Удаление ключа из B-дерева

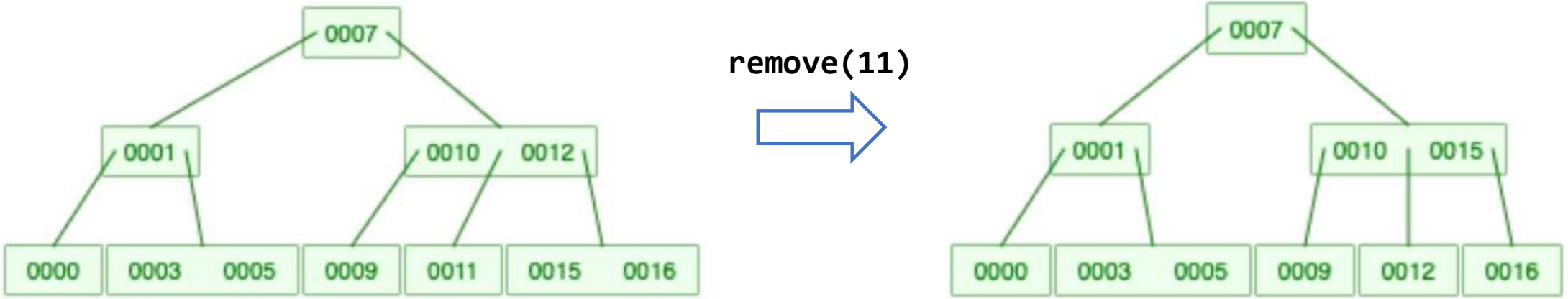
Сопровождается проверкой на достижение минимально допустимого количества ключей и детей.



`remove(11)`

Удаление ключа из B-дерева

Сопровождается проверкой на достижение минимально допустимого количества ключей и детей.



B^+-дерево

B⁺-дерево. Особенности

Проблема хранения самих данных в узлах обсуждалась с самого начала.

B⁺-деревья хранят указатели на данные только **в листовых узлах,** которые образуют развернутый список, тогда остальные вершины образуют индекс над данными.

B⁺-дерево определяется…

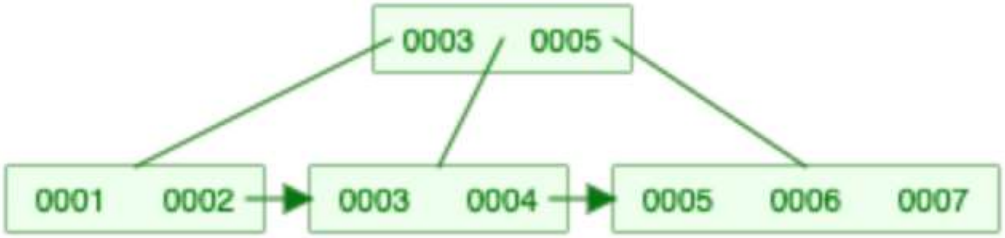
…максимальной емкостью узла **M** и следующими правилами:

1. Все листья расположены на одном уровне
2. Корень дерева имеет минимум двух детей
3. Каждый узел (кроме корня) имеет не больше **M+1** ребенка, но не меньше **(M+1) / 2**
4. Каждый узел (кроме корня) имеет не больше **M** ключей, но не меньше **M/2**

Вставка ключа в B^+-дерево

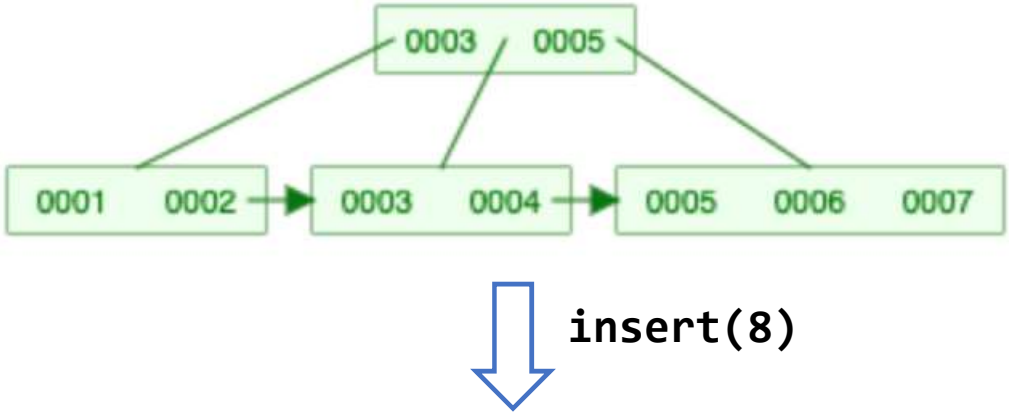
Находим подходящий лист на последнем уровне.

1. Если максимальная емкость узла **не достигнута**, вставляем в найденный лист, сохраняя порядок.
2. Если максимальная емкость узла **достигнута**, выполняем его расщепление и передаем «среднее» значение наверх.



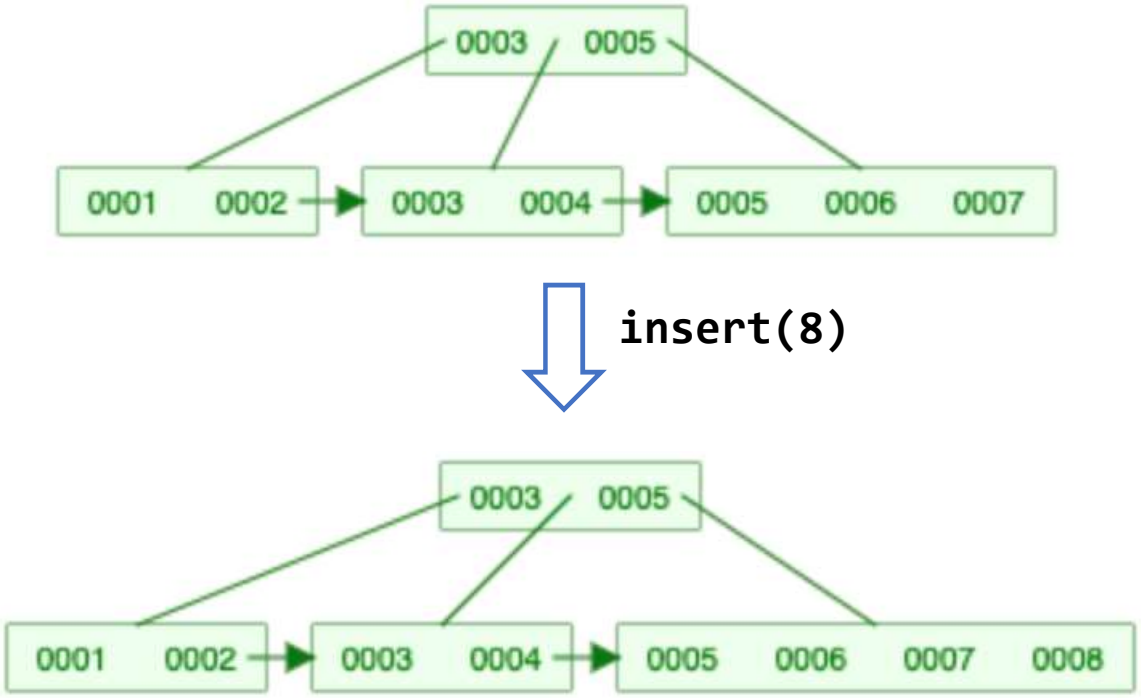
Вставка ключа в B⁺-дерево с M=4

Вставка в лист, в котором еще есть свободное место.



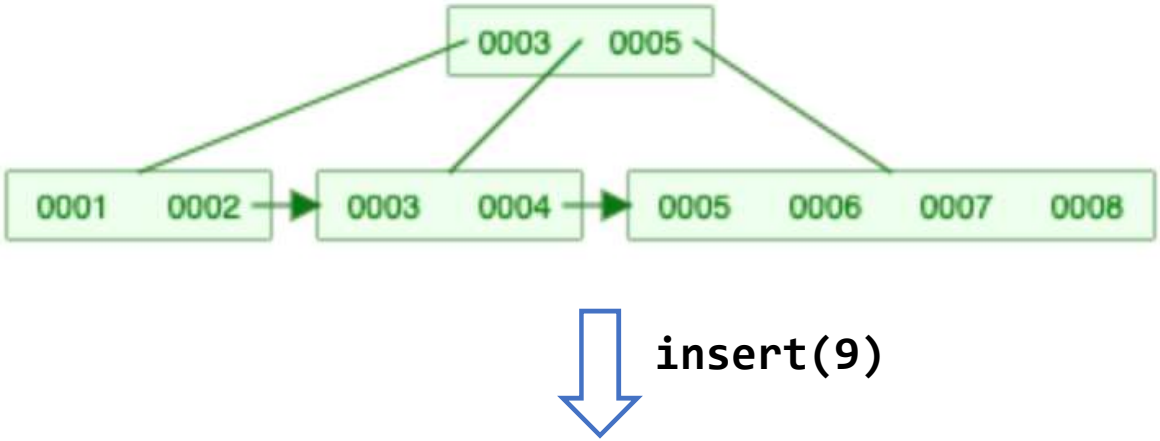
Вставка ключа в B⁺-дерево с M=4

Вставка в лист, в котором еще есть свободное место.



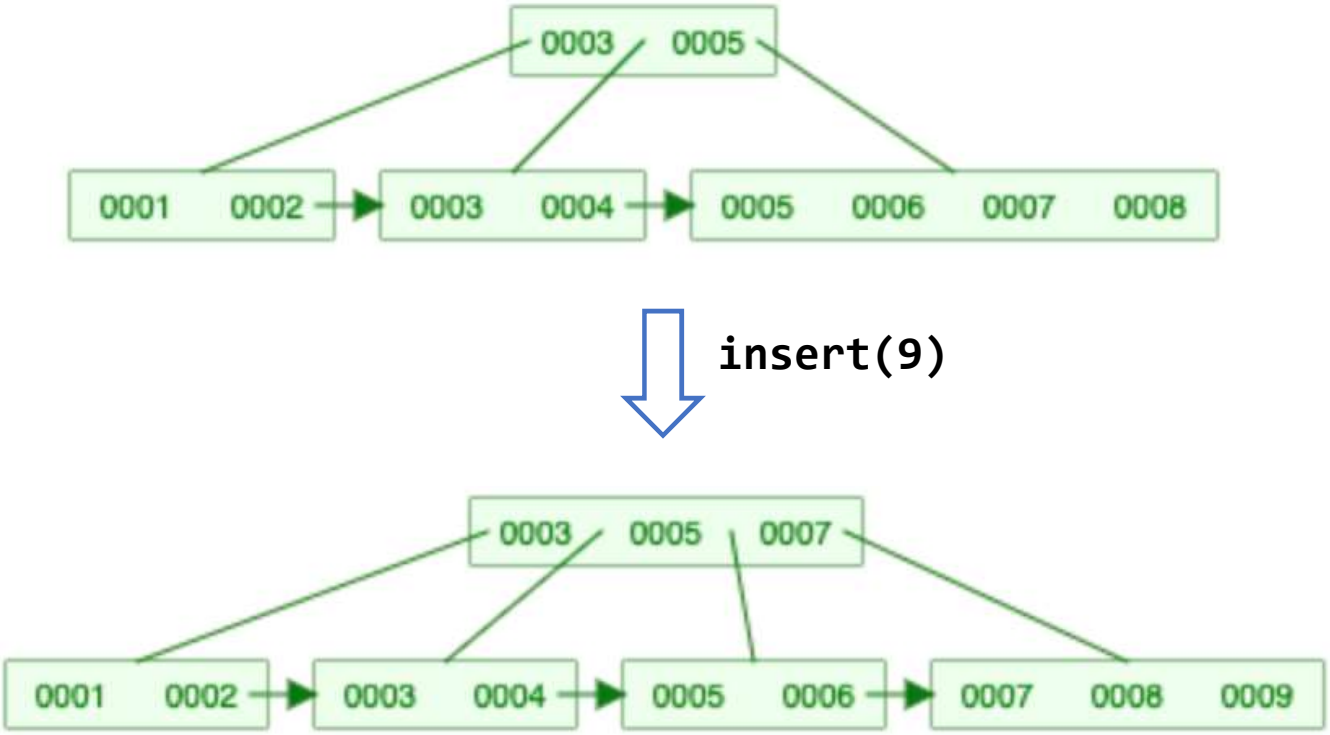
Вставка ключа в B⁺-дерево с M=4

Вставка в лист, в котором уже нет свободного места.



Вставка ключа в B⁺-дерево с M=4

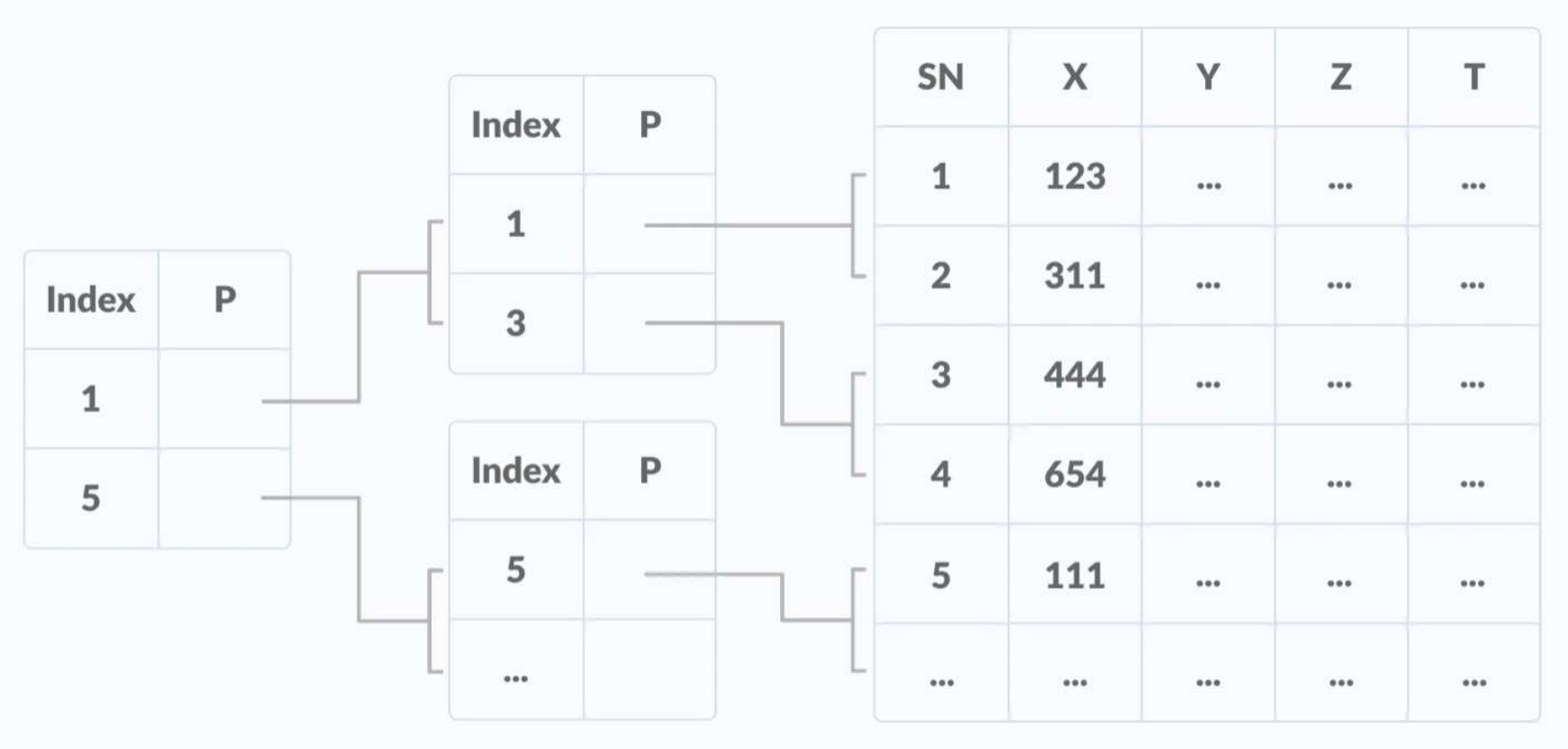
Вставка в лист, в котором уже нет свободного места.



B⁺-дерево. Применение

- Индексация данных в файловых системах
  NTFS, APFS и др.
- Индексация данных в реляционных и нереляционных СУБД
  Microsoft SQL Server, SQLite, Oracle и др.

B⁺-дерево. Многоуровневый индекс



Другие деревья...

Деревья разные важны…

- Рандомизированное бинарное дерево поиска

- Двоичная куча (пирамида) **1964 г.**
- Декартово дерево **(дер**евопир**амида, ку**чаде**рево) 1989 г.**

- Префиксное/суффиксное дерево
- …

Полезные ссылки

- https://www.cs.usfca.edu/~galles/visualization/BTree.html
- https://habr.com/ru/company/otus/blog/459216/
- https://neerc.ifmo.ru/wiki/index.php?title=B-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE
- https://neerc.ifmo.ru/wiki/index.php?title=B%2B-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE