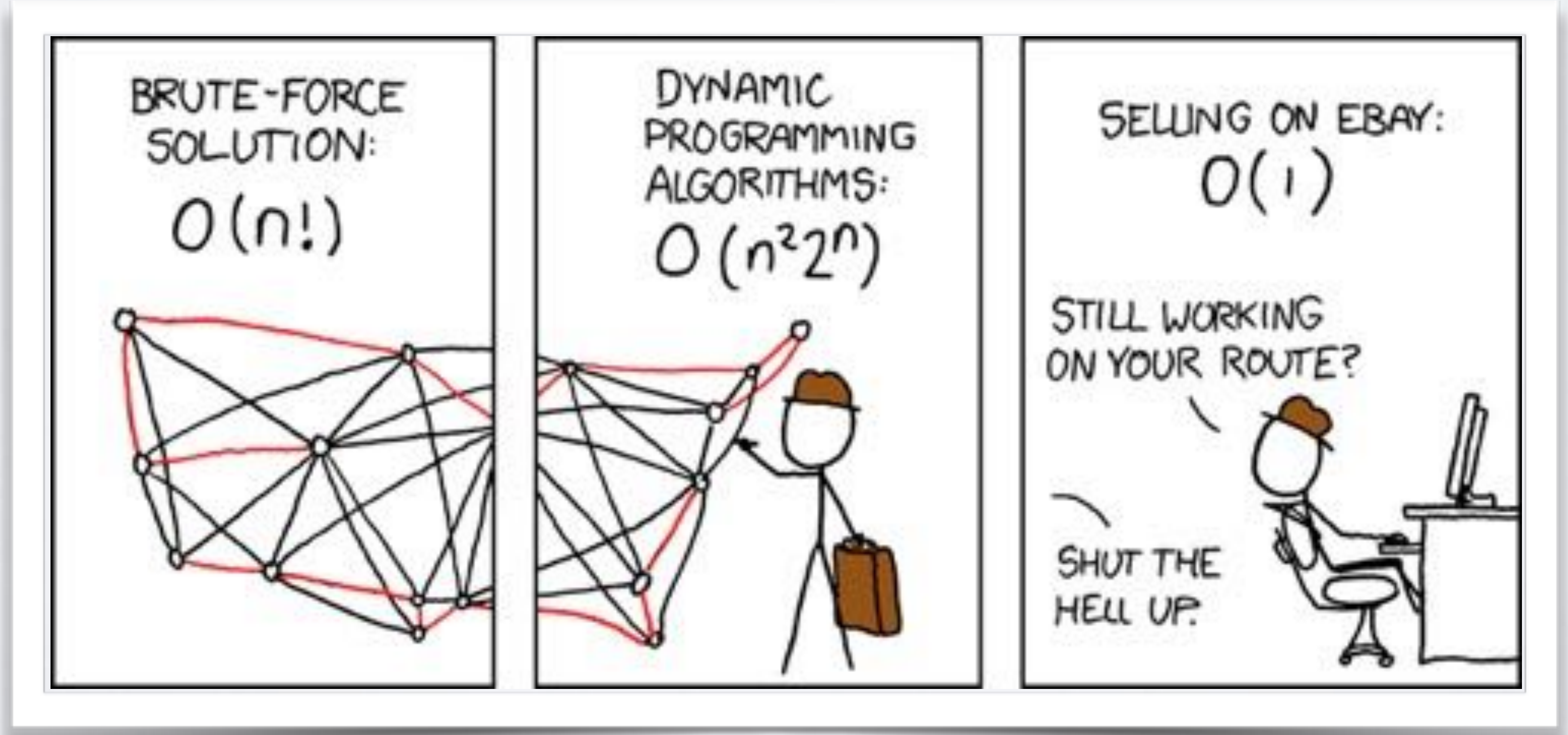
ОСНОВЫ ПРОГРАММНОГО КОНСТРУИРОВАНИЯ

Лекция № 8

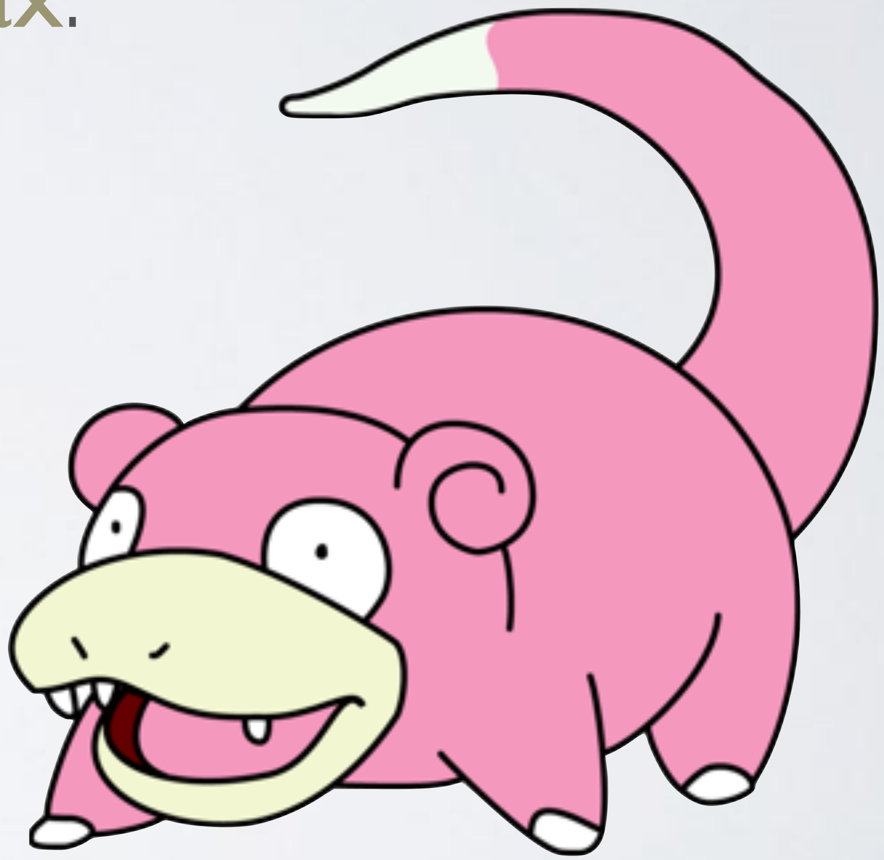
24 октября 2016 г.



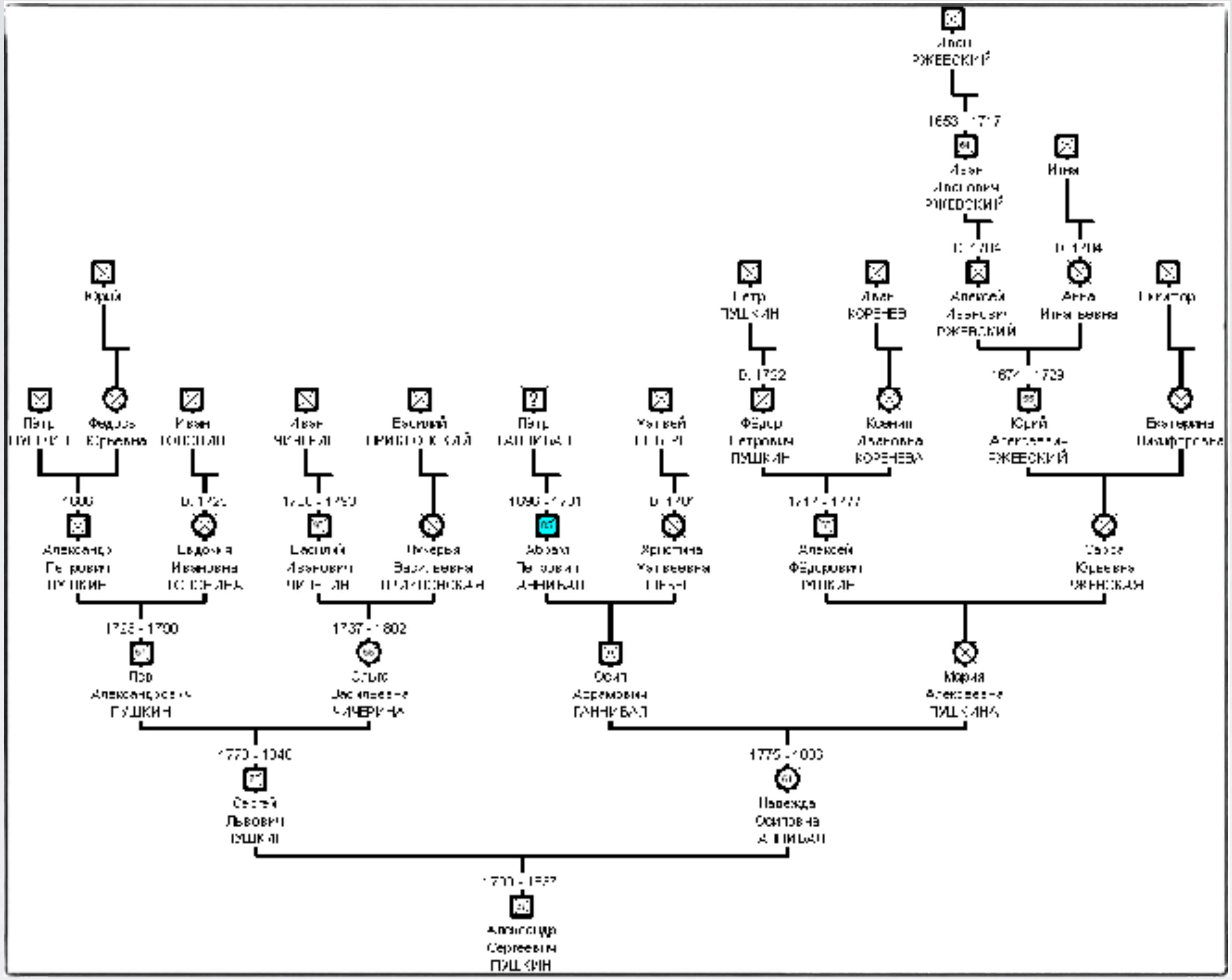
ЗНАКОМЫЕ СТРУКТУРЫ ДАННЫХ

Операции в **массивах** и **связных списках**:

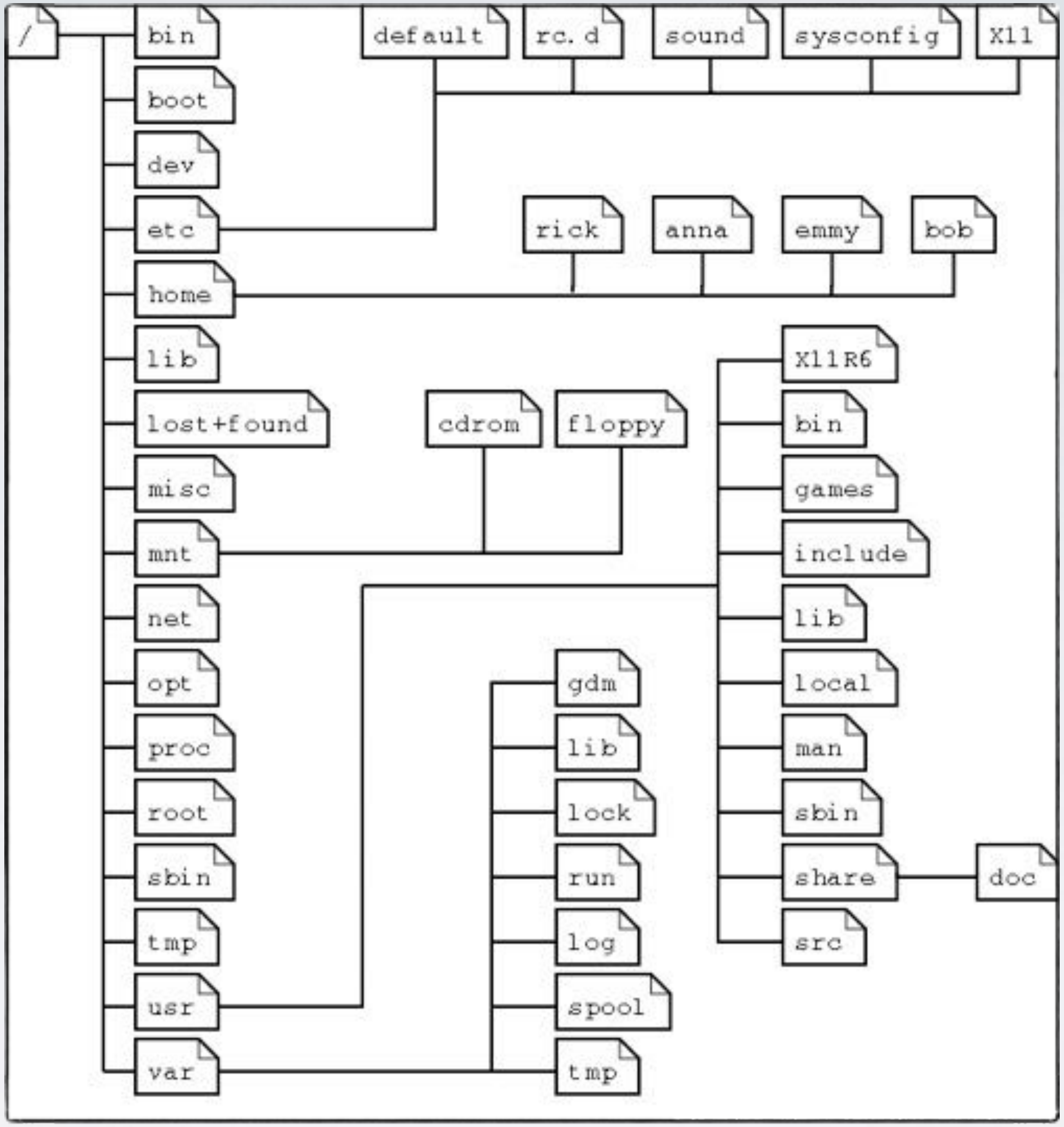
- Вставка: O(N) и O(1).
- Удаление: O(N) и O(1).
- Доступ по индексу: O(1) и O(N).
- Поиск по значению: O(N) и O(N).



ДАЛЕЕ: НАЙТИ ОБЩЕЕ



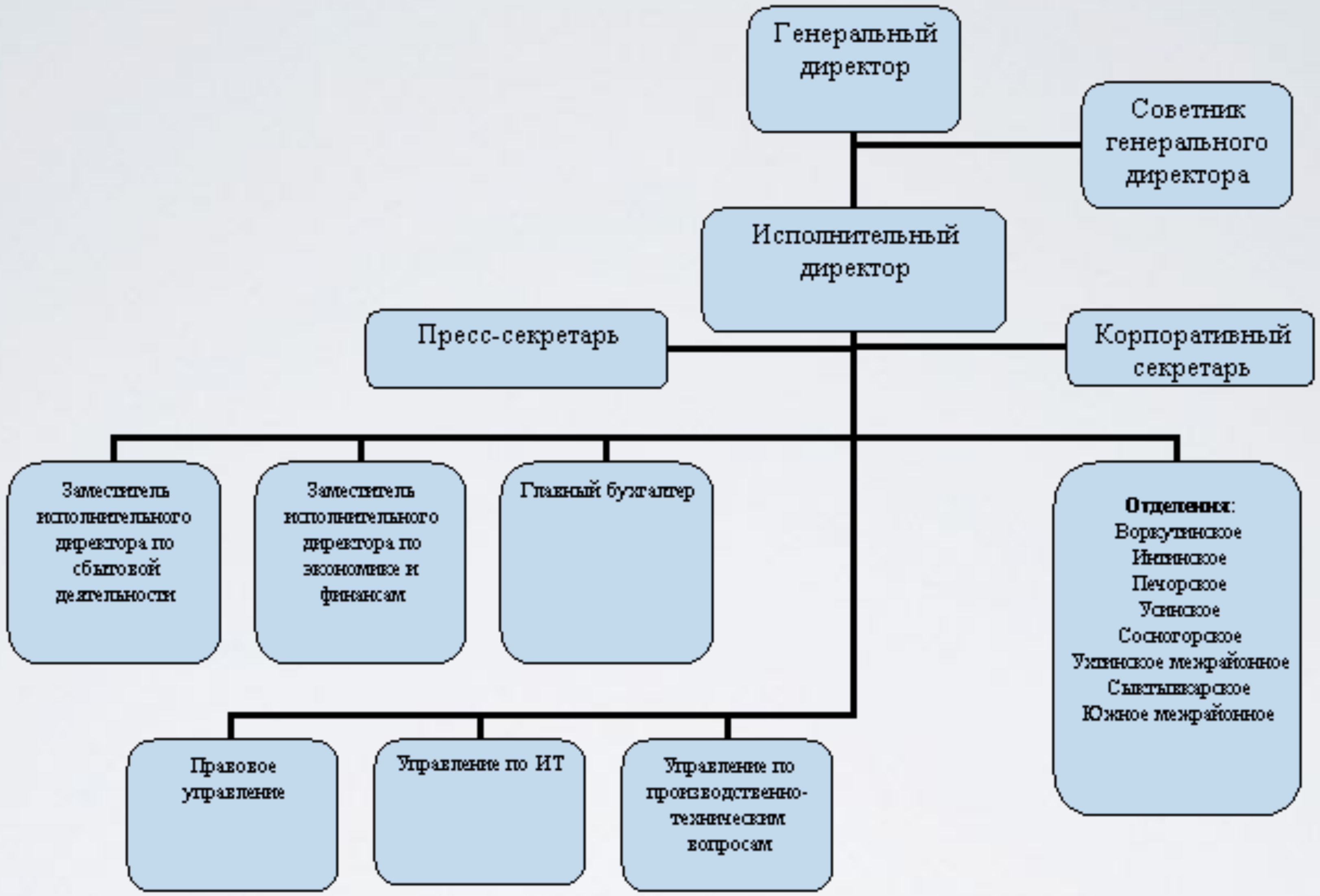
Генеалогическое дерево



Файлы и каталоги

```xml
<client>
  <address>
    <street>5401 Julio Ave.</street>
    <city>San Jose</city>
    <state>CA</state>
    <zip>95116</zip>
  </address>
  <phone>
    <work>4084630000</work>
    <home>4081111111</home>
    <cell>4082222222</cell>
  </phone>
  <fax>4087776666</fax>
  <email>love2shop@yahoo.com</email>
</client>
```

XML-документ



Организационная структура предприятия

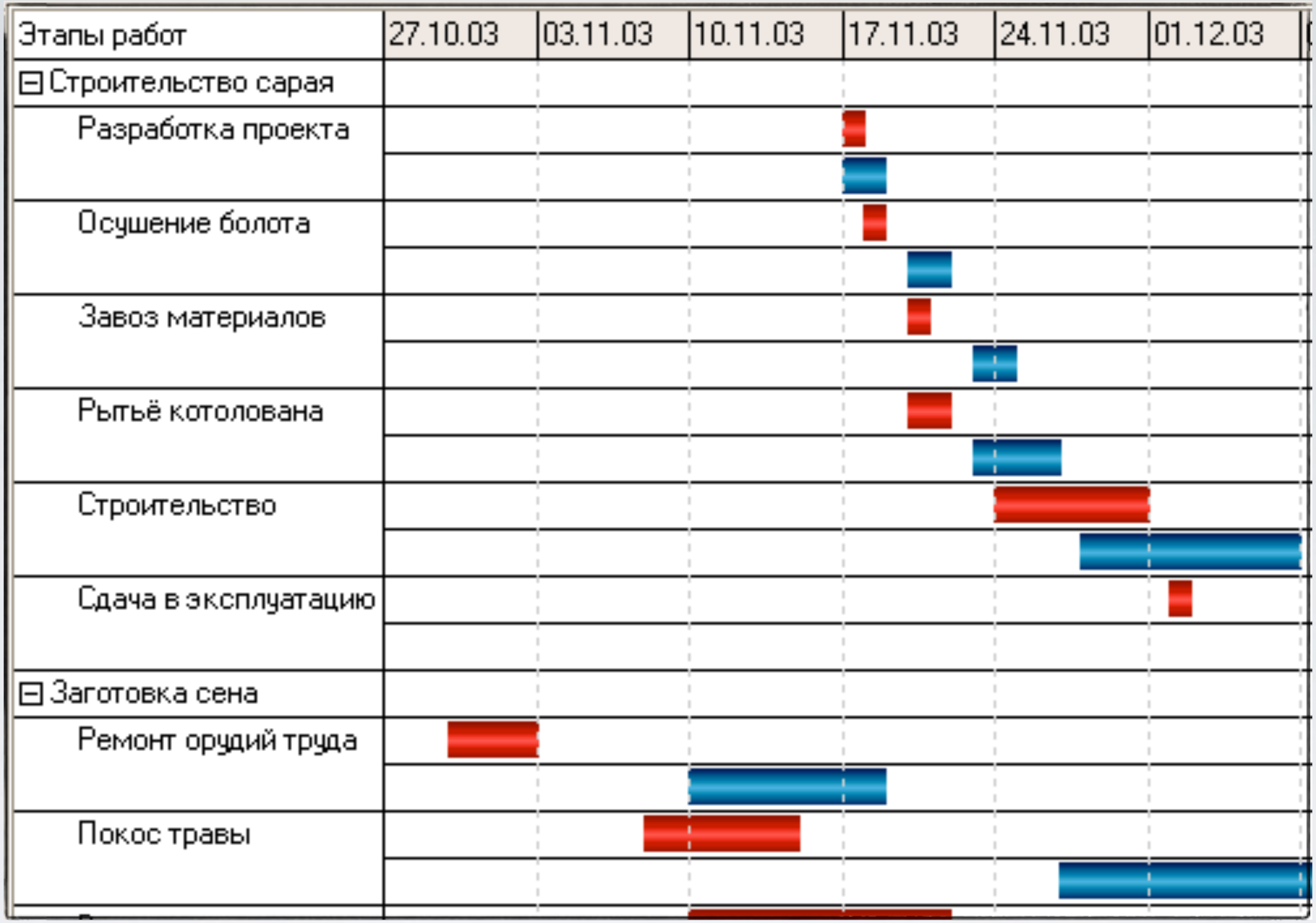
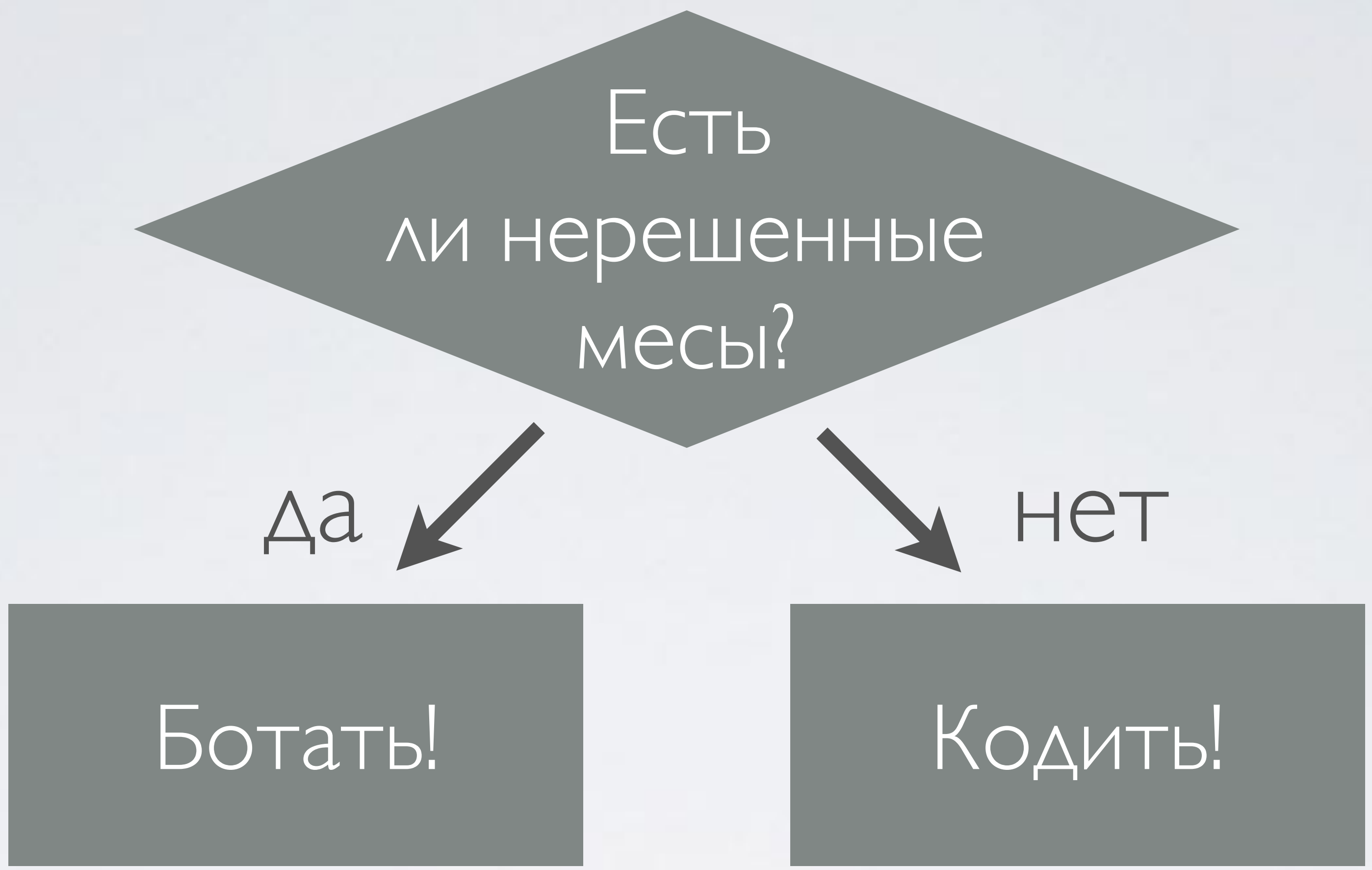


Диаграмма Ганта



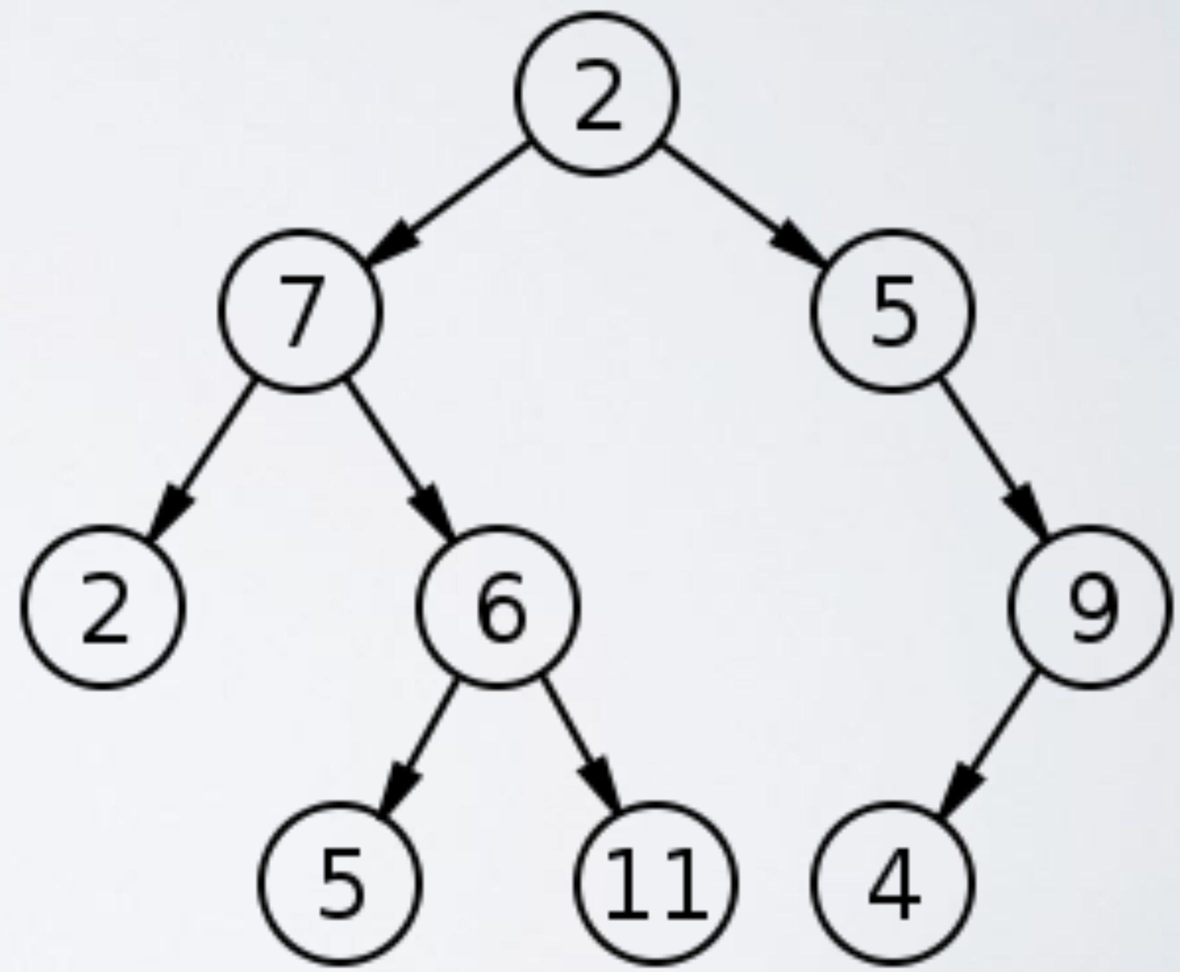
Дерево решений

СТРУКТУРА ДАННЫХ: ДЕРЕВО

- Состоит из элементов (узлов).
- Имеет корень.
- Все остальные узлы, кроме корня, распределены по непересекающимся подмножествам — поддеревьям.

ДРЕВОВЕДЕНИЕ

- **Корень** (2).
- **Внутренние узлы** (2, 7, 5, 6, 9) и **листья** (2, 5, 11, 4).
- **Родитель** (7 для 2 и 6; 9 для 4; 2 для 5 и 7) и **потомки** (дочерние узлы).
- **Сестринские узлы** (5 для 7; 2 для 6; 11 для 5).



ИНТЕРФЕЙС ДЕРЕВА

- Вставка узла.
- Удаление узла.
- Обход дерева (посещение всех узлов).
- Переходы (от потомка к родителю, от сестринского узла к другому сестринскому и т.д.)

ДЕРЕВЬЯ В C

```
struct TreeNode {
    struct TreeNode *parent;
    struct TreeNode **children;
    int nchildren;
    void *data;
};
```

Динамический массив
указателей на дочерние узлы

БИНАРНОЕ ДЕРЕВО

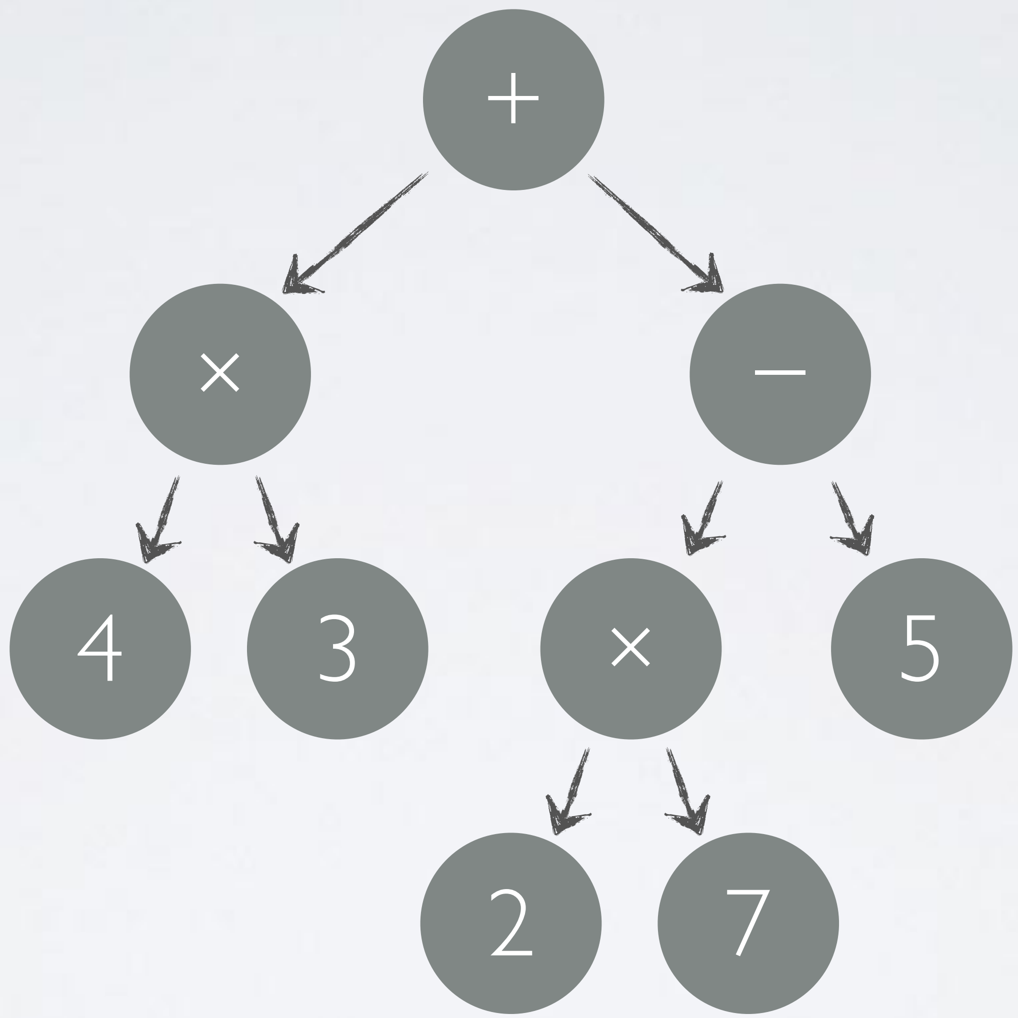
- У каждого узла максимум два потомка: **левый** и **правый**.
- Может быть так, что правый потомок присутствует, а левый — нет.
- Допустимо пустое двоичное дерево.

ДВОИЧНЫЕ ДЕРЕВЬЯ В С

```
struct TreeNode {
    struct TreeNode *parent;
    struct TreeNode *left, *right;
    void *data;
};
```

Популярность двоичных деревьев связана с удобством представления и работы с ними

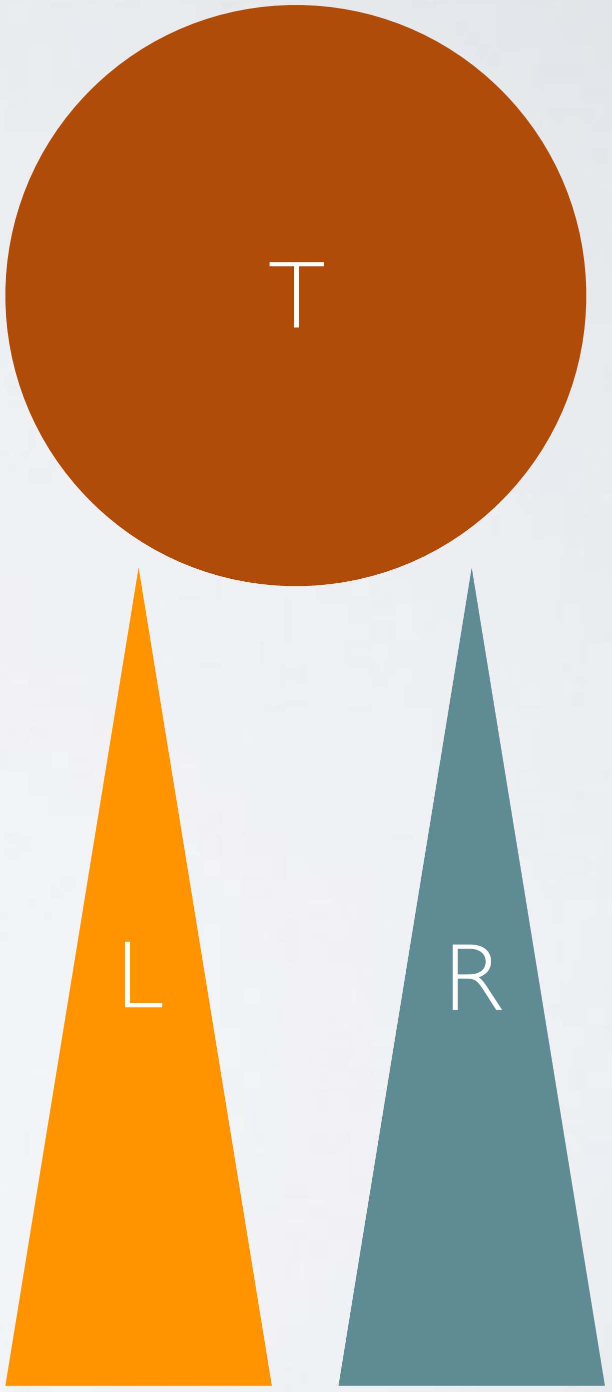
АРИФМЕТИЧЕСКОЕ ВЫРАЖЕНИЕ



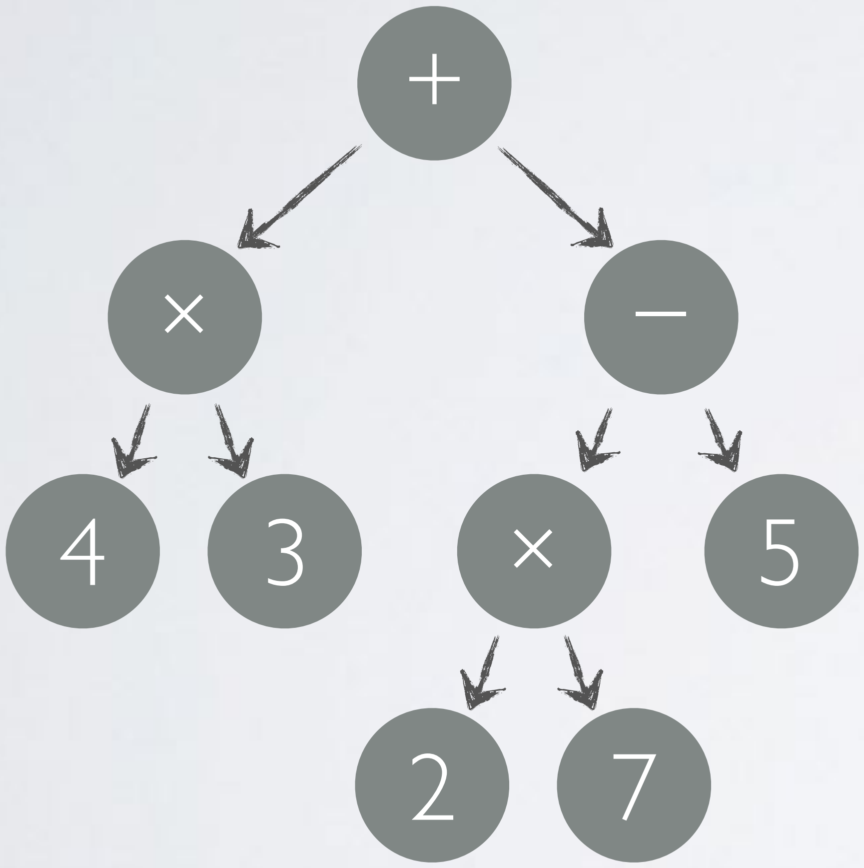
$4 \times 3 + (2 \times 7 - 5)$

ОБХОДЫ ДВОИЧНОГО ДЕРЕВА

- Сверху вниз: **T**, **L**, **R**.
- Слева направо: **L**, **T**, **R**.
- Снизу вверх: **L**, **R**, **T**.



ОБХОДЫ ДЕРЕВА ВЫРАЖЕНИЯ



Сверху вниз: префиксная запись

+ × 4 3 − × 2 7 5

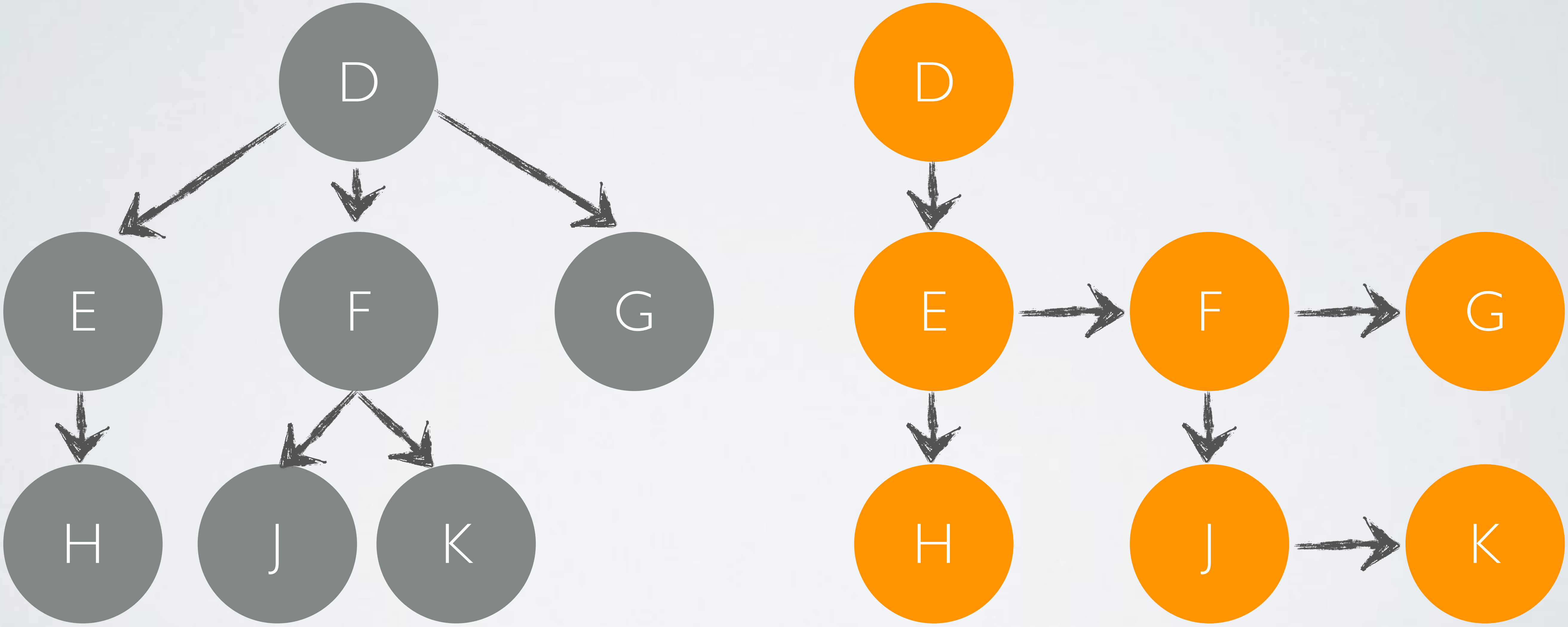
Слева направо: инфиксная запись

4 × 3 + 2 × 7 − 5

Снизу вверх: постфиксная запись

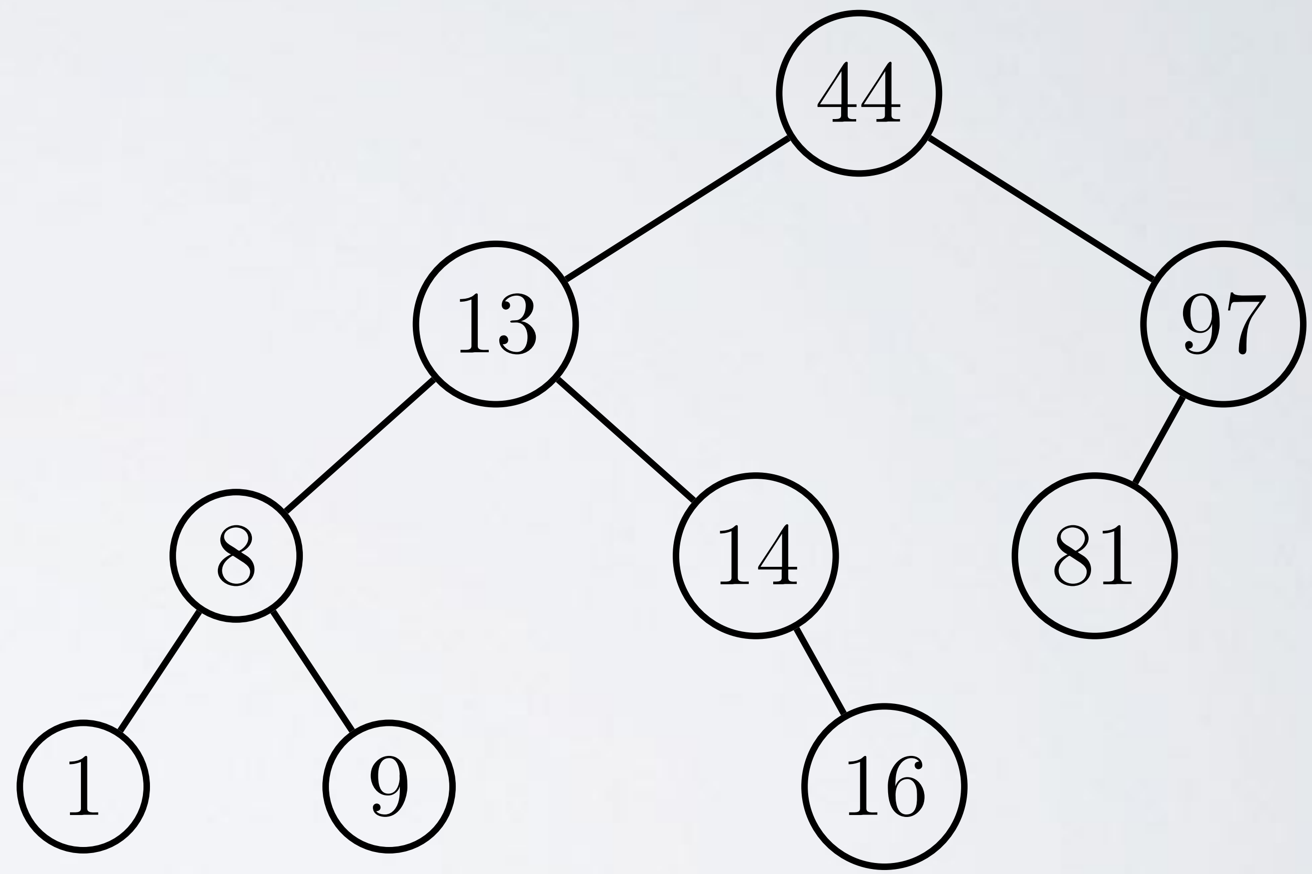
4 3 × 2 7 × 5 − +

ПРЕОБРАЗОВАНИЕ ЛЮБОГО ДЕРЕВА В ДВОИЧНОЕ



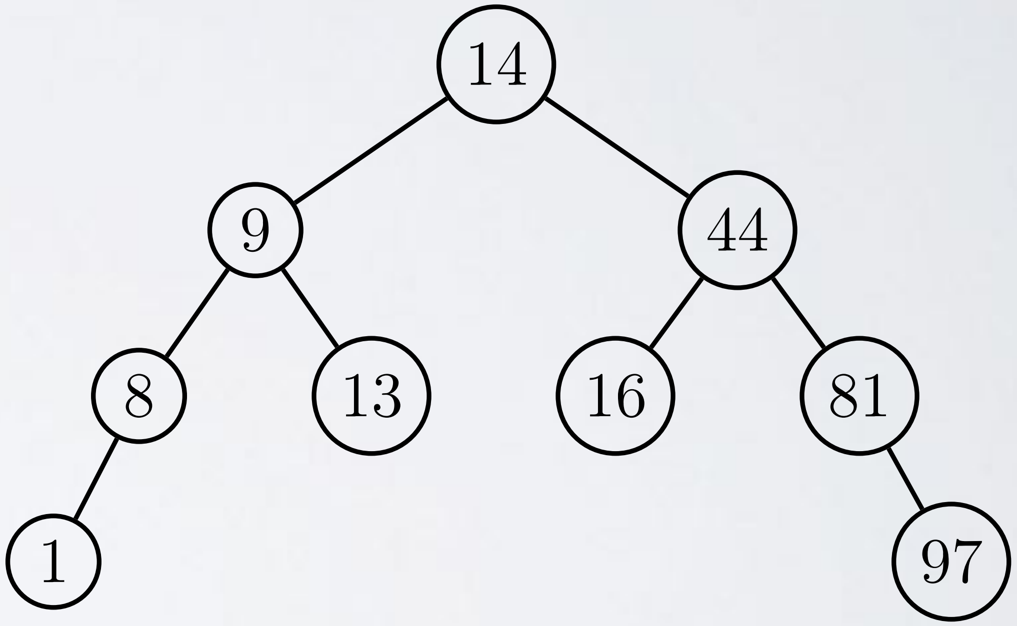
ДЕРЕВО ПОИСКА

- BST (Binary Search Tree).
- Каждому узлу **n** сопоставлен ключ **k(n)**.
- **k(x) < k(n)** для **x** ∈ **L(n)** — левое поддерево.
- **k(y) > k(n)** для **y** ∈ **R(n)** — правое поддерево.



ИНТЕРФЕЙС ДЕРЕВА ПОИСКА

- Поиск элемента по ключу
- Вставка элемента по ключу
- Удаление элемента по ключу
- Перечисление всех ключей

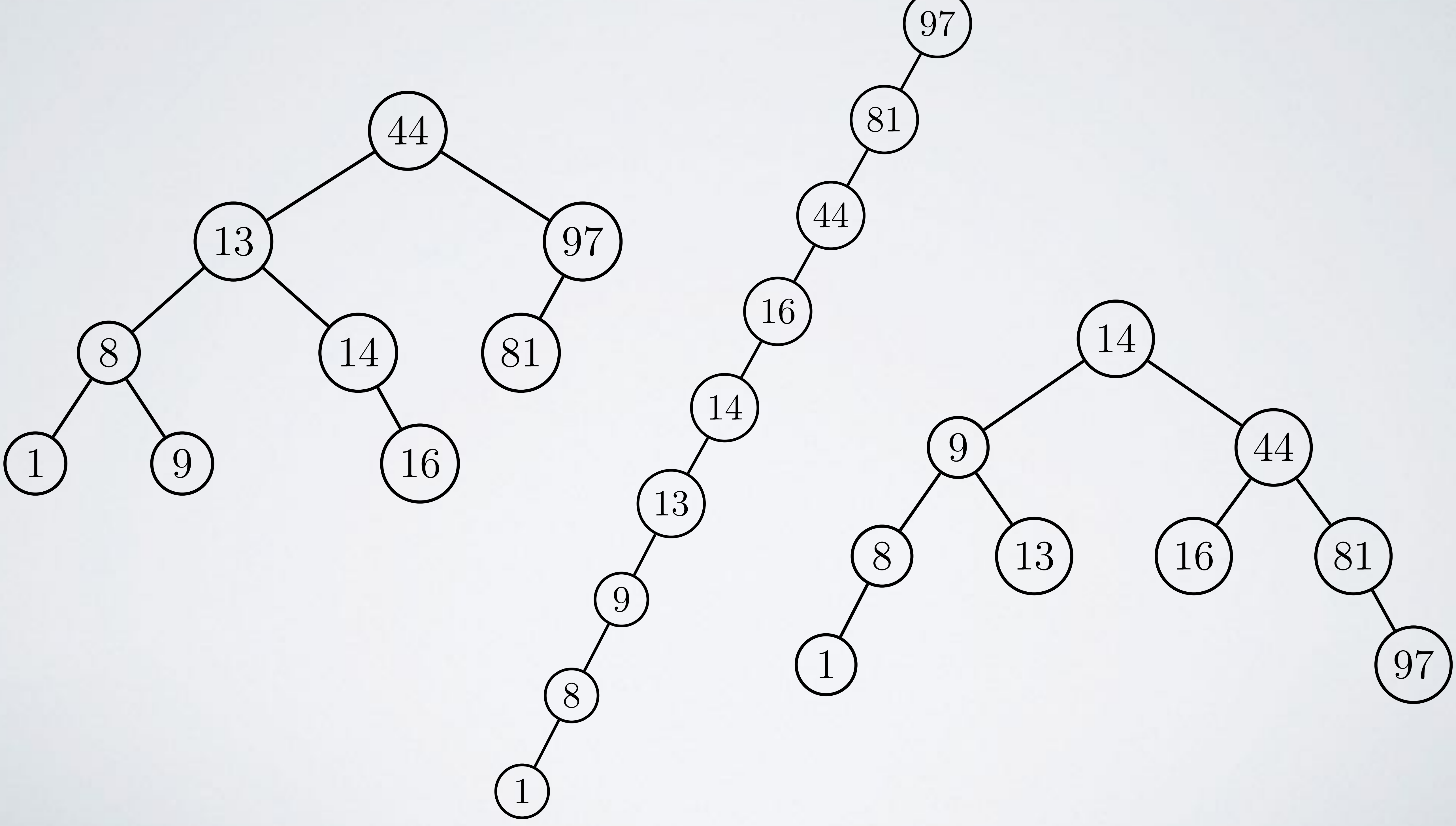


Поиск и вставка за O(h(N))!

ВЫСОТА ДЕРЕВА ПОИСКА

- Бинарное дерево высоты h содержит максимум 2^h-1 узлов.
- Значит высота $h(N) \geq \log(N)$.
- При добавлении случайных элементов $h(N) \sim 2{,}99 \log(N)$. Средняя глубина узла $\sim 1{,}39 \log(N)$.
- Но в худшем случае…

НЕ ВСЕ ДЕРЕВЬЯ ОДИНАКОВО ПОЛЕЗНЫ

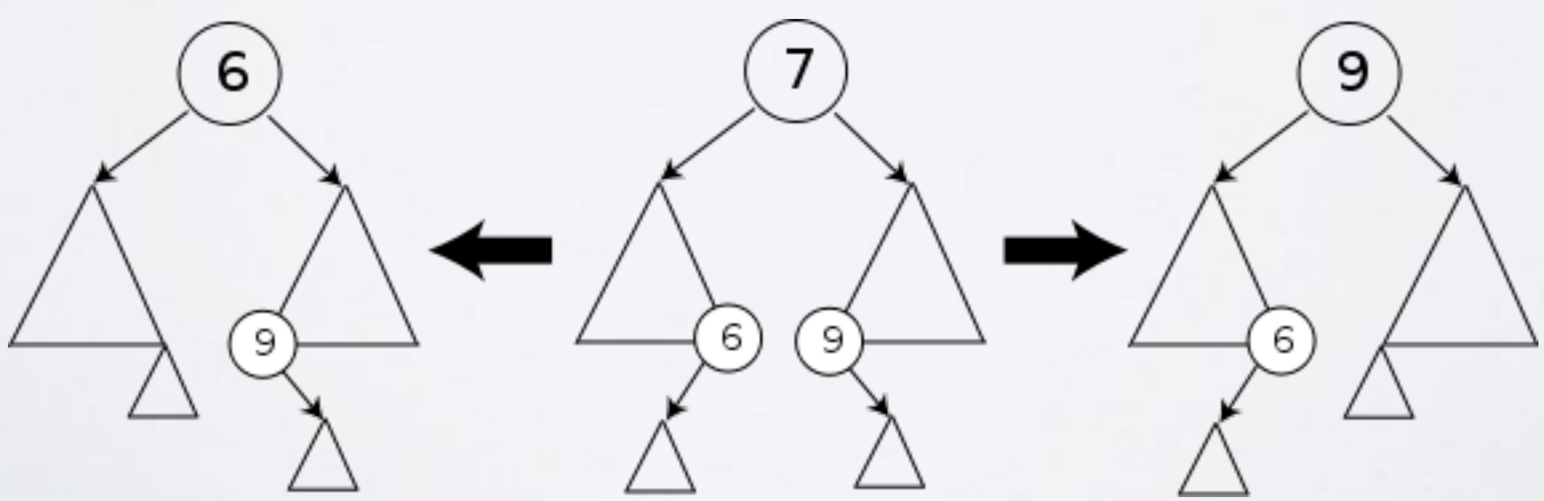


ПРИМЕНЕНИЯ ДЕРЕВА ПОИСКА

- АТД Множество (set)
- АТД Мультимножество (multiset)
- АТД Ассоциативный массив (отображение, map, словарь, dictionary)

УДАЛЕНИЕ ЭЛЕМЕНТА

- Если лист (нет потомков), то просто удаляем
- Если потомок один, он заменит удаляемый узел
- Если два потомка, то нужно найти либо самый **правый** узел **левого** поддерева, либо самый **левый** узел **правого** поддерева и поставить на место удаляемого узла

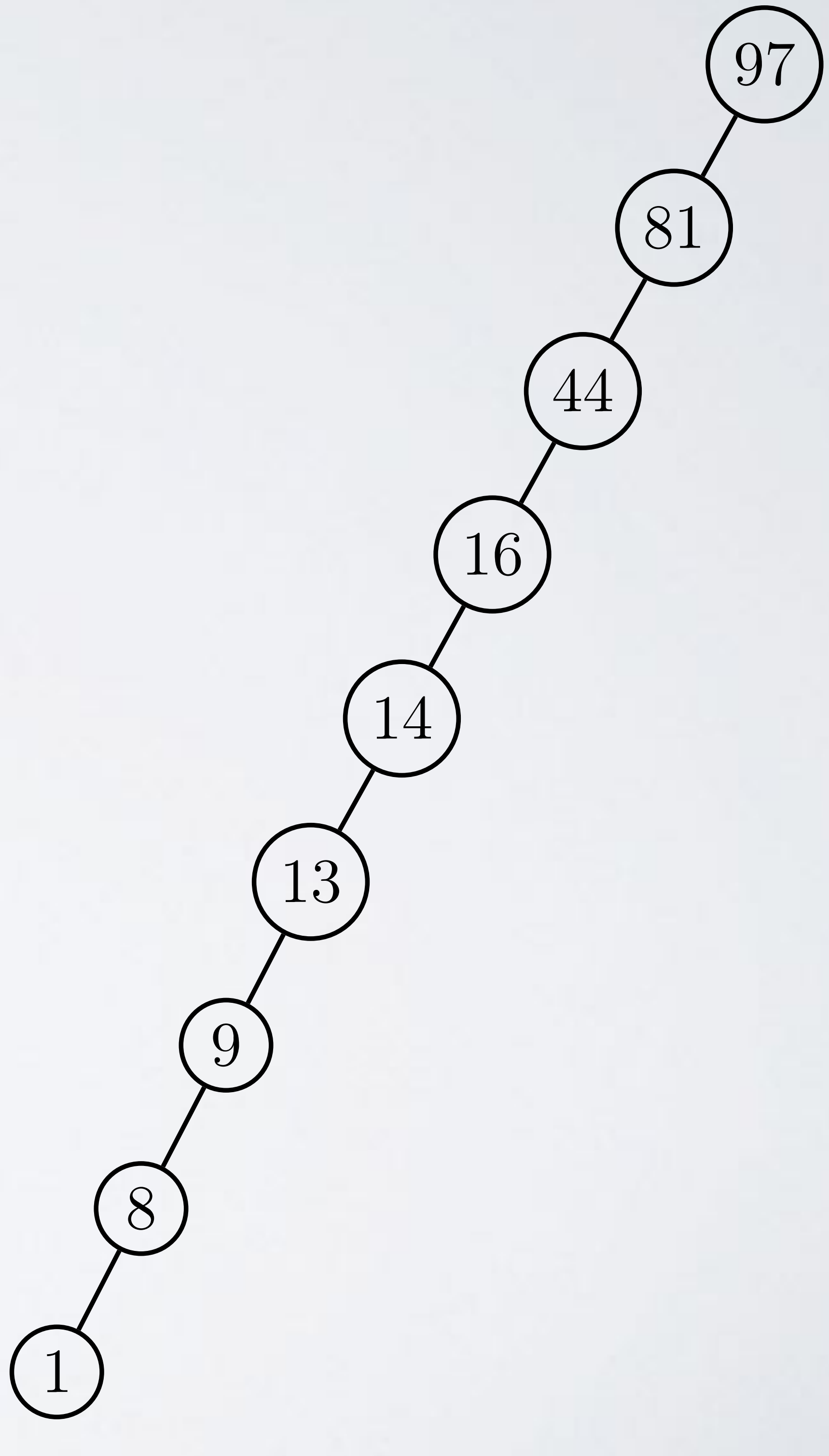


УДАЛЕНИЕ ЭЛЕМЕНТА

- Представленный алгоритм удаления приводит к тому, что высота дерева растет и становится ~ $N^{1/2}$.
- Даже если случайным образом выбирать, с какой стороны брать новый элемент.
- Есть ли способы гарантированно выполнять поиск и вставку за O(log(N))?

РЕШЕНИЯ ПРОБЛЕМЫ «КРИВЫХ» ДЕРЕВЬЕВ

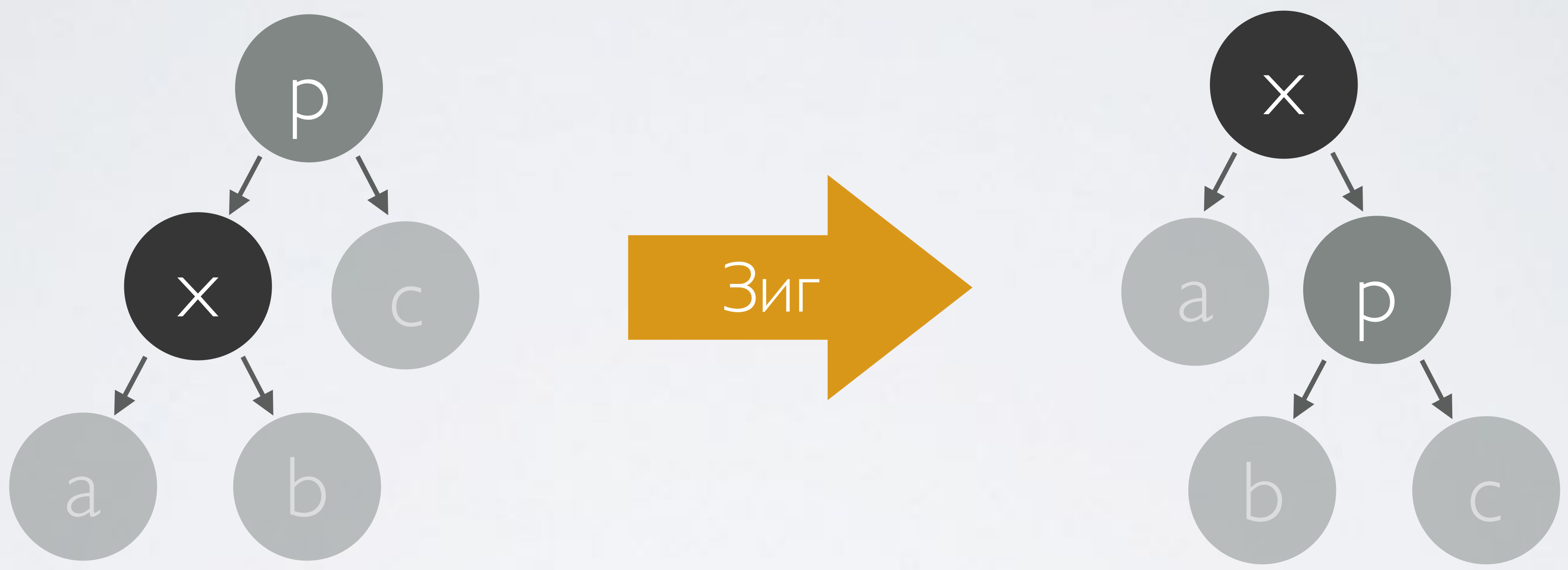
- Восстановление оптимальности:
  - «Выворачивание» *(splay trees)*,
  - АВЛ-деревья,
  - Красно-черные деревья.



SPLAY TREES

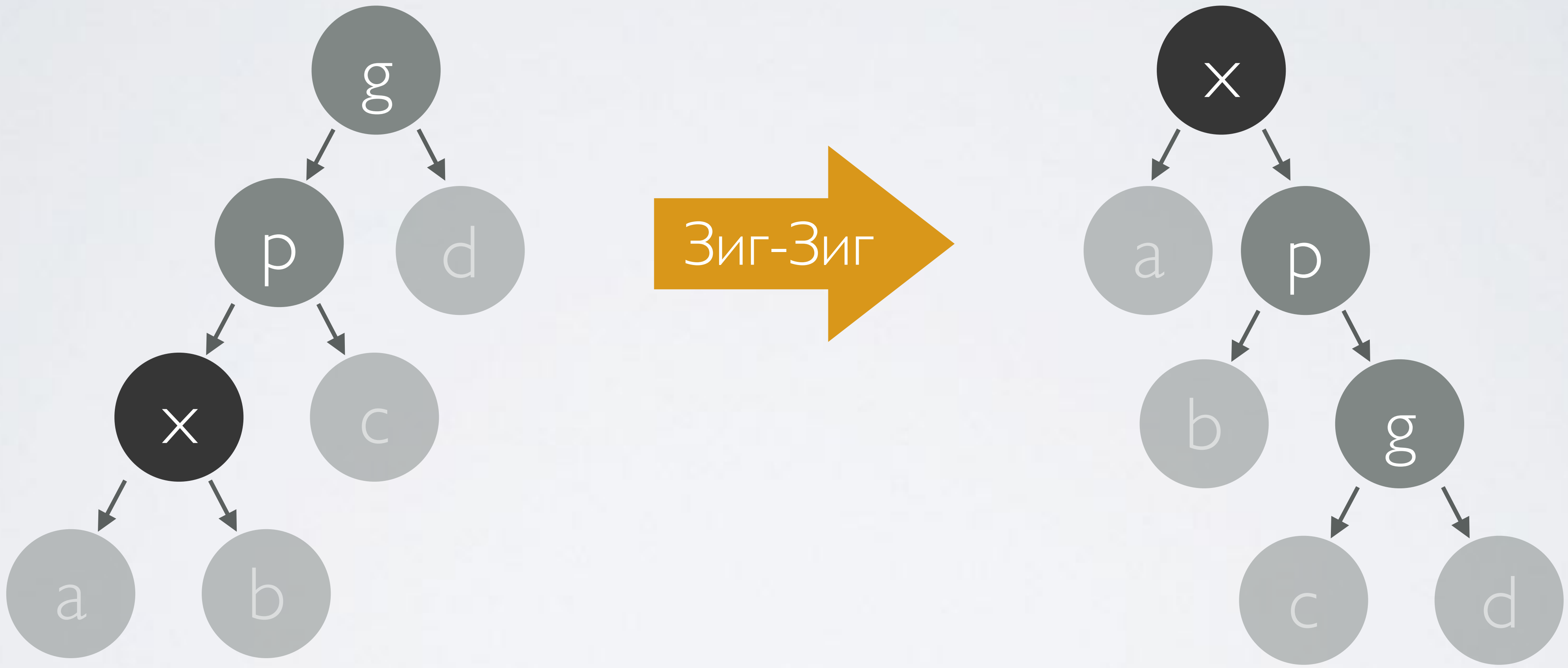
- Обычное дерево поиска, но после каждого поиска найденный элемент помещается в вершину.
- При удалении предок удаленного элемента помещается в вершину.
- Помещение в вершину происходит пошагово («всплытие»).
- «Средняя» сложность операций – O(log(N)).

ПОВОРОТ «ЗИГ»

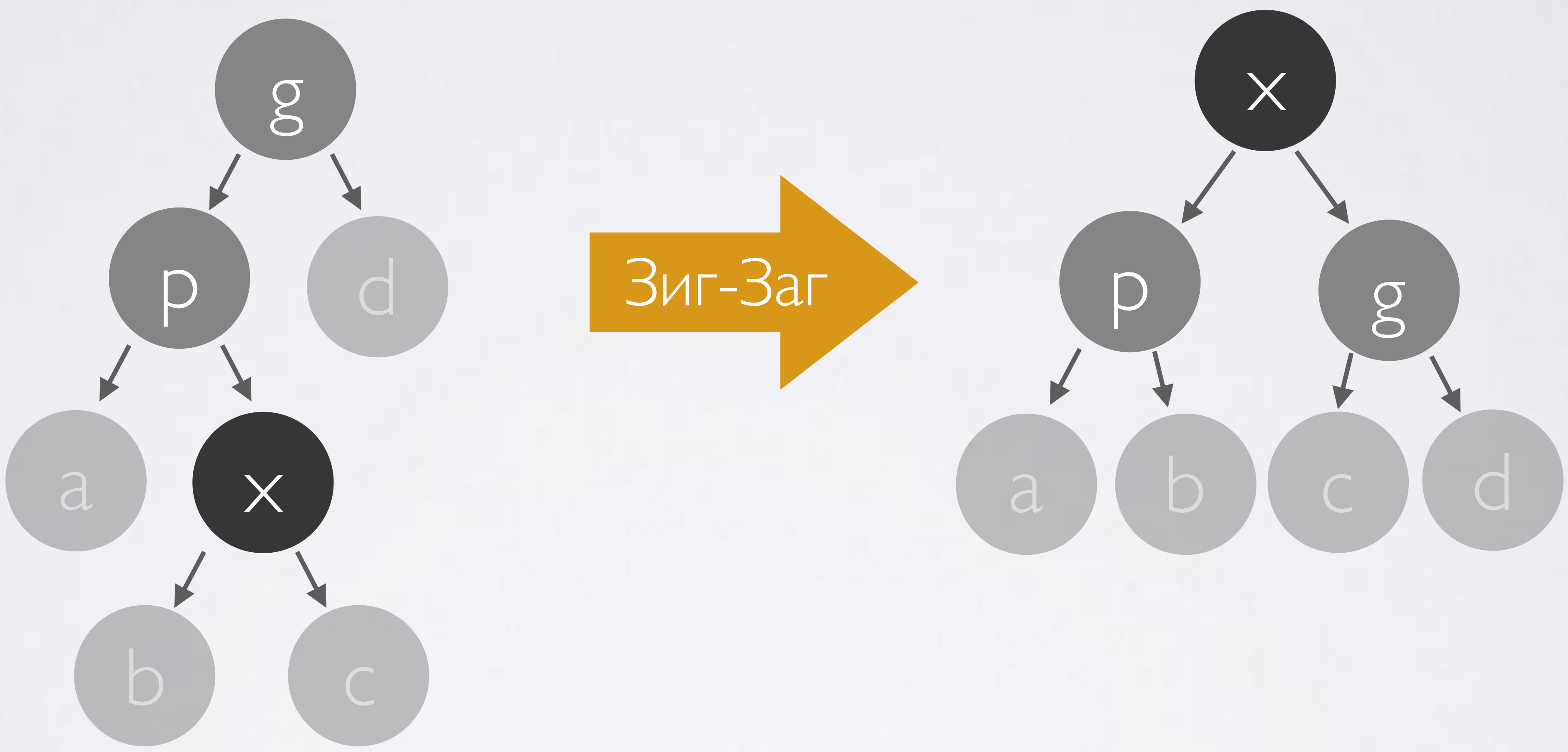


Последний шаг, если элемент X изначально на четном уровне.

ПОВОРОТ «ЗИГ-ЗИГ»



ПОВОРОТ «ЗИГ-ЗАГ»



АВЛ-ДЕРЕВЬЯ

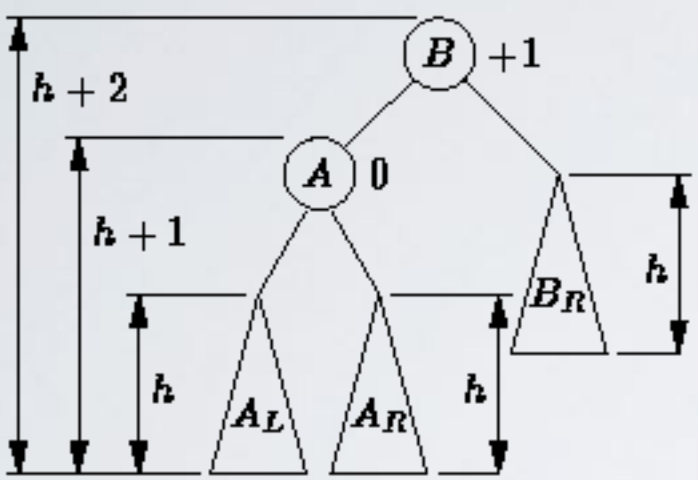
- 1962 г. Адельсон-Вельский и Ландис (СССР)

- **Сбалансированное дерево:** высоты двух родственных поддеревьев отличаются не более, чем на единицу

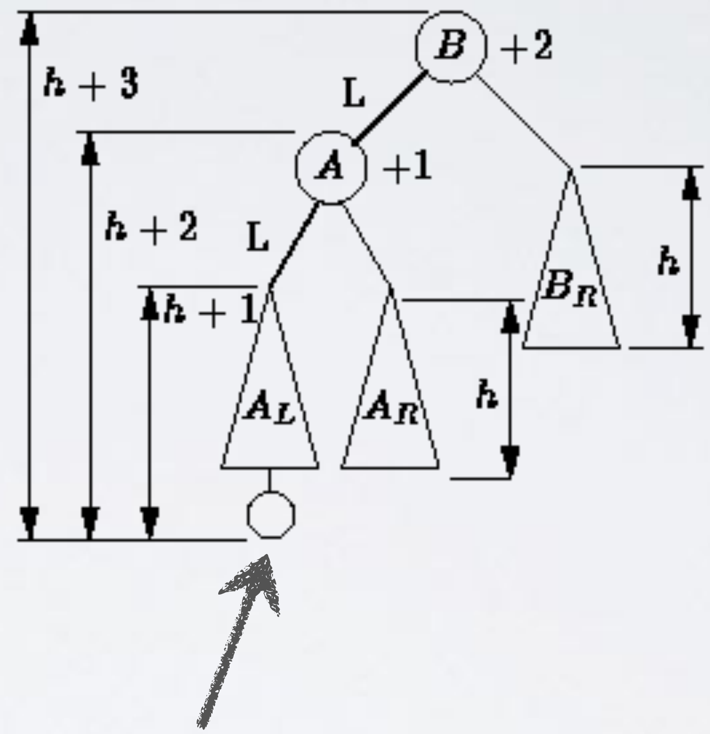
- **Перебалансировка** после операций вставки и удаления, нарушающих свойство сбалансированности. Идем снизу вверх (к корню), восстанавливая баланс.

- В узел добавляется показатель сбалансированности, равный разности высот поддеревьев (0, +1, −1).

БАЛАНСИРОВКА: МАЛЫЙ ЛЕВЫЙ ПОВОРОТ

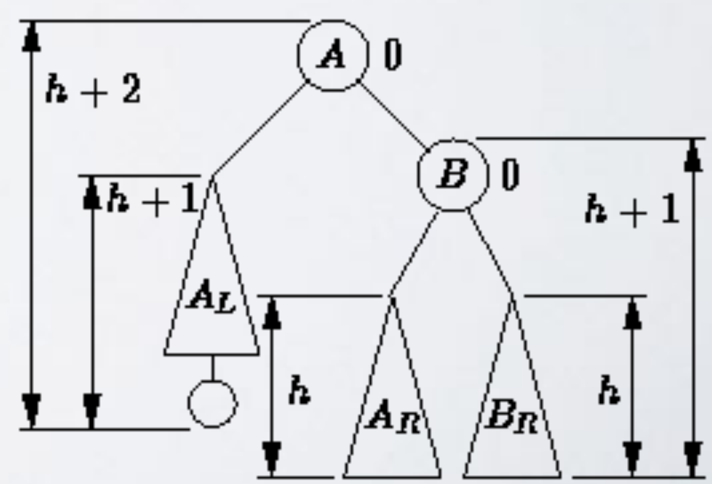


Было



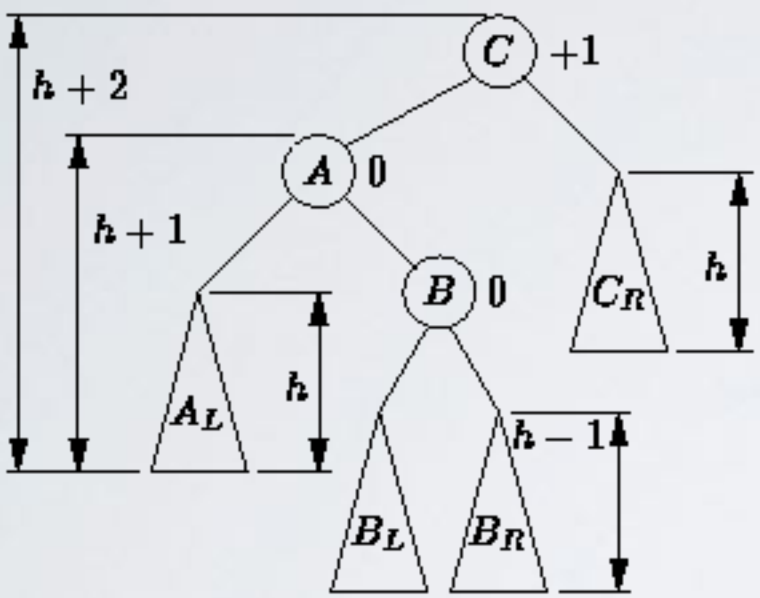
Вставили

Поворот вокруг B

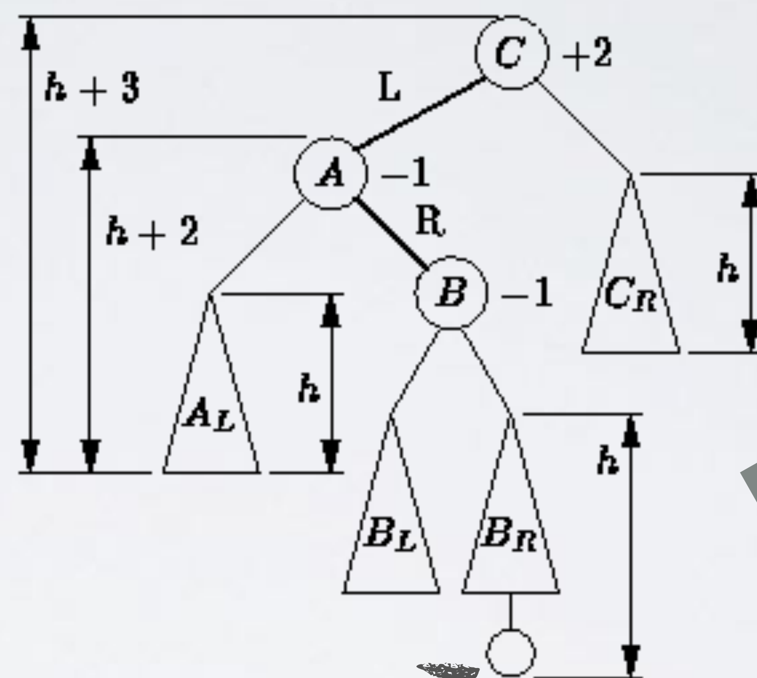


Выполняется, если B имеет баланс +2, а A имеет баланс ≥ 0.

БАЛАНСИРОВКА: БОЛЬШОЙ ЛЕВЫЙ ПОВОРОТ

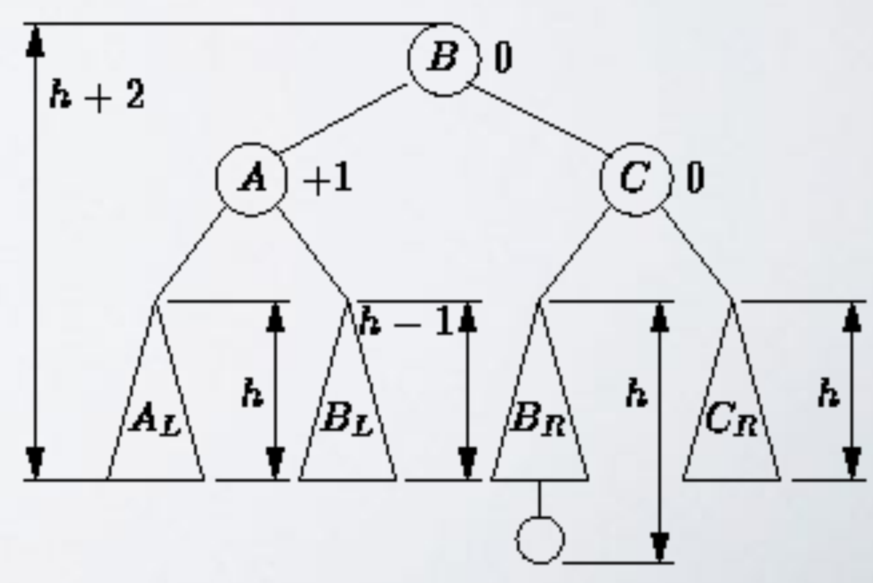


Было



Поворот вокруг A, C

Вставили



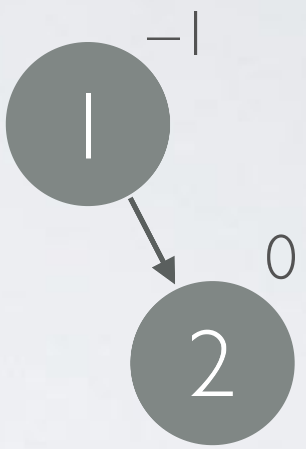
Выполняется, если C имеет баланс +2, а A имеет баланс –1.

БАЛАНСИРОВКА: ПРАВЫЕ ПОВОРОТЫ

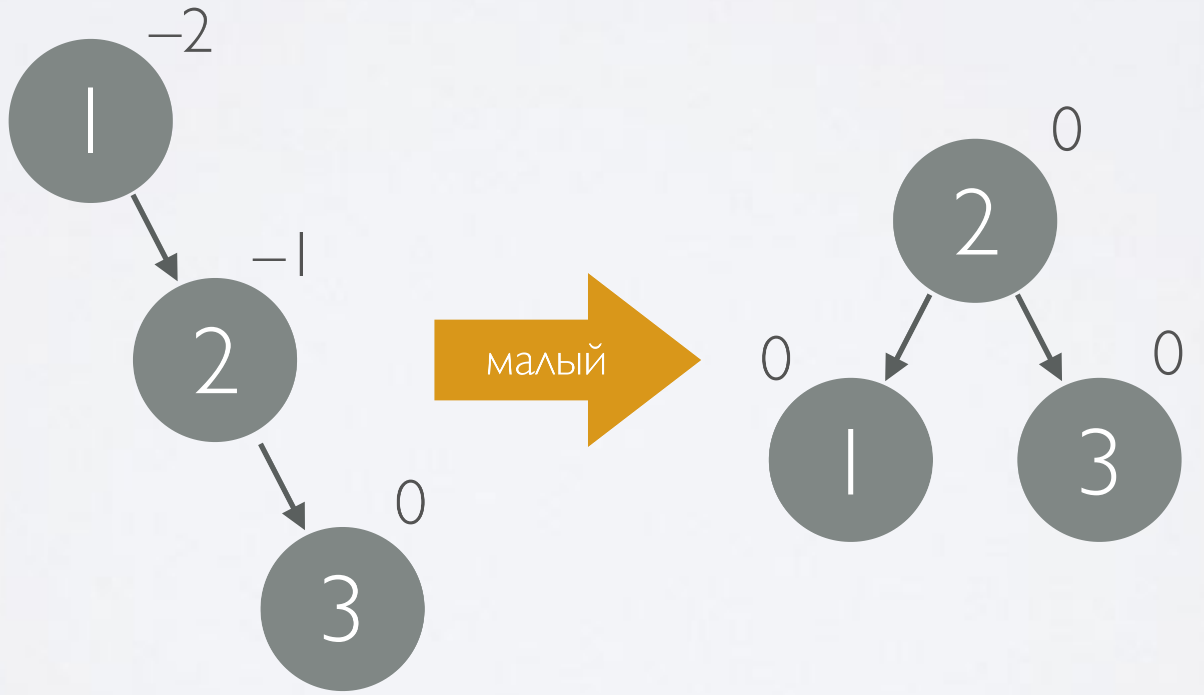
- Малый правый поворот аналогично малому левому
- Большой правый поворот аналогично большому левому

ПРИМЕР АВЛ-ДЕРЕВА

Вставляем 1

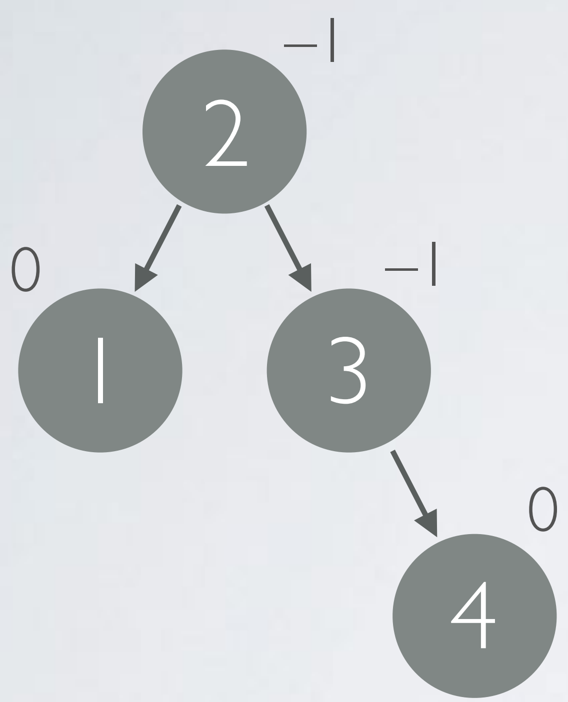


Вставляем 2

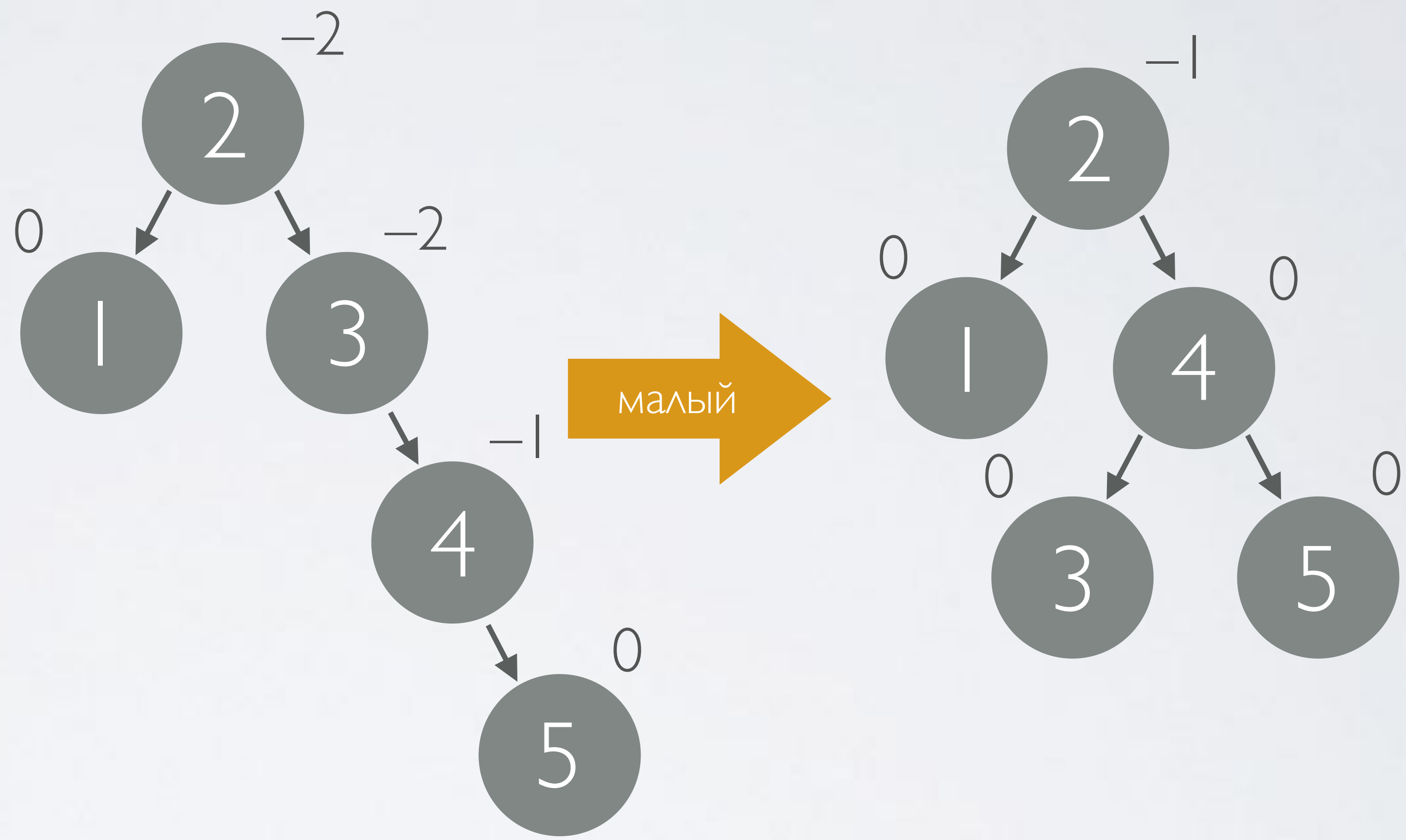


Вставляем 3

ПРИМЕР АВЛ-ДЕРЕВА

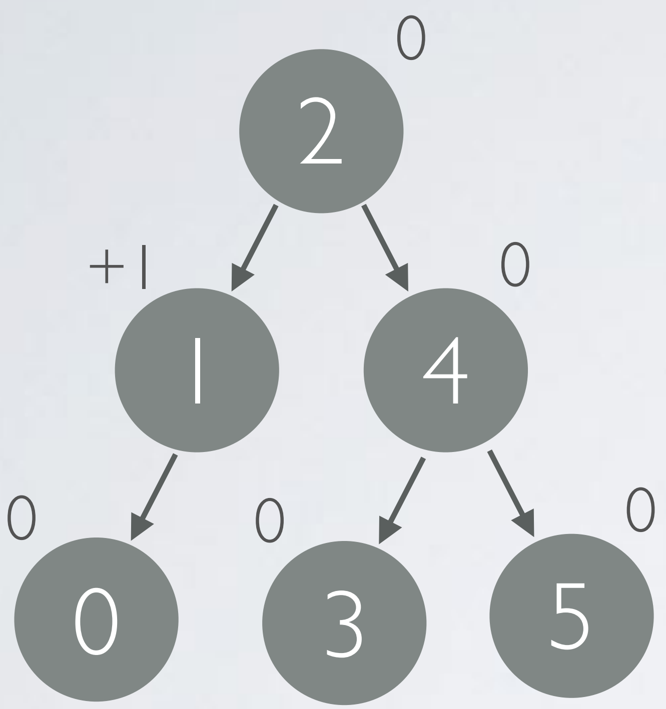


Вставляем 4



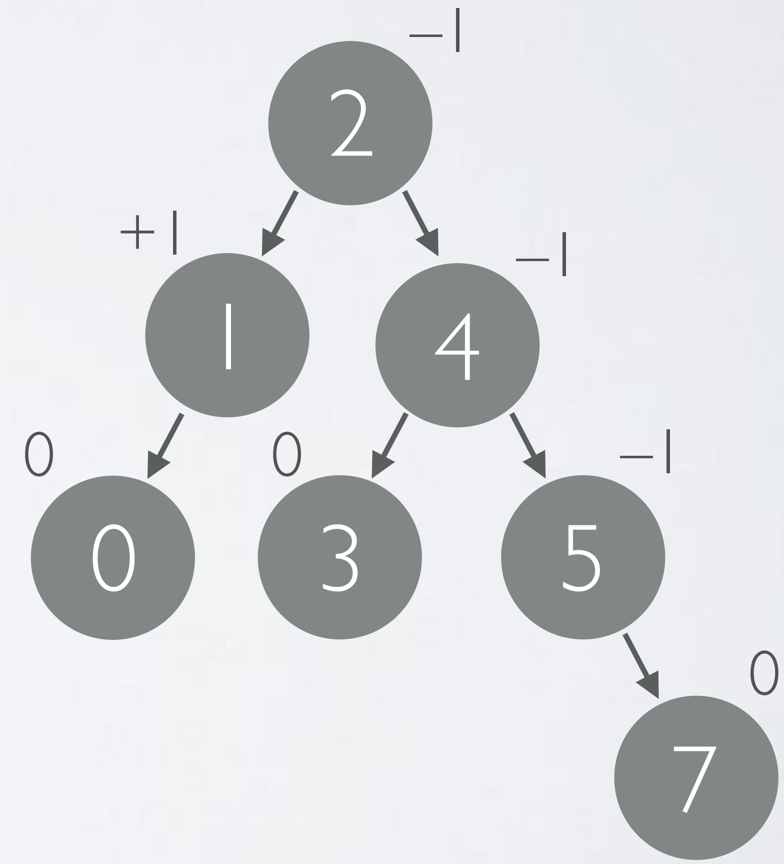
Вставляем 5

ПРИМЕР АВЛ-ДЕРЕВА

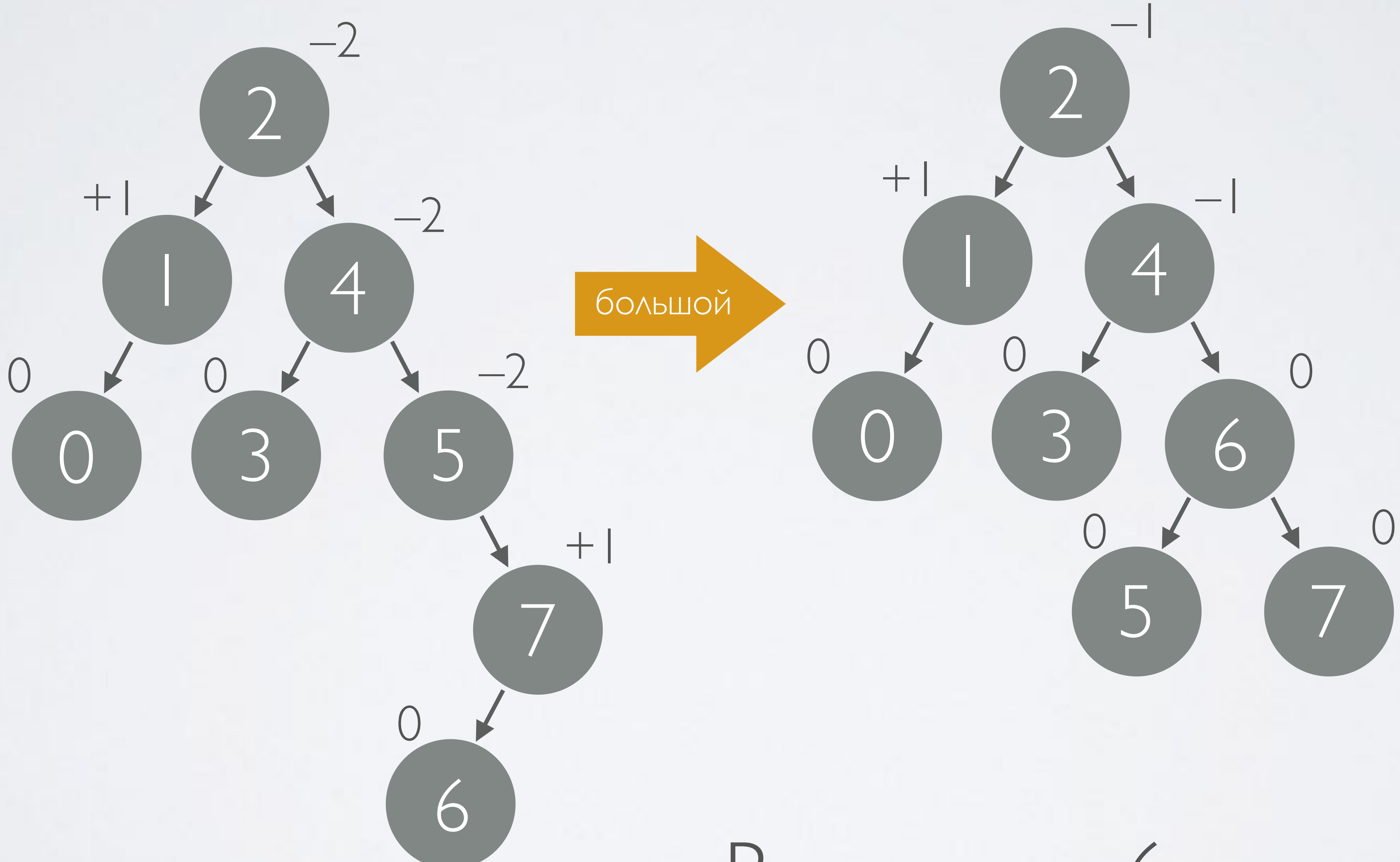


Вставляем 0

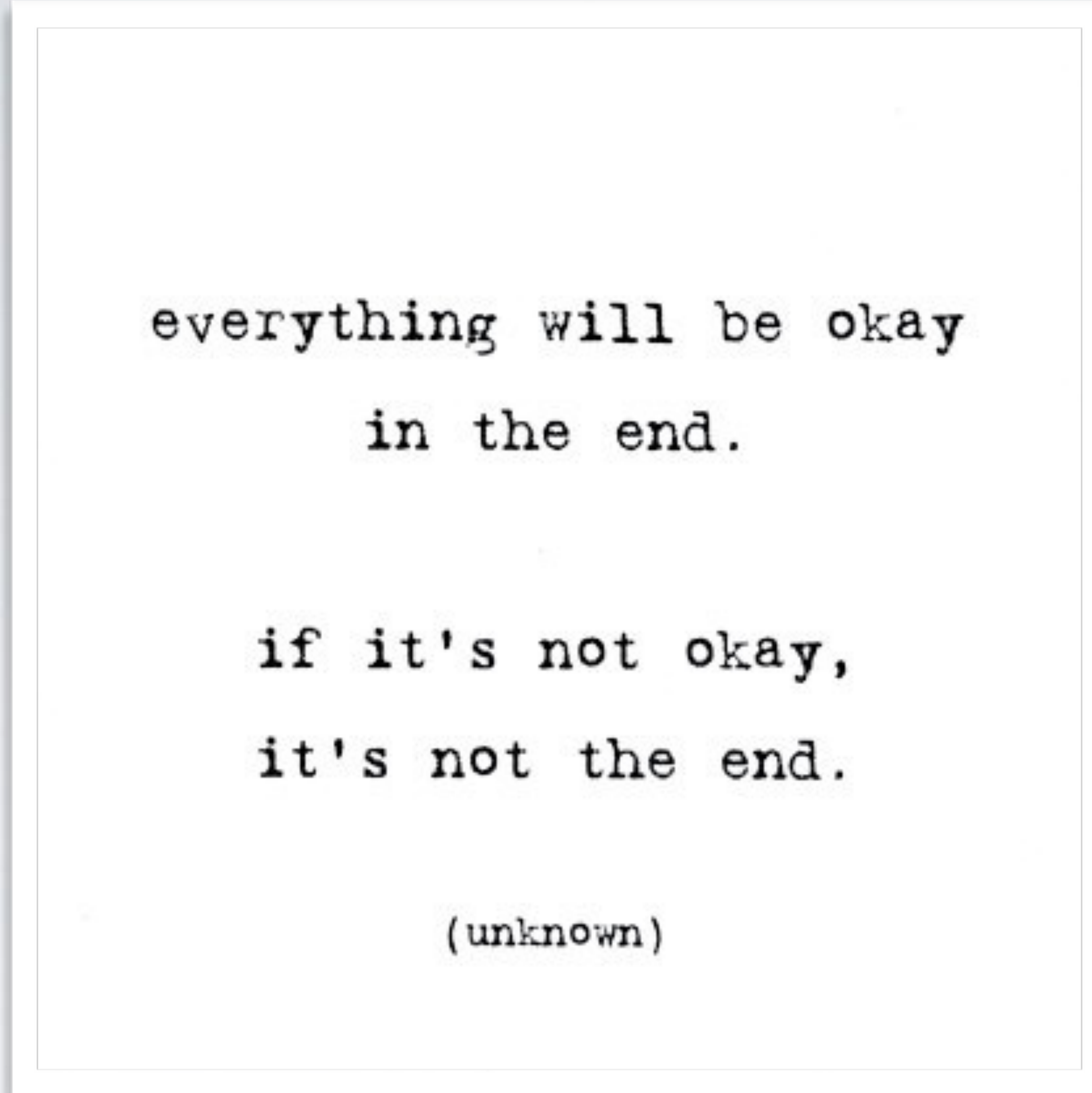
Вставляем 7



ПРИМЕР АВЛ-ДЕРЕВА



Вставляем 6



КОНЕЦ ВОСЬМОЙ ЛЕКЦИИ

Каждый программист в своей жизни должен ..., ... и реализовать модуль работы с деревом