Бинарные деревья поиска

***Источники:***

* <https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%D0%BF%D0%BE%D0%B8%D1%81%D0%BA%D0%B0>

**Двоичное дерево поиска –** двоичноедерево**,**  для которого выполняются следующие дополнительные условия:

* Оба поддерева — левое и правое — являются двоичными деревьями поиска;
* У всех узлов *левого* поддерева произвольного узла X значения ключей данных *меньше*, нежели значение ключа данных самого узла X;
* У всех узлов *правого* поддерева произвольного узла X значения ключей данных *больше либо равны*, нежели значение ключа данных самого узла X.

Данные характеристики бинарного дерева поиска позволяют воспроизводить быстрый как поиск, так и вставку или удаление элемента по ключу - O(log n), в среднем случае и O(n) - в худшем.

Двоичное дерево поиска применяется для построения более абстрактных структур, таких, как множества, мультимножества и ассоциативные массивы.

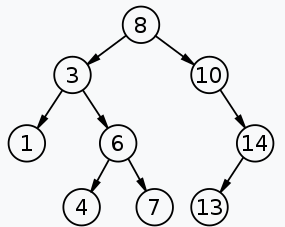


Рисунок 1. Бинарное дерево поиска

Основные операции в двоичном дереве поиска

Базовый интерфейс двоичного дерева поиска состоит из трёх операций:

* FIND(K) — поиск узла, в котором хранится пара (key, value) с key = K.
* INSERT(K, V) — добавление в дерево пары (key, value) = (K, V).
* REMOVE(K) — удаление узла, в котором хранится пара (key, value) с key = K.

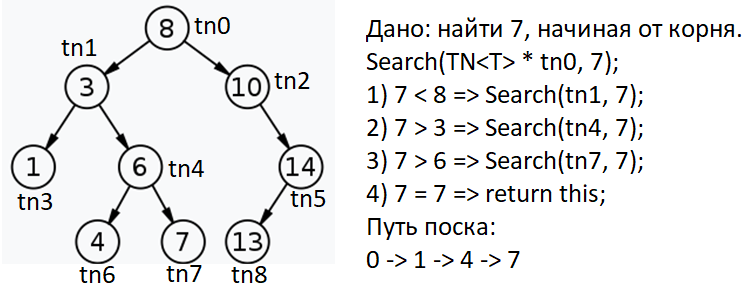
### Поиск элемента (FIND)

**Дано**: дерево Т и ключ K.

**Задача**: проверить, есть ли узел с ключом K в дереве Т, и если да, то вернуть ссылку на этот узел.

**Алгоритм**:

* Если дерево пусто, сообщить, что узел не найден, и остановиться.
* Иначе сравнить K со значением ключа корневого узла X.
  + Если K=X, выдать ссылку на этот узел и остановиться.
  + Если K>X, рекурсивно искать ключ K в правом поддереве Т.
  + Если K<X, рекурсивно искать ключ K в левом поддереве Т.



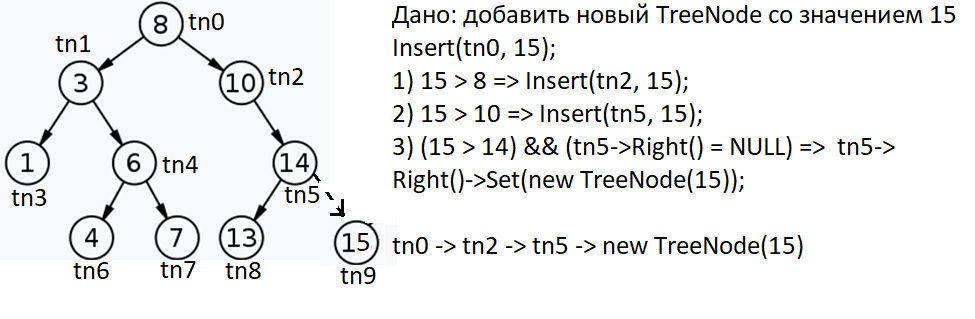
### Добавление элемента (INSERT)

**Дано**: дерево Т и пара (K, V).

**Задача**: вставить пару (K, V) в дерево Т (при совпадении K, заменить V).

**Алгоритм**:

* Если дерево пусто, заменить его на дерево с одним корневым узлом ((K, V), null, null) и остановиться.
* Иначе сравнить K с ключом корневого узла X.
  + Если K>X, рекурсивно добавить (K, V) в правое поддерево Т.
  + Если K<X, рекурсивно добавить (K, V) в левое поддерево Т.
  + Если K=X, заменить V текущего узла новым значением.



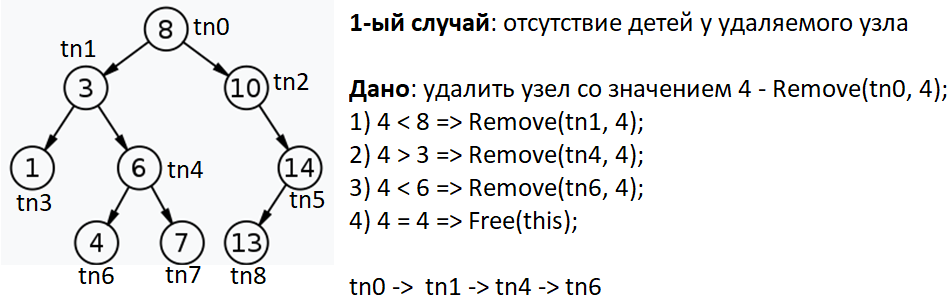
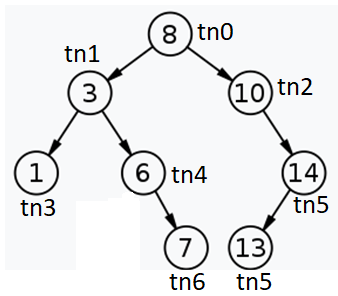
### Удаление узла (REMOVE)

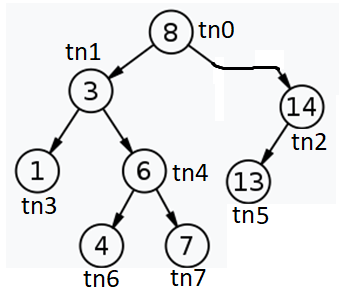
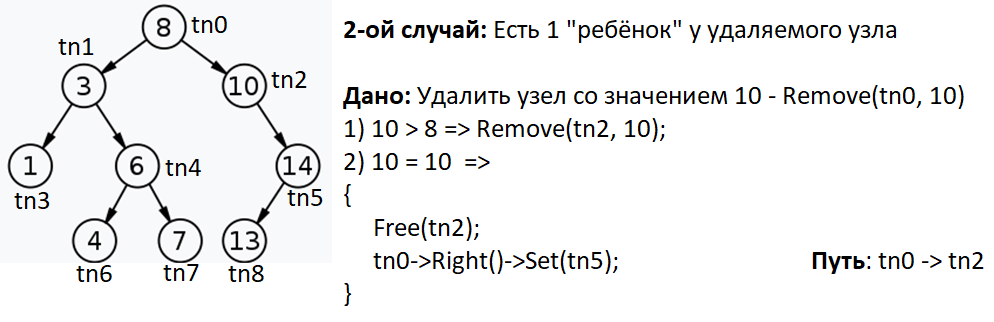
**Дано**: дерево Т с корнем n и ключом K.

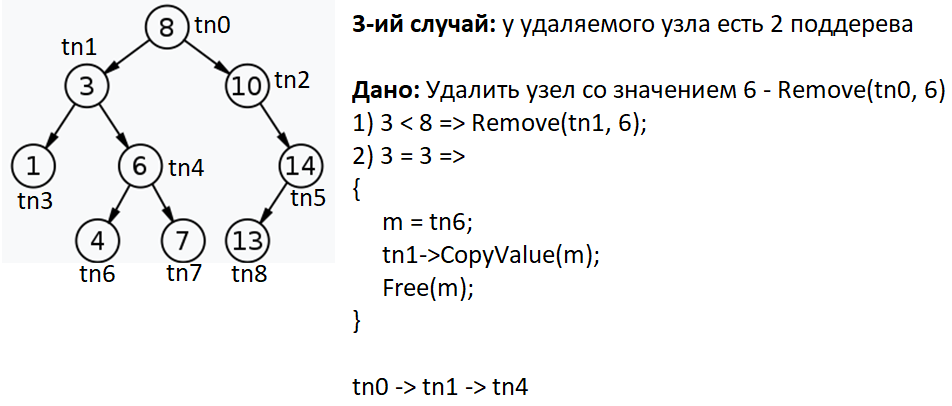
**Задача**: удалить из дерева Т узел с ключом K (если такой есть).

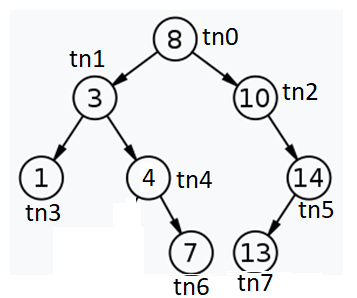
**Алгоритм**:

* Если дерево T пусто, остановиться;
* Иначе сравнить K с ключом X корневого узла n.
  + Если K>X, рекурсивно удалить K из правого поддерева Т;
  + Если K<X, рекурсивно удалить K из левого поддерева Т;
  + Если K=X, то необходимо рассмотреть три случая.
    - Если обоих детей нет, то удаляем текущий узел и обнуляем ссылку на него у родительского узла;
    - Если одного из детей нет, то значения полей ребёнка m ставим вместо соответствующих значений корневого узла, затирая его старые значения, и освобождаем память, занимаемую узлом m;
    - Если оба ребёнка присутствуют, то
      * Если левый узел m правого поддерева отсутствует (n->right->left)
        + Копируем из правого узла в удаляемый поля K, V и ссылку на правый узел правого потомка.
      * Иначе
        + Возьмём самый левый узел m, правого поддерева n->right;
        + Скопируем данные (кроме ссылок на дочерние элементы) из m в n;
        + Рекурсивно удалим узел m.







Разделение дерева по ключу

**Функция**: разбиение одного дерева на два: с ключами <K и >=K;