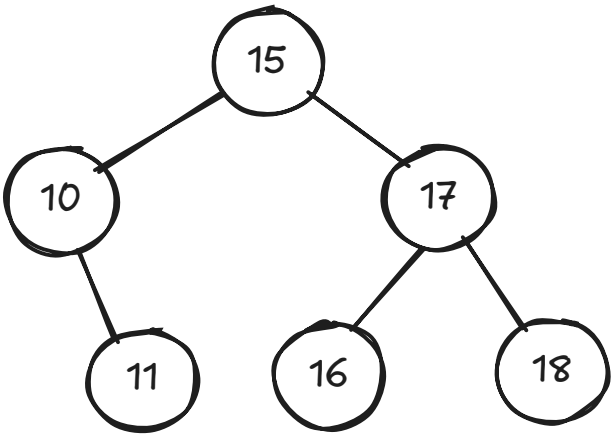
15. Binary search trees, algorithms for finding/inserting/removing an element, complexity analysis, definition of a balanced tree, asymptotical estimate of the height of a balanced tree.

> Бинарное дерево называется **бинарным деревом поиска**, если между каждым узлом N и его поддеревьями $T_l(N)$ и $T_r(N)$ выполняется следующее соотношение:
>
> $$T_l(N) < N < T_r(N)$$

(для каждого узла значения в левом поддереве *меньше* него, а в правом - *больше*).



пример бинарного дерева поиска

Алгоритм нахождения элемента с данным значением, начиная с какого-то узла.

Выполняется за O(h) (в лучшем случае - O(log n), в худшем - O(n)).

```
Node*& find(Node*& node, const T& val) {
        if(node == nullptr || val == node->val){
                return node;
        }
        if(val < node->val){
                return find(node->left, val);
        }
        return find(node->right, val);
}
```

Алгоритм вставки элемента

Выполняется за O(h) (в лучшем случае - O(log n), в худшем - O(n)).

```
void insert(const T& val) {
        Node*& spot = find(root, val);
```

```
        if(spot == nullptr) {
                spot = new Node{val};
        }
}
```

используем " * & ", чтобы была возможность менять spot

Алгоритм удаления элемента

Выполняется за O(h) (в лучшем случае - O(log n), в худшем - O(n)).

successor (s) данного элемента N это

$$s = \min\{\text{val} \mid \text{val} > N\}$$

left_subtree < s < right_subtree
минимальный элемент в правом поддереве

predecessor (p) данного элемента N это

$$p = \max\{\text{val} \mid \text{val} < N\}$$

максимальный элемент в левом поддереве

```
Node* findMin(Node* subtreeRoot) {
   while (subtreeRoot && subtreeRoot->left) {
      subtreeRoot = subtreeRoot->left;
   }
   return subtreeRoot;
}

void erase(Node*& node) {
   if (!node->right) {
      Node* temp = node->left;
      delete node;
      node = temp;
      return;
   }

   Node* successor = node->right;
   while (successor->left) {
      successor = successor->left;
   }
   node->value = std::move(successor->value);
   erase(successor);
}
```

Кроме того у нас должна быть возможность *пройтись по всем элементам дерева*. Для этого будем использовать **post-order, in-order и pre-order** traversal.

- Для *печатания* всех элеметов лучше использовать in-order traversal, т.к. напечатается в сортированном виде.
- Для *удаления* всех элементов - post-order, т.к. сначала удалятся левое и правое поддеревья, а потом только сам узел.

Сбалансированные деревья

> **Уровнем дерева** называется число узлов в самом длинном пути от корня до листа (включая корень и лист). (высота дерева число ребер, а здесь - число узлов)

- *Правой высотой узла* называется уровень его правого поддерева.
- *Левой высотой узла* называется уровень его левого поддерева.

> Дерево называется **сбалансированным** если для каждого узла разность правой и левой высот не больше 1. Эта разность называется **фактором балансировки** (balance factor).

Для любого сбалансированного бинарного дерева высота h асимптотически удовлетворяет:

$$h = O(\log n)$$

где n — общее количество узлов в дереве.