Part 1 (Implementation of the remaining functions of our `rbtree_map`).

1. move constructor
2. copy assignment operator
3. copy constructor
4. Implement **`iterator find(const key_type& k)`** function (returns nullptr if not found)
5. Implement the following constructor (test5)
   `template <typename Comp>`
   `rbtree_map(std::initializer_list<value_type> init, const Comp& comp = Compare()`

`*` Update our **`erase`** method (which is for regular BSTs) by the deletion algorithm given in Cormen's book 13.4 paragraph (which is for Red-Black trees). This task is not mandatory, but it'd be nice to have it implemented.

Part2 (algorithmic problems on our `rbtree_map`) .

Problem 1. Validate Binary Tree.

As you know, Binary Search Tree should satisfy the following 3 conditions:

- ***The left subtree of a node contains only nodes with keys less than the node's key.***
- ***The right subtree of a node contains only nodes with keys greater than the node's key.***
- ***Both the left and right subtrees must also be binary search trees.***

In this problem you need to implement the
`bool validate_btree(node* p)`
function which should check the above 3 conditions. You have to use only O(1) memory.
To test your implementation I added **`test6`** and **`test6`** test functions.
`test6` tests the following examples

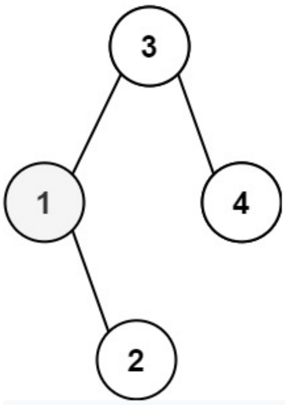
```
/*
 *          2         true
 *         / \
 *        1   3
 */

/*          5         false
 *         / \
 *        1   4
 *           / \
 *          3   6
 */

/*          5         false
 *         / \
 *        4   6
 *           / \
 *          3   7
 */
```

`test7` constructs **`rbtree_map<int, string>`** with 10 elements, then it removes each element by calling our **`erase`** method and validates the three after deletion.

Problem 2. Kth Smallest Element

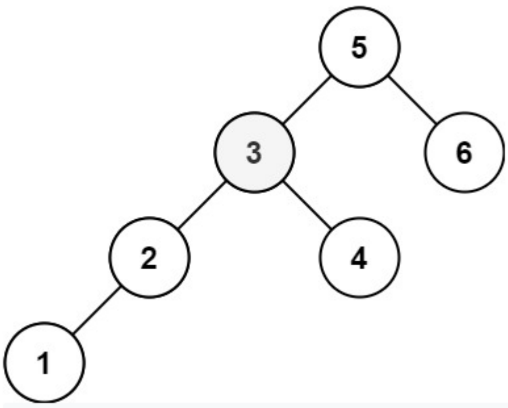
Return an iterator to the K-th element (e.g., it’s K-th smallest element when Compare = std::less<K>) 1-indexed. Return nullptr if not found.
The following examples assume **`Compare = std::less<K>`**.

Example 1:




```
Input: root = [3,1,4,null,2], k = 1
Output: 1
```

Example 2:



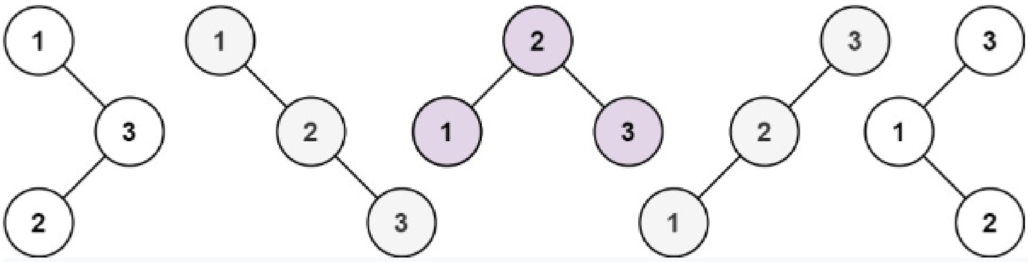

```
Input: root = [5,3,6,2,4,null,null,1], k = 3
Output: 3
```

Implement the following function.
`iterator kth_element(size_type k)`
The corresponding test function is **`test8`**.

Problem 3. Number of Unique Binary Search Trees.

Given an integer n, return the number of structurally unique BST's (binary search trees) which have exactly n nodes of unique values from 1 to n.

Example 1:




```
Input: n = 3
Output: 5
```

Example 2:

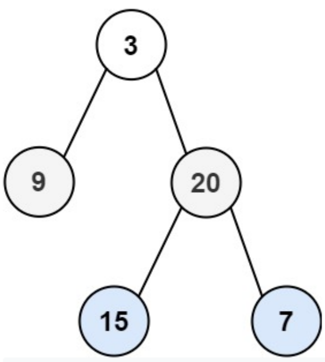
```
Input: n = 1
Output: 1
```

Implement **`unique_BSTs::num_of_unique_bsts`**function. The corresponding test function is **`test9`.**

Problem 4. Zigzag Level Order Traversal

Given the root of a binary tree, return the zigzag level order traversal of its nodes' values. (i.e., from left to right, then right to left for the next level and alternate between)

Example 1:




```
Input: root = [3,9,20,null,null,15,7]
Output: [[3],[20,9],[15,7]]
```

Example 2:

```
Input: root = [1]
Output: [[1]]
```

Example 3:

```
Input: root = []
Output: []
```

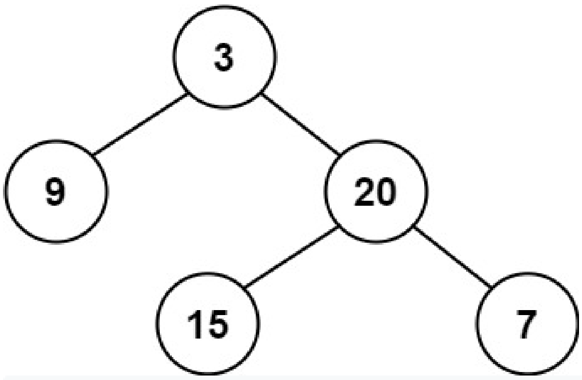
Write your implementation in the **`vector<vector<value_type>> zigzag_traversal(node* p) const`** function. The corresponding test function is `test10`.

Problem 5. Check if a BS tree is balanced

Determine if it is height-balanced.
For this problem, a height-balanced binary tree is defined as:
A binary tree in which the left and right subtrees of *every* node differ in height by no more than 1.

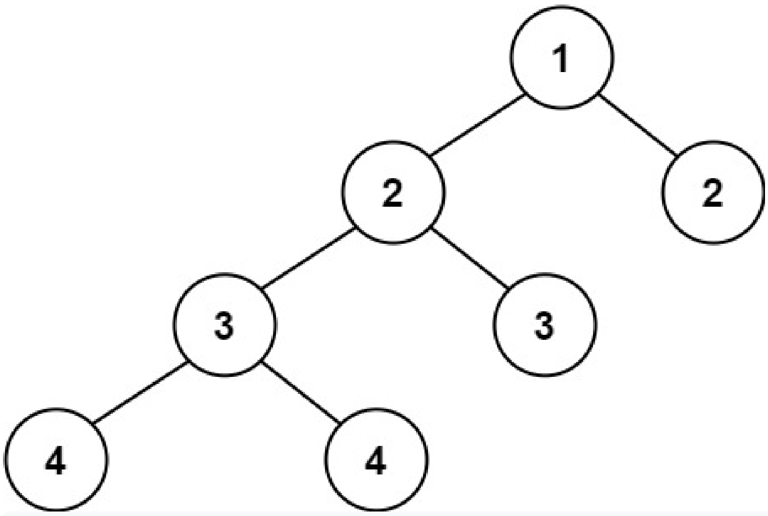
(Note that, in general Red-Black trees don't define this kind of balance. For RB trees balanced means tree's height is not longer than O(2*log(n))).

Example 1:




```
Input: root = [3,9,20,null,null,15,7]
Output: true
```

Example 2:




```
Input: root = [1,2,2,3,3,null,null,4,4]
Output: false
```

You need to implement `bool is_balanced(const node* p) const` function. `test11` checks your implementation.

Additional Problems

- An interesting problem from leetcode: construct-binary-tree-from-preorder-and-inorder-traversal.
- If you solved the previous problem you can try to solve this “hard” problem too, based on your previous solution: Serialize and Deserialize Binary Tree.