D. Broken BST

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Let T be arbitrary binary tree — tree, every vertex of which has no more than two children. Given tree is rooted, so there exists only one vertex which doesn't have a parent — it's the root of a tree. Every vertex has an integer number written on it. Following algorithm is run on every value from the tree T:

1. Set pointer to the root of a tree.
2. Return success if the value in the current vertex is equal to the number you are looking for
3. Go to the left child of the vertex if the value in the current vertex is greater than the number you are looking for
4. Go to the right child of the vertex if the value in the current vertex is less than the number you are looking for
5. Return fail if you try to go to the vertex that doesn't exist

Here is the pseudo-code of the described algorithm:

```
bool find(TreeNode t, int x) {
    if (t == null)
        return false;
    if (t.value == x)
        return true;
    if (x < t.value)
        return find(t.left, x);
    else
        return find(t.right, x);
}
find(root, x);
```

The described algorithm works correctly if the tree is binary search tree (i.e. for each node the values of left subtree are less than the value in the node, the values of right subtree are greater than the value in the node). But it can return invalid result if tree is not a binary search tree.

Since the given tree is not necessarily a binary search tree, not all numbers can be found this way. Your task is to calculate, how many times the search will fail being running on every value from the tree.

If the tree has multiple vertices with the same values on them then you should run algorithm on every one of them separately.

Input

First line contains integer number n ($1 \le n \le 10^5$) — number of vertices in the tree.

Each of the next n lines contains 3 numbers v, l, r ($0 \le v \le 10^9$) — value on current vertex, index of the left child of the vertex and index of the right child of the vertex, respectively. If some child doesn't exist then number -1 is set instead. Note that different vertices of the tree may contain the same values.

Output

Print number of times when search algorithm will fail.

Examples

input
3 15 -1 -1 10 1 3 5 -1 -1
output
2

input
8 6 2 3 3 4 5 12 6 7

```
1 -1 8
4 -1 -1
5 -1 -1
14 -1 -1
2 -1 -1
```

output

```
1
```

Note

In the example the root of the tree in vertex 2. Search of numbers 5 and 15 will return fail because on the first step algorithm will choose the subtree which doesn't contain numbers you are looking for.