

3 Is it a binary search tree? Hard version.

Problem Introduction

In this problem you are going to solve the same problem as the previous one, but for a more general case, when binary search tree may contain equal keys.

Problem Description

Task. You are given a binary tree with integers as its keys. You need to test whether it is a correct binary search tree. Note that there can be duplicate integers in the tree, and this is allowed. The definition of the binary search tree in such case is the following: for any node of the tree, if its key is x , then for any node in its left subtree its key must be strictly less than x , and for any node in its right subtree its key must be greater than **or equal** to x . In other words, smaller elements are to the left, bigger elements are to the right, and duplicates are always to the right. You need to check whether the given binary tree structure satisfies this condition. You are guaranteed that the input contains a valid binary tree. That is, it is a tree, and each node has at most two children.

Input Format. The first line contains the number of vertices n . The vertices of the tree are numbered from 0 to $n - 1$. Vertex 0 is the root.

The next n lines contain information about vertices 0, 1, ..., $n - 1$ in order. Each of these lines contains three integers key_i , $left_i$ and $right_i$ — key_i is the key of the i -th vertex, $left_i$ is the index of the left child of the i -th vertex, and $right_i$ is the index of the right child of the i -th vertex. If i doesn't have left or right child (or both), the corresponding $left_i$ or $right_i$ (or both) will be equal to -1 .

Constraints. $0 \leq n \leq 10^5$; $-2^{31} \leq key_i \leq 2^{31} - 1$; $-1 \leq left_i, right_i \leq n - 1$. It is guaranteed that the input represents a valid binary tree. In particular, if $left_i \neq -1$ and $right_i \neq -1$, then $left_i \neq right_i$. Also, a vertex cannot be a child of two different vertices. Also, each vertex is a descendant of the root vertex. Note that the minimum and the maximum possible values of the 32-bit integer type are allowed to be keys in the tree — beware of integer overflow!

Output Format. If the given binary tree is a correct binary search tree (see the definition in the problem description), output one word “CORRECT” (without quotes). Otherwise, output one word “INCORRECT” (without quotes).

Time Limits.

language	C	C++	Java	Python	C#	Haskell	JavaScript	Ruby	Scala
time (sec)	2	2	3	10	3	4	10	10	6

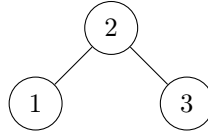
Memory Limit. 512MB.

Sample 1.

Input:

```
3
2 1 2
1 -1 -1
3 -1 -1
```

Output:

CORRECT

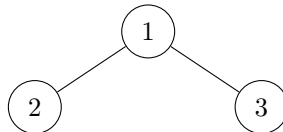
Left child of the root has key 1, right child of the root has key 3, root has key 2, so everything to the left is smaller, everything to the right is bigger.

Sample 2.

Input:

```
3
1 1 2
2 -1 -1
3 -1 -1
```

Output:

INCORRECT

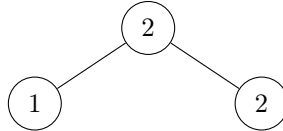
The left child of the root must have smaller key than the root.

Sample 3.

Input:

```
3
2 1 2
1 -1 -1
2 -1 -1
```

Output:

CORRECT

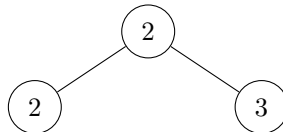
Duplicate keys are allowed, and they should always be in the right subtree of the first duplicated element.

Sample 4.

Input:

```
3
2 1 2
2 -1 -1
3 -1 -1
```

Output:

INCORRECT

The key of the left child of the root must be strictly smaller than the key of the root.

Sample 5.

Input:

```
0
```

Output:

CORRECT

Empty tree is considered correct.

Sample 6.

Input:

```
1
2147483647 -1 -1
```

Output:

```
CORRECT
```

Explanation:



The maximum possible value of the 32-bit integer type is allowed as key in the tree.

Sample 7.

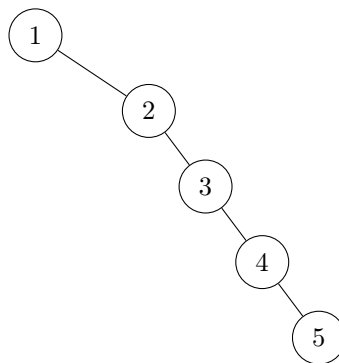
Input:

```
5
1 -1 1
2 -1 2
3 -1 3
4 -1 4
5 -1 -1
```

Output:

```
CORRECT
```

Explanation:



The tree doesn't have to be balanced. We only need to test whether it is a correct binary search tree, which the tree in this example is.

Sample 8.

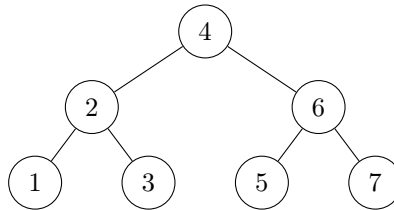
Input:

```
7
4 1 2
2 3 4
6 5 6
1 -1 -1
3 -1 -1
5 -1 -1
7 -1 -1
```

Output:

```
CORRECT
```

Explanation:



This is a full binary tree, and the property of the binary search tree is satisfied in every node.

Starter Files

The starter solutions for this problem read the input data from the standard input, pass it to a blank procedure, and then write the result to the standard output. You are supposed to implement your algorithm in this blank procedure if you are using **C++**, **Java**, or **Python3**. For other programming languages, you need to implement a solution from scratch. Filename: `is_bst_hard`

What to Do

Try to adapt the algorithm from the previous problem to the case when duplicate keys are allowed, and beware of integer overflow!

Need Help?

Ask a question or see the questions asked by other learners at [this forum thread](#).