1 Binary tree traversals

Problem Introduction

In this problem you will implement in-order, pre-order and post-order traversals of a binary tree. These traversals were defined in the **Basic Data Structures** module's lecture on **tree traversals**, but it is very useful to practice implementing them to understand binary search trees better.

Problem Description

Task. You are given a rooted binary tree. Build and output its in-order, pre-order and post-order traversals.

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 $0, 1, \ldots, n-1$ 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. $1 \le n \le 10^5$; $0 \le key_i \le 10^9$; $-1 \le left_i, right_i \le n-1$. It is guaranteed that the input represents a valid binary tree. In particular, if $left_i \ne -1$ and $right_i \ne -1$, then $left_i \ne right_i$. Also, a vertex cannot be a child of two different vertices. Also, each vertex is a descendant of the root vertex.

Output Format. Print three lines. The first line should contain the keys of the vertices in the in-order traversal of the tree. The second line should contain the keys of the vertices in the pre-order traversal of the tree. The third line should contain the keys of the vertices in the post-order traversal of the tree.

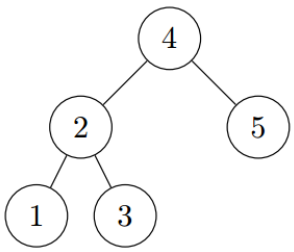
Sample 1.

Input:

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

Output:

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



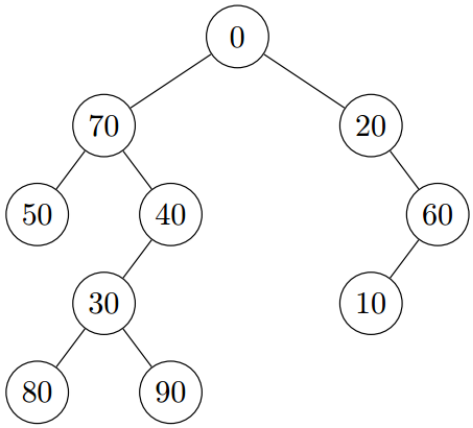
Sample 2.

Input:

```
10
0 7 2
10 -1 -1
20 -1 6
30 8 9
40 3 -1
50 -1 -1
60 1 -1
70 5 4
80 -1 -1
90 -1 -1
```

Output:

```
50 70 80 30 90 40 0 20 10 60
0 70 50 40 30 80 90 20 60 10
50 80 90 30 40 70 10 60 20 0
```



Starter Files

There are starter solutions for `C++`, `Java`, and `Python3`, and if you use other languages, you need to implement solution from scratch. Starter solutions read the input, define the methods to compute different traversals of the binary tree and write the output. You need to implement the traversal methods.

What To Do

Implement the traversal algorithms from the lectures. Note that the tree can be very deep in this problem, so you should be careful to avoid stack overflow problems if you're using recursion, and definitely test your solution on a tree with the maximum possible height.

2 Is it a binary search tree?

Problem Introduction

In this problem you are going to test whether a binary search tree data structure from some programming language library was implemented correctly. There is already a program that plays with this data structure by inserting, removing, searching integers in the data structure and outputs the state of the internal binary tree after each operation. Now you need to test whether the given binary is indeed a correct binary search tree. In other words, you want to ensure that you can search for integers in this binary tree using binary search through the tree, and you will always get the correct result: if the integer is in the tree, you will find it, otherwise you will not.

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. The definition of the binary search tree 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 strictly greater than x. 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 $0, 1, \ldots, n - 1$ 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. $0 \le n \le 10^5$; $-2^{31} \le key_i \le 2^{31} - 1$; $-1 \le left_i, right_i \le n - 1$. It is guaranteed that the input represents a valid binary tree. In particular, if $left_i \ne -1$ and $right_i \ne -1$, then $left_i \ne right_i$. Also, a vertex cannot be a child of two different vertices. Also, each vertex is a descendant of the root vertex. All keys in the input will be different.

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).

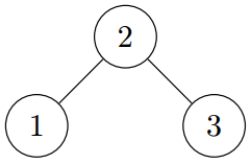
Sample 1.

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 2, root has key 2, so everything to the left is smaller, everything to the right is bigger.

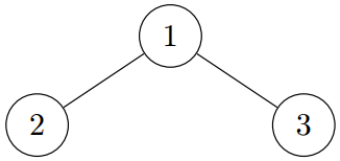
Sample 2.

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:

```
0
```

Output:

```
CORRECT
```

Empty tree is considered correct.

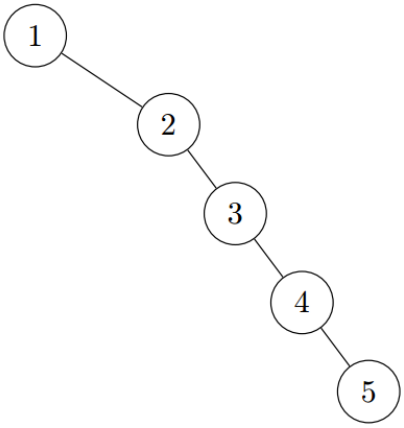
Sample 4.

Input:

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

Output:

```
CORRECT
```



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.

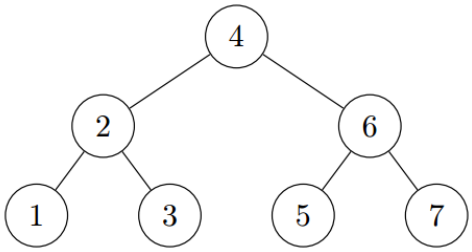
Sample 5.

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
```



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

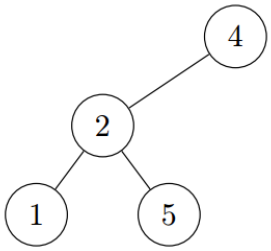
Sample 6.

Input:

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

Output:

```
INCORRECT
```



Node 5 is in the left subtree of the root, but it is bigger than the key 4 in the root.

Starter Files

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

What To Do

Testing the binary search tree condition for each node and every other node in its subtree will be too slow. You should come up with a faster algorithm.

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 $0, 1, \ldots, n - 1$ 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. $0 \le n \le 10^5$; $-2^{31} \le key_i \le 2^{31} - 1$; $-1 \le left_i, right_i \le n - 1$. It is guaranteed that the input represents a valid binary tree. In particular, if $left_i \ne -1$ and $right_i \ne -1$, then $left_i \ne right_i$. 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).

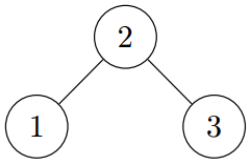
Sample 1.

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 2, so everything to the left is smaller, everything to the right is bigger.

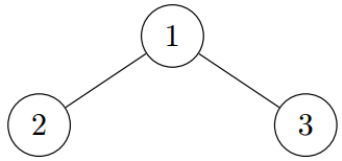
Sample 2.

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.

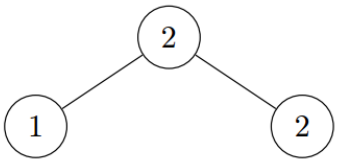
Sample 3.

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.

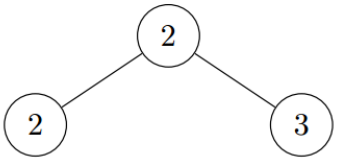
Sample 4.

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.

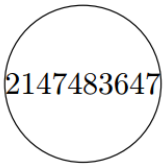
Sample 6.

Input:

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

Output:

```
CORRECT
```



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

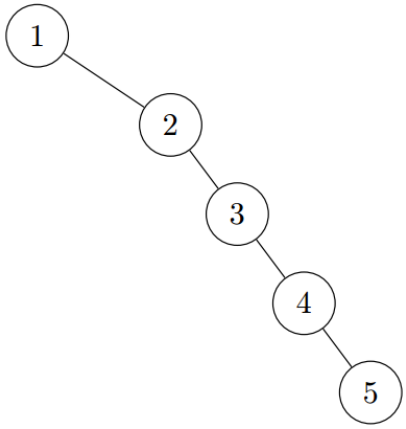
Sample 7.

Input:

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

Output:

```
CORRECT
```



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.

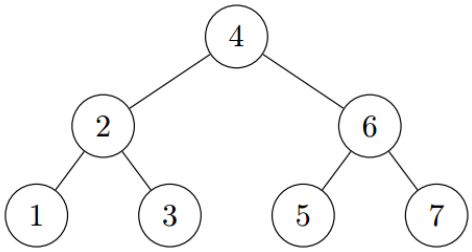
Sample 8.

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
```



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 form the standard input, pass it to a blank procedure, and then write the result to the standard output. You are supposed to implement you algorithm in this blank procedure if you are using `C++`, `Java`, and `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!