Problem A. Cartesian Tree

Input file:	`tree.in`
Output file:	`tree.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

Let us consider a special type of binary search trees, called *cartesian trees*. Recall that a binary search tree is a rooted ordered binary tree, such that for its every node x the following condition is satisfied: each node in its left subtree has the key less than the key of x, and each node in its right subtree has the key greater than the key of x.

That is, if we denote the left subtree of the node x by $L(x)$, its right subtree by $R(x)$ and its key by k_x, for each node x we will have

- if $y \in L(x)$ then $k_y < k_x$
- if $z \in R(x)$ then $k_z > k_x$

The binary search tree is called *cartesian* if its every node x in addition to the main key k_x also has an auxiliary key that we will denote by a_x, and for these keys the *heap condition* is satisfied, that is

- if y is the parent of x then $a_y < a_x$

Thus a cartesian tree is a binary rooted ordered tree, such that each of its nodes has a pair of two keys (k, a) and three conditions described are satisfied.

Given a set of pairs, construct a cartesian tree out of them, or detect that it is not possible.

Input

The first line of the input file contains an integer number N — the number of pairs you should build cartesian tree out of ($1 \leq N \leq 50\,000$). The following N lines contain two integer numbers each — given pairs (k_i, a_i). For each pair $|k_i|, |a_i| \leq 30\,000$. All main keys and all auxiliary keys are different, i.e. $k_i \neq k_j$ and $a_i \neq a_j$ for each $i \neq j$.

Output

On the first line of the output file print `YES` if it is possible to build a cartesian tree out of given pairs or `NO` if it is not. If the answer is positive, output the tree itself in the following N lines. Let the nodes be numbered from 1 to N corresponding to pairs they contain as these pairs are given in the input file. For each node output three numbers — its parent, its left child and its right child. If the node has no parent or no corresponding child, output `0` instead.

If there are several possible trees, output any one.

Example

`tree.in`	`tree.out`
`7`	`YES`
`5 4`	`2 3 6`
`2 2`	`0 5 1`
`3 9`	`1 0 7`
`0 5`	`5 0 0`
`1 3`	`2 4 0`
`6 6`	`1 0 0`
`4 11`	`3 0 0`