Problem A. DFS Order

Input file:	standard input
Output file:	standard output
Time limit:	3 seconds
Memory limit:	1024 megabytes

Prof. Pang has a rooted tree which is rooted at 1 with n nodes. These n nodes are numbered from 1 to n.

Now he wants to start the depth-first search at the root. He wonders for each node v, what is the minimum and the maximum position it can appear in the **depth-first search order**. The depth-first search order is the order of nodes visited during the depth-first search. A node appears in the j-th ($1 \le j \le n$) position in this order means it is visited after $j-1$ other nodes. Because sons of a node can be iterated in arbitrary order, multiple possible depth-first orders exist. Prof. Pang wants to know for each node v, what are the minimum value and the maximum value of j such that v appears in the j-th position.

Following is a pseudo-code for the depth-first search on a rooted tree. After its execution, `dfs_order` is the depth-first search order.

```
let dfs_order be an empty list

def dfs(vertex x):
    append x to the end of dfs_order.
    for (each son y of x): // sons can be iterated in arbitrary order.
        dfs(y)

dfs(root)
```

Input

The first line contains a single integer T ($1 \le T \le 10^6$) denoting the number of test cases.

For each test case, the first line contains an integer n ($1 \le n \le 10^5$). Each of the next $n-1$ lines contains two integers x and y, indicating node x is node y's parent ($1 \le x, y \le n$). These edges form a tree rooted at 1.

It is guaranteed that the sum of n over all test cases is no more than 10^6.

Output

For each test case, print n lines. The i-th line contains two integers denoting the minimum and the maximum position node i can appear in the depth-first search order.

Example

standard input	standard output
2 4 1 2 2 3 3 4 5 1 2 2 3 2 4 1 5	1 1 2 2 3 3 4 4 1 1 2 3 3 5 3 5 2 5