D. Water Tree

time limit per test: 4 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Mad scientist Mike has constructed a rooted tree, which consists of n vertices. Each vertex is a reservoir which can be either empty or filled with water.

The vertices of the tree are numbered from 1 to n with the root at vertex 1. For each vertex, the reservoirs of its children are located below the reservoir of this vertex, and the vertex is connected with each of the children by a pipe through which water can flow downwards.

Mike wants to do the following operations with the tree:

1. Fill vertex v with water. Then v and all its children are filled with water.
2. Empty vertex v. Then v and all its ancestors are emptied.
3. Determine whether vertex v is filled with water at the moment.

Initially all vertices of the tree are empty.
Mike has already compiled a full list of operations that he wants to perform in order. Before experimenting with the tree Mike decided to run the list through a simulation. Help Mike determine what results will he get after performing all the operations.

Input

The first line of the input contains an integer n ($1 \le n \le 500000$) — the number of vertices in the tree. Each of the following $n - 1$ lines contains two space-separated numbers a_i, b_i ($1 \le a_i, b_i \le n$, $a_i \ne b_i$) — the edges of the tree.

The next line contains a number q ($1 \le q \le 500000$) — the number of operations to perform. Each of the following q lines contains two space-separated numbers c_i ($1 \le c_i \le 3$), v_i ($1 \le v_i \le n$), where c_i is the operation type (according to the numbering given in the statement), and v_i is the vertex on which the operation is performed.

It is guaranteed that the given graph is a tree.

Output

For each type 3 operation print 1 on a separate line if the vertex is full, and 0 if the vertex is empty. Print the answers to queries in the order in which the queries are given in the input.

Examples

input
5 1 2 5 1 2 3 4 2 12 1 1 2 3 3 1 3 2 3 3 3 4 1 2 2 4 3 1 3 3 3 4 3 5

output

```
0
0
0
1
0
1
0
1
```