Extracting Weights

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	1024 megabytes

"Emily is very busy"... I feel like Emily is actually pretty free.

—Grilled-chicken

This is an interactive problem.

Emily has a tree containing n nodes, where the weight of the node numbered i is w_i. Node 1 is the root node, and the weight of the root node is 0.

Emily wants you to guess the weight of each node. Specifically, you can make at most n queries, with the i-th query containing two integers u_i and v_i. If the simple path from u_i to v_i contains exactly k edges, Emily will tell you the bitwise XOR[†] of the weights of all the nodes on the simple path from node u_i to node v_i (including the endpoints). Otherwise, Emily will respond with "-1", indicating that she does not want to answer that question. Emily is very busy, so she will not start answering until you have asked all your questions.

Of course, you may not be able to guess the weights of all nodes with no more than n queries for some cases. In such cases, you should not make any queries and instead directly tell Emily that it is impossible.

[†]The bitwise XOR operation refers to the addition of each bit of two binary numbers under modulo 2. For example: $(0011)_2 \oplus (0101)_2 = (0110)_2$.

Input

There is only one test case in each test file.

The first line contains two integers n and k ($2 \le n \le 250$, $1 \le k \le n-1$), representing the number of nodes in the tree and a parameter mentioned in the query.

The next $n-1$ lines each contain two integers x_i and y_i ($1 \le x_i, y_i \le n$), indicating that there is an edge between nodes x_i and y_i.

It is guaranteed that all edges form a tree, and the correct answer satisfies $0 \le w_i < 2^{30}$.

Interaction Protocol

First, you need to determine whether it is possible to guess the weights of all nodes within n queries. If not, your program should output "`No`" and exit immediately. Otherwise, your program should output "`Yes`" and proceed to querying. You can output the answer in any case (upper or lower). For example, the strings "`yEs`", "`yes`", "`Yes`", and "`YES`" will be recognized as positive responses.

To make a query, output **all** your q queries **at once** in the format "`?` q u_1 v_1 u_2 v_2 ... u_q v_q" ($1 \le q \le n$, $1 \le u_i, v_i \le n$). After flushing your output, your program should read a line containing q integers. The i-th integer represents the response to the i-th query.

To guess the weights, output your guess in the format "`!` w_2 w_3 ... w_n" ($0 \le w_i < 2^{30}$). After flushing your output, your program should exit immediately.

Note that the answer for each test case is pre-determined. That is, the interactor is not adaptive. Also, note that your guess does not count as a query.

To flush your output, you can use:

- `fflush(stdout)` (if you use `printf`) or `cout.flush()` (if you use `cout`) in C and C++.

- `System.out.flush()` in Java and Kotlin.
- `stdout.flush()` in Python.

Examples

standard input	standard output
4 1 1 2 2 3 2 4 1 3 2	 YES ? 3 1 2 2 3 2 4 ! 1 2 3
5 2 1 2 2 3 3 4 3 5 1 2 3 4	 YES ? 4 1 3 2 4 2 5 4 5 ! 4 5 3 2
6 2 1 2 2 3 3 4 4 5 4 6	 NO