Dependency Hell

Dependency hell, as Wikipedia wrote, "is the frustration of some software users who have installed software packages which have dependencies on specific versions of other software packages". In other words, a software may requires some other softwares or libraries to be installed prior installing it, and in turn, those other softwares may also require some other softwares/libraries to be installed first, and so on. There are many possible solutions to mitigate this dependency hell problem, but we are not interested in those at this moment.

Supposed we have an empty system with no pre-installed software. A particular software (named HELL) is to be installed, and its dependency resembles a rooted-tree structure, i.e. each software is required by at most one other software, and there is no circular dependency. Needless to say, a software can only be installed if and only if all of its required softwares are already installed.

Note that the system can only install one software at a time (no parallelism). How many valid installation sequences are there such that HELL can be properly installed? An installation sequence is considered as valid if and only if before a software being installed, all of its required softwares are already installed.

Consider the following examples where there are 5 softwares numbered from 1 to 5 (HELL is numbered 1). Let's say, to install HELL (software 1), software 2 and 3 are required; to install software 2, software 4 and 5 are required. Then, all of the valid installation sequences are as follow:

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

Therefore, in this example, there are 8 valid installation sequences.

As the output can be very large, you should modulo the output by 1,000,000,007.

Input

Input begins with an integer: T (1 ≤ T ≤ 20) denoting the number of cases.

Each case contains the following input block: Each case begins with an integer N (1 ≤ N ≤ 100,000) denoting the number of softwares in a dependency tree. The next N-1 lines, each contains two integer a_i b_i (1 ≤ a_i, b_i ≤ 100,000) denoting that software a_i requires b_i, i.e. before a_i is installed, b_i should already been installed. HELL is always denoted as software 1. You may safely assume that the given dependency resembles a connected rooted-tree with N nodes, and 1 (HELL) as its root.

The sum of all N in the input is no larger than 200,000.

Output

For each case, output in a line "`Case #X: Y`" where `X` is the case number (starts from 1) and `Y` is the output for the respective case modulo 1,000,000,007.

Examples

input **Example #1**

```
4
5
1 2
1 3
2 4
2 5
4
1 2
1 3
1 4
3
3 2
1 3
7
1 4
4 3
4 7
```

```
3 2
3 5
3 6
```

output

```
Case #1: 8
Case #2: 6
Case #3: 1
Case #4: 30
```

explanation

Case 1: This is the example given in the problem statement.

Case 2: As software 2, 3, and 4 are independent to each other, then there are 3! = 1 * 2 * 3 = 6 valid installation sequences.

Case 3: The software dependency is a straight line (of 1-3-2), thus, there is only one valid installation sequence.