If Chain

Consider the following code:

```
if (a)
if (b)
if (c)
foo();
```

where *a,b* and *c* are boolean variables. If we run this code in C++, the function *foo()* is called if and only if all three variables are true. However, recently a new language - C-- - is being developed. In this language, when an *if()* evaluates to false, only the statement directly following it is not executed; for example, if *a* was false, the program would jump from *if(a)* to *if(c)*.

Using this convention, there are 5 different possible assignments of truth values to the variables *a*,*b*,*c* which end up calling *foo()*. Considering *a*,*b*,*c* as three bits in that order, they are 111, 101, 100, 011 and 001.

Given **n** boolean variables within a chain of **m** *if()*'s, where the variables within one *if()* are separated using **logical or**, count the number of ways to assign truth values to them which end up calling the function *foo()* (the call to *foo()* is after the last *if()*).

Input

The first line of the input is the number of test cases **1 ≤ T ≤ 30**. **T** test cases follow.

The first line of each test case contains two nonnegative integers $\mathbf{n \le 10^5}$ - the number of boolean variables (they are numbered **1** through **n**) - and $\mathbf{m \le 10^5}$ - the number of *if()*'s. **m** lines follow, the **i**-th line describing the **i**-th *if()*. The first integer in each line is a positive integer $\mathbf{k_i}$ - the number of variables in the **i**-th *if()* (implicitly separated by the **logical or** operator) - followed by $\mathbf{k_i}$ positive integers in the range **[1,n]**: the variables in the **i**-th *if()*. Not all variables need necessarily appear within the *if()* chain, and the variables within one *if()* need not be distinct.

The sum of $\mathbf{k_i}$ within a test case will not exceed $\mathbf{5*10^5}$. Additionally, the sum of **n**,**m** and $\mathbf{k_i}$ within a single input file will not exceed $\mathbf{2*10^6}$**.**

The input is quite large - make sure to read it efficiently.

Output

For each case, output the string "Case **x**: **y**" in a single line, where **x** is the number of the test case, starting from 1, and **y** is the number of ways of assigning truth values to the **n** boolean variables (out of $\mathbf{2^n}$), which when run in C-- end up calling *foo()*, modulo $\mathbf{10^9+9}$.

Example

Input:

```
2
3 3
1 1
```

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

Output:

```
Case 1: 5
Case 2: 24
```