A company is expecting bad weather to be coming. If the weather is bad enough, the president of the company will close the company for the day. In order to inform all the employees of the company, the president will inform all of the employees working directly under her by phone, and they will inform the employees directly under them by phone, etc. It takes one minute for each phone call to be made, and each person can only call one other person at a time. Each employee will call a person directly under her as soon as she is informed, and she will keep calling people under her every minute until she is done. The employees are not well aware of the structure under them except for the employees directly under them. What is the most number of minutes it could take for everybody in the company to be informed?

**Input Specification**

The input consists of a number $k$ indicating how many cases there are, followed by the cases. Each case begins with a number $n$ which indicates how many people work in the company. The employees are numbered from 0 to $n-1$. The number $n$ is followed by pairs of numbers $i$ and $j$ indicating that $i$ is the boss of $j$. For simplicity, $i$ is always smaller than $j$. Note that nobody is allowed to have two bosses.

**Output Specification**

Your program should print one line of output for each test case. For each case, print out the most number of minutes it could take for all employees to be informed.

<table>
<thead>
<tr>
<th>Sample Input</th>
<th>Sample Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>3 0 1 0 2</td>
<td>4</td>
</tr>
<tr>
<td>5 0 1 0 2 1 3 1 4</td>
<td>5</td>
</tr>
<tr>
<td>7 0 1 0 2 1 3 1 4 1 5 2 6</td>
<td>6</td>
</tr>
<tr>
<td>10 0 2 0 3 2 4 2 5 2 6 3 7 0 1 1 8 8 9</td>
<td></td>
</tr>
</tbody>
</table>

Note: In the last example input, the worst case is that employee 0 calls 3 at time 0. At time 1, 0 calls 1 and 3 calls 7. At time 2, 0 calls 2 and 1 calls 8. At time 3, 8 calls 9 and 2 calls 4. At time 4, 2 calls 5. At time 5, 2 calls 6. They will all have been informed after 6 minutes.
A manager in charge of distribution for a factory of custom parts needs to maximize the space used in his delivery boxes. He has available boxes of multiple sizes that hold multiples of 100 parts (e.g. boxes that can hold 100, 200, 300 parts, etc). But clients often order a random number of parts. To help him, we will write a prototype program to check if for the set of orders placed in one day, it is possible to fill one or more boxes to maximum capacity. Clients cannot order more than 100 parts at once.

**Input Specification**

The input consists of a number k indicating how many cases there are, followed by the cases. Each case begins with a number of orders, n, followed by n positive integers m that indicate the number of parts in each order, with 0 < m ≤ 100.

**Output Specification**

Your program should print one line of output for each test case. For each case, print “Yes” if some subset of the n orders sums to a number that can fill exactly one or more boxes, and print “No” if there is no such subset.

<table>
<thead>
<tr>
<th>Sample Input</th>
<th>Sample Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 4 76 1 76 48 3 78 74 48 4 76 1 75 50</td>
<td>Yes Yes No</td>
</tr>
</tbody>
</table>
C: The Proper Key

Many people think that Tetris was invented by two Russian programmers. But that is not the whole truth. The idea of the game is very old – even the Egyptians had something similar. But they did not use it as a game. Instead, it was used as a very complicated lock. The lock was made of wood and consisted of a large number of square fields, laid out in regular rows and columns. Each field was either completely filled with wood, or empty. The key for this lock was two-dimensional and it was made by joining square parts of the same size as the fields of the lock. So they had a 2D lock and 2D key that could be inserted into the lock from the top. The key was designed so that it was not possible to move it upwards. It could only fall down and it could slide sideways – exactly like a Tetris game. The only difference is that the key could not be rotated. Rotation in Tetris is really a Russian invention.

You need to help a group of archeologists who have found several keys to a pyramid and one of them belongs to the lock with a very high probability. They need to try them out and find which one to use. Because it is too time-consuming to try all of them, it is better to begin with those keys that may be inserted deeper into the lock. Your program should determine how deep a given key can be inserted into a given lock.

Input Specification

The input consists of $T$ test cases. The number $T$ is given on the first line of the input. Each test case begins with a line containing two integers $R$ and $C$ ($1 \leq R, C \leq 100$) indicating the key size. Then exactly $R$ rows follow, each containing $C$ characters. Each character is either a hash mark (#) or a period (.). A hash mark represents one square field made of wood; a period is an empty field. The wooden fields are always connected, i.e., the whole key is made of one piece. Moreover, the key remains connected even if we cut off an arbitrary number of rows from its top. There is always at least one non-empty field in the top-most and bottom-most rows and the left-most and right-most columns.

After the key description, there is a line containing two integers $D$ and $W$ ($1 \leq D \leq 10000$, $1 \leq W \leq 1000$). The number $W$ is the lock width, and $D$ is its depth. The next $D$ lines contain $W$ characters each. The character may be either a hash mark (representing the wood) or a period (the free space).

Output Specification

Your program should print one line of output for each test case. If it is possible to move the key through the whole lock and take it away at the bottom side, output the sentence “The key can fall through”. Otherwise the line should contain the statement “The key falls to depth X.”. Replace $X$ with the maximum depth to which the key can be inserted by moving it down and sliding it to the left or right only. The depth is measured as the distance between the bottom side of the key and the top side of the lock.
<table>
<thead>
<tr>
<th>Sample Input</th>
<th>Sample Output</th>
</tr>
</thead>
</table>
| 4
2 4
#.##
###.
3 6
#....#
#.....#
#:###
2 3
##.
.##
2 7
#.#.#.#
#.#.#.#
1 1
# 1 10
###....###
3 2
##
.#
.#
1 5
#.#.# |
| The key falls to depth 2.
The key falls to depth 0.
The key can fall through.
The key falls to depth 2. |
Consider the following sequence of numbers starting with a positive integer \( n \), and that is generated by the following:

- if \( n \) is even, divide it by 2 to get \( n' \)
- if \( n \) is odd, multiply it by 3 and add 1 to get \( n' \)

Then take \( n' \) as the next number in the sequence, and continue the process. For example:

\( n = 5 \) gives the sequence: \( 5, 16, 8, 4, 2, 1, 4, 2, 1, \ldots \)
\( n = 11 \) gives the sequence: \( 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, \ldots \)

The numbers in these sequences go up and down, like a yoyo. An interesting (but unproven) conjecture is that, for any positive integer \( n \), the sequence will always end in the repeating cycle: \( 4, 2, 1, 4, 2, 1, \ldots \) So consider that when \( n = 1 \), the sequence has ended.

Write a program to determine the largest value in the sequence for a given \( n \).

**Input Specification**

The first line of input contains a single integer \( P \), \( 1 \leq P \leq 100000 \), which is the number of data sets that follow. Each data set should be processed identically and independently.

Each data set consists of a single line of input consisting of two space separated decimal integers. The first integer is the data set number. The second integer is \( n \), \( 1 \leq n \leq 100,000 \), which is the starting value.

**Output Specification**

For each data set there is a single line of output consisting of the data set number, a single space, and the largest value in the sequence starting at and including \( n \).

<table>
<thead>
<tr>
<th>Sample Input</th>
<th>Sample Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 1</td>
<td>1 1</td>
</tr>
<tr>
<td>1 1</td>
<td>2 16</td>
</tr>
<tr>
<td>2 3</td>
<td>3 101248</td>
</tr>
<tr>
<td>3 9999</td>
<td>4 100000</td>
</tr>
<tr>
<td>4 100000</td>
<td></td>
</tr>
</tbody>
</table>
The members of a fraternity house throw their socks on the floor when they take them off every day. As a result, there are various piles of socks lying around the house. One day the fraternity members decide to pick up all the socks. They each search around the house for the sock piles. Every member of the house finds some subset of the piles (the subsets are not necessarily disjoint). Each member mentally pairs all of the socks he has found, without changing the piles, and announces the odds socks left over after pairing. For example, one house member may find piles 1, 3 and 5, and after pairing socks determine there is an odd red sock and an odd blue sock left over. Another member may find piles 2, 3 and 5 and after pairing socks determine that there is an odd red sock and an odd green sock left over. It may be possible that some house members have counted incorrectly or are lying about what they found. Your job is to determine if the results the house members give are consistent with each other.

**Input Specification**

The input consists of a number k indicating how many cases there will be, followed by the cases. Each case begins with a number m indicating how many piles of socks there are, a number n indicating how many colors of socks there are and a number p indicating how many members of the fraternity house there are. After that, there is a line of input for each fraternity member. The line begins with the number of piles the member examined and the number of odd socks that the member found. These two numbers are followed by a list of which piles the member examined, followed by which odd socks were left over. The piles are numbered 0 through n-1 and the colors of socks are numbered 0 through p-1.

**Output Specification**

Your program should print one line of output for each test case. For each case, print out whether the information is consistent or inconsistent.

<table>
<thead>
<tr>
<th>Sample Input</th>
<th>Sample Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>3&lt;br&gt;3 3 2&lt;br&gt;2 2 0 1 0 1&lt;br&gt;2 2 0 2 0 2&lt;br&gt;3 4 3&lt;br&gt;2 2 0 1 0 1&lt;br&gt;2 2 0 2 0 2&lt;br&gt;2 3 1 2 1 2 3&lt;br&gt;10 8 3&lt;br&gt;5 4 0 1 2 3 4 0 1 2 3&lt;br&gt;5 4 5 6 0 8 9 4 5 6 7&lt;br&gt;2 4 2 5 4 6 1 3</td>
<td>consistent&lt;br&gt;inconsistent&lt;br&gt;consistent</td>
</tr>
</tbody>
</table>
You would like to have a party and invite all of your friends. But some of your friends hate each other, so you
cannot invite both of any pair of enemies at the same time. So you want to know if it is possible to have two
parties, so that everyone can come to at least one party, and no pair of enemies will both be at the same
party. You must write a program which checks if this is possible.

**Input Specification**

The input consists of a number k indicating how many cases there will be, followed by the cases. Each case
begins with a number n and a number m. The number n will indicate how many pairs of enemies there are.
The number m will indicate how many people you want to invite to your party. This is followed by a set of n
pairs of integers, indicating pairs of enemies. Each integer will be from 0 to m-1, and refer to one of your
friends.

**Output Specification**

Your program should print one line of output for each test case. For each case, print “Yes” if it is possible to
invite all of your friends to at least one party, and print “No” otherwise.

<table>
<thead>
<tr>
<th>Sample Input</th>
<th>Sample Output</th>
</tr>
</thead>
</table>
| 3
4 4 0 1 3 2 2 1 0 3
3 3 0 1 2 1 2 0
2 4 0 1 2 3 | Yes
|               | No
|               | Yes |