## A. Subrectangle Guess

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Michael and Joe are playing a game. The game is played on a grid with $n$ rows and $m$ columns, **filled with distinct integers**. We denote the square on the $i$-th ($1 \le i \le n$) row and $j$-th ($1 \le j \le m$) column by $(i, j)$ and the number there by $a_{ij}$.

Michael starts by saying two numbers $h$ ($1 \le h \le n$) and $w$ ($1 \le w \le m$). Then Joe picks any $h \times w$ subrectangle of the board (without Michael seeing).

Formally, an $h \times w$ subrectangle starts at some square $(a, b)$ where $1 \le a \le n - h + 1$ and $1 \le b \le m - w + 1$. It contains all squares $(i, j)$ for $a \le i \le a + h - 1$ and $b \le j \le b + w - 1$.

| | | | |
|---|---|---|---|
| 2 | 12 | 6 | 10 |
| 3 | 15 | 16 | 4 |
| 1 | 13 | 8 | 11 |
| 14 | 7 | 9 | 5 |

Possible move by Joe if Michael says $3 \times 2$ (with maximum of $15$).

Finally, Michael has to guess the maximum number in the subrectangle. He wins if he gets it right.

Because Michael doesn't like big numbers, he wants the area of the chosen subrectangle (that is, $h \cdot w$), to be as small as possible, while still ensuring that he wins, not depending on Joe's choice. Help Michael out by finding this minimum possible area.

It can be shown that Michael can always choose $h, w$ for which he can ensure that he wins.

### Input

Each test contains multiple test cases. The first line contains the number of test cases $t$ ($1 \le t \le 20$). Description of the test cases follows.

The first line of each test case contains two integers $n$ and $m$ ($1 \le n, m \le 40$) — the size of the grid.

Each of the following $n$ lines contains $m$ integers. The $j$-th integer on the $i$-th line is $a_{ij}$ ($-10^9 \le a_{ij} \le 10^9$) — the element in the cell $(i, j)$.

It is guaranteed that all the numbers are **distinct** (that is, if $a_{i_1 j_1} = a_{i_2 j_2}$, then $i_1 = i_2, j_1 = j_2$).

### Output

For each test case print a single positive integer — the minimum possible area the subrectangle can have while still ensuring that Michael can guarantee the victory.

**Example**

**input**

```
3
1 1
3
4 4
2 12 6 10
3 15 16 4
1 13 8 11
14 7 9 5
2 3
-7 5 2
0 8 -3
```

**output**

```
1
9
4
```

### Note

In the first test case, the grid is $1 \times 1$, so the only possible choice for $h, w$ is $h = 1, w = 1$, giving an area of $h \cdot w = 1$.

The grid from the second test case is drawn in the statement. It can be shown that with $h = 3, w = 3$ Michael can guarantee the victory and that any choice with $h \cdot w \le 8$ doesn't.

## B. Circle Game

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Mike and Joe are playing a game with some stones. Specifically, they have $n$ piles of stones of sizes $a_1, a_2, \ldots, a_n$. These piles are arranged in a circle.

The game goes as follows. Players take turns removing some positive number of stones from a pile in clockwise order starting from pile $1$. Formally, if a player removed stones from pile $i$ on a turn, the other player removes stones from pile $((i \bmod n) + 1)$ on the next turn.

If a player cannot remove any stones on their turn (because the pile is empty), they lose. Mike goes first.

If Mike and Joe play optimally, who will win?

### Input

Each test contains multiple test cases. The first line contains the number of test cases $t$ ($1 \le t \le 1000$). Description of the test cases follows.

The first line of each test case contains a single integer $n$ ($1 \le n \le 50$) — the number of piles.

The second line contains $n$ integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) — the size of the piles.

### Output

For each test case print the winner of the game, either "Mike" or "Joe" on its own line (without quotes).

**Example**

**input**

```
2
1
37
2
100 100
```

**output**

```
Mike
Joe
```

### Note

In the first test case, Mike just takes all $37$ stones on his first turn.

In the second test case, Joe can just copy Mike's moves every time. Since Mike went first, he will hit $0$ on the first pile one move before Joe does so on the second pile.

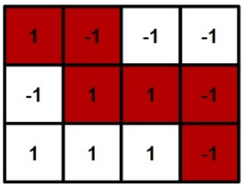
## C. Zero Path

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a grid with $n$ rows and $m$ columns. We denote the square on the $i$-th ($1 \le i \le n$) row and $j$-th ($1 \le j \le m$) column by $(i, j)$ and the number there by $a_{ij}$. All numbers are equal to $1$ or to $-1$.

You start from the square $(1, 1)$ and can move one square down or one square to the right at a time. In the end, you want to end up at the square $(n, m)$.

Is it possible to move in such a way so that the sum of the values written in all the visited cells (including $a_{11}$ and $a_{nm}$) is $0$?




### Input

Each test contains multiple test cases. The first line contains the number of test cases $t$ ($1 \le t \le 10^4$). Description of the test cases follows.

The first line of each test case contains two integers $n$ and $m$ ($1 \le n, m \le 1000$) — the size of the grid.

Each of the following $n$ lines contains $m$ integers. The $j$-th integer on the $i$-th line is $a_{ij}$ ($a_{ij} = 1$ or $-1$) — the element in the cell $(i, j)$.

It is guaranteed that the sum of $n \cdot m$ over all test cases does not exceed $10^6$.

### Output

For each test case, print "YES" if there exists a path from the top left to the bottom right that adds up to $0$, and "NO" otherwise. You can output each letter in any case.

**Example**

**input**

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

**output**

```
NO
YES
YES
YES
NO
```

### Note

One possible path for the fourth test case is given in the picture in the statement.

## D1. Tree Queries (Easy Version)

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

**The only difference between this problem and D2 is the bound on the size of the tree.**

You are given an unrooted tree with $n$ vertices. There is some hidden vertex $x$ in that tree that you are trying to find.

To do this, you may ask $k$ queries $v_1, v_2, \ldots, v_k$ where the $v_i$ are vertices in the tree. After you are finished asking all of the queries, you are given $k$ numbers $d_1, d_2, \ldots, d_k$, where $d_i$ is the number of edges on the shortest path between $v_i$ and $x$. Note that you know which distance corresponds to which query.

What is the minimum $k$ such that there exists some queries $v_1, v_2, \ldots, v_k$ that let you always uniquely identify $x$ (no matter what $x$ is).

Note that you don't actually need to output these queries.

### Input

Each test contains multiple test cases. The first line contains the number of test cases $t$ ($1 \le t \le 100$). Description of the test cases follows.

The first line of each test case contains a single integer $n$ ($1 \le n \le 2000$) — the number of vertices in the tree.

Each of the next $n - 1$ lines contains two integers $x$ and $y$ ($1 \le x, y \le n$), meaning there is an edges between vertices $x$ and $y$ in the tree.

It is guaranteed that the given edges form a tree.

It is guaranteed that the sum of $n$ over all test cases does not exceed $2000$.

### Output

For each test case print a single nonnegative integer, the minimum number of queries you need, on its own line.

**Example**

**input**

```
3
1
2
1 2
10
2 4
2 1
5 7
3 10
8 6
6 1
1 3
4 7
9 6
```

**output**

```
0
```

```
1
2
```

**Note**

In the first test case, there is only one vertex, so you don't need any queries.

In the second test case, you can ask a single query about the node $1$. Then, if $x = 1$, you will get $0$, otherwise you will get $1$.

## D2. Tree Queries (Hard Version)

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

**The only difference between this problem and D1 is the bound on the size of the tree.**

You are given an unrooted tree with $n$ vertices. There is some hidden vertex $x$ in that tree that you are trying to find.

To do this, you may ask $k$ queries $v_1, v_2, \ldots, v_k$ where the $v_i$ are vertices in the tree. After you are finished asking all of the queries, you are given $k$ numbers $d_1, d_2, \ldots, d_k$, where $d_i$ is the number of edges on the shortest path between $v_i$ and $x$. Note that you know which distance corresponds to which query.

What is the minimum $k$ such that there exists some queries $v_1, v_2, \ldots, v_k$ that let you always uniquely identify $x$ (no matter what $x$ is).

Note that you don't actually need to output these queries.

**Input**

Each test contains multiple test cases. The first line contains the number of test cases $t$ ($1 \le t \le 10^4$). Description of the test cases follows.

The first line of each test case contains a single integer $n$ ($1 \le n \le 2 \cdot 10^5$) — the number of vertices in the tree.

Each of the next $n - 1$ lines contains two integers $x$ and $y$ ($1 \le x, y \le n$), meaning there is an edges between vertices $x$ and $y$ in the tree.

It is guaranteed that the given edges form a tree.

It is guaranteed that the sum of $n$ over all test cases does not exceed $2 \cdot 10^5$.

**Output**

For each test case print a single nonnegative integer, the minimum number of queries you need, on its own line.

**Example**

**input**

```
3
1
2
1 2
10
2 4
2 1
5 7
3 10
8 6
6 1
1 3
4 7
9 6
```

**output**

```
0
1
2
```

**Note**

In the first test case, there is only one vertex, so you don't need any queries.

In the second test case, you can ask a single query about the node $1$. Then, if $x = 1$, you will get $0$, otherwise you will get $1$.

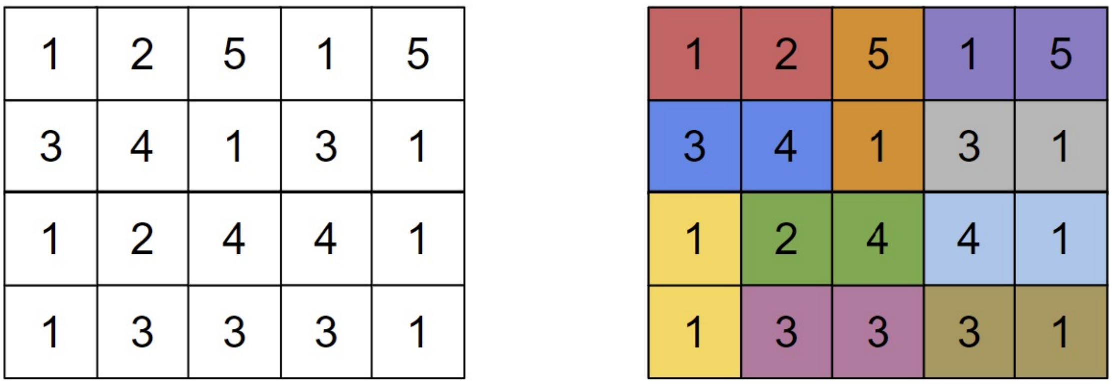
## E. Ambiguous Dominoes

time limit per test: 8 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Polycarp and Monocarp are both solving the same puzzle with dominoes. They are given the same set of $n$ dominoes, the $i$-th of which contains two numbers $x_i$ and $y_i$. They are also both given the same $m$ by $k$ grid of values $a_{ij}$ such that $m \cdot k = 2n$.

The puzzle asks them to place the $n$ dominoes on the grid in such a way that none of them overlap, and the values on each domino match the $a_{ij}$ values that domino covers. Dominoes can be rotated arbitrarily before being placed on the grid, so the domino $(x_i, y_i)$ is equivalent to the domino $(y_i, x_i)$.

They have both solved the puzzle, and compared their answers, but noticed that not only did their solutions not match, but none of the $n$ dominoes were in the same location in both solutions! Formally, if two squares were covered by the same domino in Polycarp's solution, they were covered by different dominoes in Monocarp's solution. The diagram below shows one potential $a$ grid, along with the two players' solutions.




Polycarp and Monocarp remember the set of dominoes they started with, but they have lost the grid $a$. Help them reconstruct one possible grid $a$, along with both of their solutions, or determine that no such grid exists.

**Input**

The first line contains a single integer $n$ ($1 \le n \le 3 \cdot 10^5$).

The $i$-th of the next $n$ lines contains two integers $x_i$ and $y_i$ ($1 \le x_i, y_i \le 2n$).

**Output**

If there is no solution, print a single integer $-1$.

Otherwise, print $m$ and $k$, the height and width of the puzzle grid, on the first line of output. These should satisfy $m \cdot k = 2n$.

The $i$-th of the next $m$ lines should contain $k$ integers, the $j$-th of which is $a_{ij}$.

The next $m$ lines describe Polycarp's solution. Print $m$ lines of $k$ characters each. For each square, if it is covered by the upper half of a domino in Polycarp's solution, it should contain a "U". Similarly, if it is covered by the bottom, left, or right half of a domino, it should contain "D", "L", or "R", respectively.

The next $m$ lines should describe Monocarp's solution, in the same format as Polycarp's solution.

If there are multiple answers, print any.

**Examples**

**input**

```
1
1 2
```

**output**

```
-1
```

**input**

```
2
1 1
1 2
```

**output**

```
2 2

2 1
1 1

LR
LR

UU
DD
```

**input**

```
10
1 3
1 1
2 1
3 4
1 5
1 5
3 1
2 4
3 3
4 1
```

**output**

```
4 5

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

LRULR
LRDLR
ULRLR
DLRLR

UULRU
DDUUD
LRDDU
LRLRD
```

### Note

Extra blank lines are added to the output for clarity, but are not required.

The third sample case corresponds to the image from the statement.