Task: GAM
Game

Day 1. Source file `gam.*` *19–04–2008*

Available memory: 8 MB. Maximum running time: 6 s.

Two players, *A* and *B*, play a game on a square board of size $n \times n$. The squares of the board are either white or black. The game is played only on the white squares — the black ones are excluded from the game. Each player has one piece, initially placed at this player's *starting point* — one of the white squares on the board. The starting point of *A* is different than that of *B*.

In each move a player moves his piece to one of the neighboring white squares (either up, down, left or right). If the player moves his piece to the square currently occupied by his opponent's piece, he gets an extra move (this way he may jump over the opponent). Note that in this case the direction of the second move can be different than that of the first move.

Player *A* moves first, then players alternate. The goal of the game is to reach the opponent's starting point. The player whose piece reaches his opponent's starting point first, wins the game. We want to determine which player has a winning strategy (a player has a winning strategy if he can win regardless of his opponent's moves).

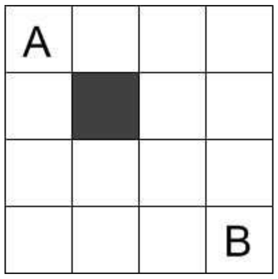


Figure 1. If *A* moves to the right on his first three moves, *B* will move up the first three moves. Thus, on the third move player *B* will reach the square with *A*'s piece and will be allowed to move again. Because of this, *B* will reach *A*'s starting point first and will win the game.

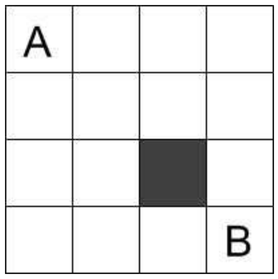


Figure 2. *A* can start by moving one step to the right and one step down. Then, depending on the first two moves of *B*, he will either go down or right and evade *B*. This way *A* will reach *B*'s starting point first, thus winning the game. In fact we proved that *A* has a winning strategy.

Task

Write a program, that:

- reads the layout of the grid and the starting points of the two players from the standard input,
- finds the player who has a winning strategy,
- writes the result to the standard output.

Input

The first line of the standard input contains one integer t the number of test cases ($1 \leq t \leq 10$). After it the description of t tests appears. Each test is described as follows. In the first line of the test there is one integer n ($2 \leq n \leq 300$), the length of the side of the grid. Then next n lines contain the description of the grid. Each line consists of n characters (with no white-spaces between them). Each character is either '`.`' (a white square), '`#`' (a black square), '`A`' (the starting point of A) or '`B`' (the starting point of B).

You may assume that there exists a path of white squares between the starting points of A and B.

Additionally, in test cases worth 60% of points, $n \leq 150$ and in test cases worth 40% of points, $n \leq 40$.

Output

For each test case exactly one line should be printed to standard output containing a single character '`A`' or '`B`', indicating the player who has a winning strategy.

Example

For the input data:

```
2
4
A...
.#..
....
...B
4
A...
....
..#.
...B
```

the correct result is:

```
B
A
```

Task: GAT
Gates

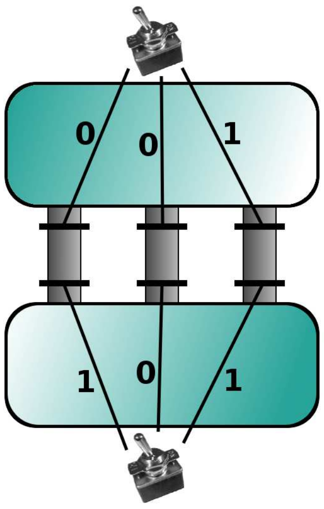
Day 1. Source file `gat.*` *19–04–2008*

Available memory: 96 MB. Maximum running time: 4 s.

After many years of working as a software developer you have decided to try something entirely different, and started looking at random job offers. The one that really caught your eye was a job in fish farming (a form of aquaculture). 'Cool!', you thought, and besides, fish are nice creatures. So you applied, got accepted, and today is your first day at work.

Your boss has already assigned you a task. You have to isolate one water reservoir from another. After looking at some schemes you've been given, here's what you've figured out.

The two water reservoirs are connected by several channels. Each channel has two gates. The channel is open when both gates are open, and is closed otherwise. The gates are controlled using switches. The same switch may operate several gates, but each gate is operated by exactly one switch. It is possible that both gates on a channel are controlled by the same switch and that a switch controls no gates.



Example with three channels and two switches.

The switch may operate the gate in one of two ways:

- the gate is open when the switch is on, and is closed when the switch is off,
- the gate is closed when the switch is on, and is open when the switch is off.

After playing a bit with the switches you suddenly realize that your programming experience will come in very handy. Write a program that, given the configuration of gates and switches, determines whether it is possible to close all channels, and if it is, then finds a state of every switch in one such valid configuration.

Input

The first line of the standard input contains two integers n ($1 \leq n \leq 250\,000$) and m ($1 \leq m \leq 500\,000$), the number of channels and switches respectively. Switches are numbered from 1 to m. Additionally, in test cases worth at least 30% points, n will not exceed 40 and m will not exceed 20.

The following n lines describe channels, each channel is described by a separate line containing four integers: a, s_a, b, s_b. Numbers a and b represent switches ($1 \le a, b \le m$) that operate gates of this channel. Numbers s_a and s_b can be either 0 or 1 and correspond to the described operation modes: $s_i = 0$ means that the gate is closed if and only if the switch i is off and $s_i = 1$ means that the gate is closed if and only if the switch i is on.

Output

If it is possible to close all the channels, the standard output should contain m lines. Line i should contain 0, if switch i should be off, and 1 if switch i should be on. If there are many possible solutions, your program may output any of them.

If it is impossible to close all channels, your program should output one line, containing a single word `IMPOSSIBLE`.

Example

For the input data:

```
3 2
1 0 2 1
1 0 2 0
1 1 2 1
```

the correct result is:

```
0
1
```

and for the input data:

```
2 1
1 0 1 0
1 1 1 1
```

the correct result is:

```
IMPOSSIBLE
```

The first example corresponds to the picture from the task description.

Task: MAG
Magical stones

Day 1. Source file `mag.*` *19–04–2008*
Available memory: 32 MB.

Famous stones Xi-*n*-*k* can only be found in Wonderland. Such a stone is simply a granite board with an inscription consisting only of letters `X` and `I`. Each board contains exactly *n* letters. There are not more than *k* positions in each board where letters `X` and `I` are next to each other.

The top and bottom sides of the stones are not fixed, so the stones can be rotated upside-down. For instance two figures below depict exactly the same stone:

`IXXIIXXX` `XXXIIXXI`

Figure 1: Two ways of looking at the same stone. This stone is of type Xi-8-3, but also Xi-8-4 (and also of any type Xi-8-*k* for $k \geq 3$).

No two magic stones in Wonderland are the same, i.e. no two stones contain the same inscription (remember that the upside-down rotation of a stone is allowed).

If it is possible to read the inscription of some stone in two different ways (using the upside-down rotation) then the *canonical representation* of the stone is defined as the lexicographically less* of these two ways of reading the inscription.

If a stone's inscription is symmetrical, i.e. the upside-down rotation does not change it, then its canonical representation is defined as the unique way of reading this inscription.

Example: There are exactly 6 stones of type Xi-3-2. Their canonical representations written in lexicographical order are: `III`, `IIX`, `IXI`, `IXX`, `XIX` and `XXX`.

Alice is a well-known expert on the Xi-*n*-*k* stones from Wonderland. She would like to create a lexicographical index of the canonical representations of all stones of type Xi-*n*-*k* (for some specific values of *n* and *k*). What inscription should be written at position *i* of the index, for a given value of *i*?

Task

Write a programme which:

- reads numbers *n*, *k* and *i* from the standard input,
- determines the i^{th} (in the lexicographical order) canonical representation of a Xi-*n*-*k* stone,
- writes the result to the standard output.

Input

The first and only line of the standard input contains three integers *n*, *k* and *i* ($0 \leq k < n \leq 60$, $0 < i < 10^{18}$) separated by single spaces.

*We say that inscription *A* is lexicographically less than *B* (assuming that lengths of *A* and *B* are the same) if *A* contains letter `I` and *B* contains letter `X` at the first position where the inscriptions differ.

Output

The first and only line of the standard output should contain the i^{th} (in the lexicographical order) canonical representation of a Xi-*n*-*k* stone.

If the number of Xi-*n*-*k* stones is less than *i* then the first and only line of output should contain expression `NO SUCH STONE`.

Example

For the input data:

```
3 2 5
```

the correct result is:

```
XIX
```

and for the input data:

```
3 2 7
```

the correct result is:

```
NO SUCH STONE
```