

## Problem A. Alakazam

Input file:            **standard input**  
Output file:         **standard output**  
Time limit:          2 seconds  
Memory limit:       512 megabytes

Hello, fellow Pokémon trainer! So you're on a safari as well? It's nice you're here, you were the one to spot an Alakazam colony!

The colony consists of  $n$  Alakazam specimens. Each of them holds a number of spoons increasing their psychic abilities. They've just formed up in a line in order to perform a psychic training. It's not just *any* kind of training — it's the teleportation training!

The training consists of a number of group teleportations. During each teleportation, a contiguous group of Alakazam specimens disappear and then materialize in the same positions as before, but in a totally random order.

Just before the training, you counted the spoons held by each Alakazam. However, after the training started, you were too far away to count the spoons — you only saw the teleportations. Basing on your observations, can you predict how many spoons are held by Alakazam at some positions in the line throughout the training? Obviously, as the process is random, we're only asking you about the expected number of spoons.

### Input

The first line contains two integers  $n, q$  ( $1 \leq n \leq 250\,000$ ,  $1 \leq q \leq 250\,000$ ) — the number of Alakazam in the line and the number of actions during the training. The following line contains  $n$  integers  $a_1, a_2, \dots, a_n$  ( $1 \leq a_i \leq 10^6$ ) — the number of spoons held by each specimen in the line.

Each of the next  $q$  lines describes a single action and is in one of the following formats:

- **shuffle**  $l\ r$  ( $1 \leq l \leq r \leq n$ ) — the group of Alakazam between the  $l$ -th and the  $r$ -th specimen (inclusive) teleport and materialize in random order.
- **get**  $i$  ( $1 \leq i \leq n$ ) — you need to predict the expected number of spoons held by the  $i$ -th Alakazam in the line.

There will be at least one **get** query in the file.

### Output

For each **get** query, output a single decimal number: the expected number of spoons held by the specified Alakazam.

Your answer will be considered correct if its absolute or relative error doesn't exceed  $10^{-9}$ .

### Example

standard input	standard output
3 6	1.000000000000
1 2 3	3.000000000000
get 1	1.500000000000
get 3	2.250000000000
shuffle 1 2	
shuffle 2 3	
get 1	
get 3	

## Note

Before the teleportations, Alakazam hold 1, 2 and 3 spoons:



After the first teleportation, two permutations of Alakazam are equally probable:  $(1, 2, 3)$  and  $(2, 1, 3)$ . Meanwhile, after both teleportations, the following permutations are equally probable:  $(1, 2, 3)$ ,  $(1, 3, 2)$ ,  $(2, 1, 3)$ ,  $(2, 3, 1)$ .

## Problem B. Bulbasaur

Input file:            `standard input`  
Output file:         `standard output`  
Time limit:          6 seconds  
Memory limit:       512 megabytes

Bulbasaur is standing next to the field full of holes in the ground. There are  $n \cdot k$  holes partitioned into  $n$  groups, each consisting of  $k$  holes. The groups are numbered with integers from 1 to  $n$  and the holes in each group are numbered from 1 to  $k$ . There are also some one-directional tunnels between the holes. They can only connect the holes in the  $i$ -th group with the holes in the  $(i + 1)$ -th group for any  $i$  from the range  $[1, n - 1]$ .

Bulbasaur wants to penetrate the holes and tunnels with its vines. Its vine can enter any hole, go through the tunnels and then finish in some other hole (without going out from the ground). Each taken tunnel must lead **from the lower-indexed group to the higher-indexed group**. As the holes and tunnels are rather tight, only one vine can be in any hole or any tunnel at the same time.

Let's denote by  $f(i, j)$  the maximum number of vines which Bulbasaur can simultaneously put in some of the holes in the  $i$ -th group and which can reach the holes in the  $j$ -th group.

Your task is to calculate

$$\sum_{i=1}^n \sum_{j=i+1}^n f(i, j).$$

### Input

The first line contains two integers  $n$  and  $k$  ( $2 \leq n \leq 40\,000$ ,  $1 \leq k \leq 9$ ) — the number of groups and the number of holes in each group.

Each of next  $n - 1$  blocks describes the connections between the neighboring groups.

Each block consists of  $k$  lines, and each line contains  $k$  characters. Blocks are separated with single empty lines.

If there is a tunnel between the  $p$ -th hole in the  $i$ -th group and the  $q$ -th hole in the  $(i + 1)$ -th group, then the  $q$ -th character in the  $p$ -th line of the  $i$ -th block is '1'. Otherwise, this character is '0'.

### Output

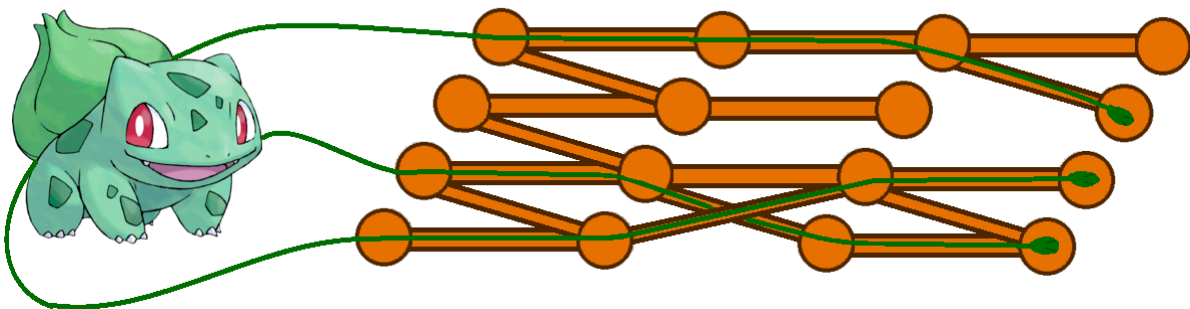
Output the value of the sum described in the task statement.

## Examples

standard input	standard output
<pre> 4 4 1000 1100 0110 0011  0100 1100 0010 0001  1000 1100 0000 0011 </pre>	<pre> 21 </pre>
<pre> 5 3 000 100 010  000 100 010  010 101 010  010 101 010 </pre>	<pre> 17 </pre>

## Note

Here's the illustration of the first sample test and a sample way to put 3 vines all the way from the first group to the last group:



The holes in each group are numbered bottom-up. It's impossible to use 4 vines, so  $f(1, 4)$  equals 3.

In the second sample test,  $f(3, 4) = f(4, 5) = f(3, 5) = 2$  as Bulbasaur cannot have more than one vine in any hole or tunnel.

## Problem C. Cloyster

Input file:            `standard input`  
Output file:         `standard output`  
Time limit:          2 seconds  
Memory limit:       512 megabytes

*This is an interactive problem.*

You just found yourself in a huge field of Cloyster specimens somewhere on the seabed off the coast of Japan. They're quite mad at you as you tried to catch one of them a moment ago. You must immediately find the leader of the pack and beg him for forgiveness. Where is he, though?

The field contains  $n^2$  square cells grouped into  $n$  rows and  $n$  columns. Each cell contains a single Cloyster hidden in its own shell. Now, I can reveal to you a couple of little-known facts about Cloyster:

- The leader has the largest shell. *Obviously.*
- All Cloyster in the pack have shells of different sizes.
- Each Cloyster other than the leader has a neighbor with a larger shell in some cell adjacent to its own cell by an edge or by a corner.

You can query individual specimens for the sizes of their shells. Use this information to locate the leader of the pack. Be quick, though — if you don't find him in  $3n + 210$  queries, Cloyster will run out of patience and they'll attack you!

### Interaction Protocol

The first line contains a single integer  $n$  ( $2 \leq n \leq 2000$ ) — the size of the field. Your solution should read it first. Afterwards, your program can make two types of requests:

- “?  $i$   $j$ ” ( $1 \leq i, j \leq n$ ) — query the size of the shell of the specimen in the cell in the  $i$ -th row and the  $j$ -th column. You can use this query at most  $3n + 210$  times. The response will be a single integer  $a_{ij}$  ( $1 \leq a_{ij} \leq 10^9$ ) printed on a separate line.
- “!  $i$   $j$ ” ( $1 \leq i, j \leq n$ ) — answer that the leader of the pack is in the  $i$ -th row and the  $j$ -th column. This should be your last query. Your solution should then terminate immediately.










The interactor isn't adaptive, i.e. the sizes of the shells are fixed before the run of your program and won't change between your queries. Remember to flush the output after printing each line!


























## Examples

standard input	standard output
3	? 1 1
1	? 2 3
4	? 3 2
8	? 3 3
9	? 2 2
5	! 3 3
5	? 4 4
2	! 1 1

## Note

Here are the sizes of each Cloyster in both sample tests:

 1	 2	 3
 7	 5	 4
 6	 8	 9

 97	 29	 31	 37	 41
 89	 23	 19	 17	 43
 83	 13	 11	 7	 47
 79	 5	 3	 2	 53
 73	 71	 67	 61	 59

Note, that the communication in the second sample test is correct — your program is allowed to make a guess even if it isn't certain about the correctness of the answer.

## Problem D. Dedenne

Input file: `standard input`  
Output file: `standard output`  
Time limit: 5 seconds  
Memory limit: 512 megabytes

Have you ever heard about the famous Dedenne antennas? They allow them to communicate with each other even when they're miles and miles apart. However, they suspect that they don't communicate in an optimal way. That's why Dedenne came to you in order to optimize their language.

The language used by Dedenne consists of  $n$  words. In order to transmit a word, Dedenne must first use their memorized dictionary to convert the word into a non-empty binary string (a codeword) and then transmit the resulting string. The recipient will then receive the sequence and use the same dictionary to recover the word. In order to preserve the correctness of the transmission, no two codewords in the dictionary should be equal or even no string should be a prefix of any other string.

Due to the peculiar nature of Dedenne, the codewords aren't allowed to have two consecutive 0 bits — when transmitted, they merge together into  $\infty$  and hang up the transmitter indefinitely.

The dictionary is obviously quite painful to memorize. Formally speaking, let  $C(s)$  be the *cost* of any binary string  $s$ , equal to exactly  $\sum_{j=1}^k \lfloor 1 + \log_2 j \rfloor$  if  $s$  is the prefix of exactly  $k$  codewords. Then, the cost to memorize the dictionary is equal to the sum of  $C(s)$  over all binary strings  $s$  which are the prefixes of at least one codeword.

For example, consider the dictionary consisting of 4 codewords: 0, 10, 110, 111. We can now see that the cost of memorizing the dictionary is equal to  $C(\varepsilon) + C(0) + C(1) + C(10) + C(11) + C(110) + C(111) = 8 + 1 + 5 + 1 + 3 + 1 + 1 = 20$ . Note that  $\varepsilon$  denotes an empty word.

Now you probably know what Dedenne want from you. What is the minimum possible cost of memorizing the dictionary?

### Input

The first line contains a single integer  $t$  ( $1 \leq t \leq 50\,000$ ) — the number of independent testcases. Each of the following  $t$  lines describes a single testcase and contains one integer  $n$  ( $2 \leq n \leq 10^{15}$ ) — the number of words that Dedenne need to memorize.

### Output

For each testcase, output a single integer — the minimum cost of a dictionary consisting of  $n$  words that can be achieved by Dedenne.

### Example

standard input	standard output
3	5
2	20
4	98
10	

### Note

Here is a dictionary which might be used by Dedenne for  $n = 10$  to minimize the cost of memorizing:





## Problem E. Eevee

Input file: standard input  
Output file: standard output  
Time limit: 5 seconds  
Memory limit: 512 megabytes

As you may know, Eevee can evolve in various ways. For instance, it will evolve when it comes close to one of the evolution stones.

Let's assume that there are  $n$  different evolution stones, numbered with integers from 1 to  $n$ . Each of them was split into  $k$  fragments. Eevee has grouped all the fragments into  $k$  stacks, numbered with integers from 1 to  $k$ . Each stack contains exactly  $n$  fragments, each of which comes from a different stone. Therefore, we can consider each stack as a permutation of the fragments of the different stones.

Eevee will perform the following operation  $k \cdot n$  times: pick one of the non-empty stacks, remove its topmost fragment and add it to the end of a sequence of fragments (which is initially empty). Two ways of combining the stacks into one sequence are considered distinct if at any step we choose a stack with a different index. If after this process there are  $k$  neighboring fragments of the same stone in the sequence, Eevee can accidentally evolve. As he is the cutest without any evolutions, we call some way to combine the stacks **good** if there is no interval of length  $k$  containing only the fragments of the same stone.

Let  $f(i, j)$  denote the number of good ways to combine the stacks with indices from the range  $[i, j]$ . Here, when we consider some interval of length  $\ell$ , we assume that Eevee performs the operation only  $\ell \cdot n$  times and that we prohibit any interval of length  $\ell$  from containing only the fragments of the same stone.

Your task is to calculate

$$\left( \sum_{i=1}^k \sum_{j=i+1}^k f(i, j) \right) \mod (10^9 + 7).$$

However, it turned out that Eevee **shuffled each stack** before the process! Therefore, each stack contains a random permutation of the fragments. Each of the input permutations is chosen equiprobably from all  $n!$  possible permutations and independently from each other.

### Input

The first line contains two integers  $k$  and  $n$  ( $2 \leq k, n \leq 300$ ) — the number of stacks and the number of fragments in each stack.

Each of the next  $k$  lines contains  $n$  integers. The  $j$ -th number in the  $i$ -th line is  $a_{i,j}$  ( $1 \leq a_{i,j} \leq n$ ,  $a_{i,j} \neq a_{i,l}$  for  $j \neq l$ ) and denotes the  $j$ -th number from the top in the  $i$ -th stack.

### Output

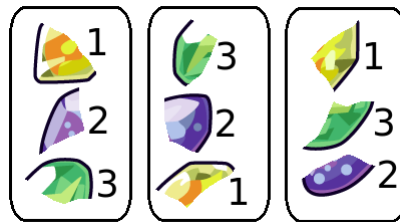
Output the value of the sum described in the problem statement.

### Examples

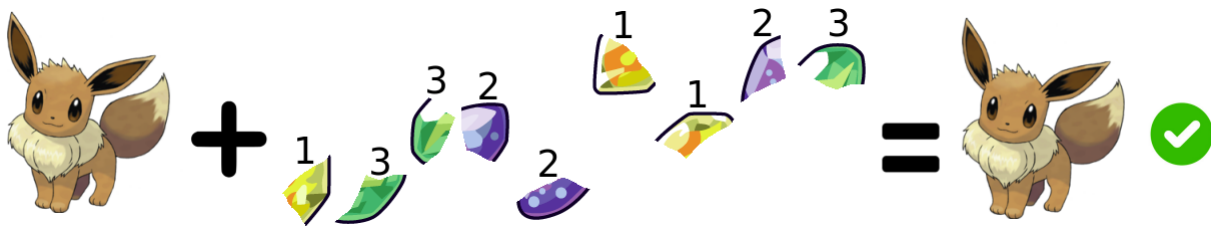
standard input	standard output
<pre>3 3 1 2 3 3 2 1 1 3 2</pre>	1464
<pre>4 2 1 2 2 1 1 2 2 1</pre>	2466

## Note

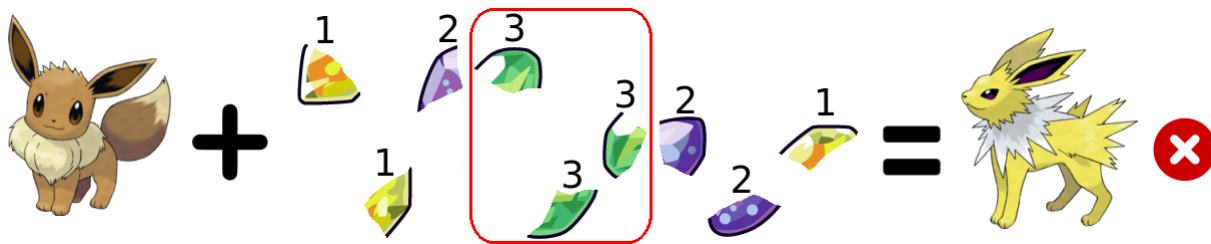
Consider calculating  $f(1, 3)$  in the first sample. Eevee has three stacks of the fragments of the stones:



One of the good ways to combine them is as follows:



The following one isn't good as it contains three 3s in a row:



In the first sample  $f(1, 2) = 8$ ,  $f(1, 3) = 1446$  and  $f(2, 3) = 10$ .

In the second sample  $f(1, 2) = 2$ ,  $f(1, 3) = 66$ ,  $f(1, 4) = 2328$ ,  $f(2, 3) = 2$ ,  $f(2, 4) = 66$  and  $f(3, 4) = 2$ .

## Problem F. Flaaffy

Input file: `standard input`  
Output file: `standard output`  
Time limit: 15 seconds  
Memory limit: 512 megabytes

So you want to see how Pokémon play games, eh? It's a good day for you — Flaaffy, a sheep-like electric Pokémon, just found an electronic Number Guessing Board™ and it wants to have fun with it!

The board is a five-digit electronic display that can show all integers from 0 to 99 999. When Flaaffy turned it on, all five digits were initially set to 0. On its startup, the board chose a secret integer  $x$  in the interval  $[L, R]$ . Flaaffy wants to guess this number. It can use electric shocks to operate the board in two following ways:

- Change a single digit on the display.
- Ask the board if  $x$  is smaller, equal or larger than the number shown on the display.

The game ends if Flaaffy can correctly determine what the hidden number is.

However, each operation depletes the amount of electricity stored by Flaaffy. Therefore, it wants to determine the hidden number in the minimum possible number of shocks. Flaaffy already figured out the optimal strategy, can you?

### Input

The first line contains a single integer  $t$  ( $1 \leq t \leq 50$ ) — the number of independent testcases in the file. Each of the following  $t$  lines describes a single testcase and contains two integers  $L, R$  ( $1 \leq L < R \leq 99\,999$ ).

### Output

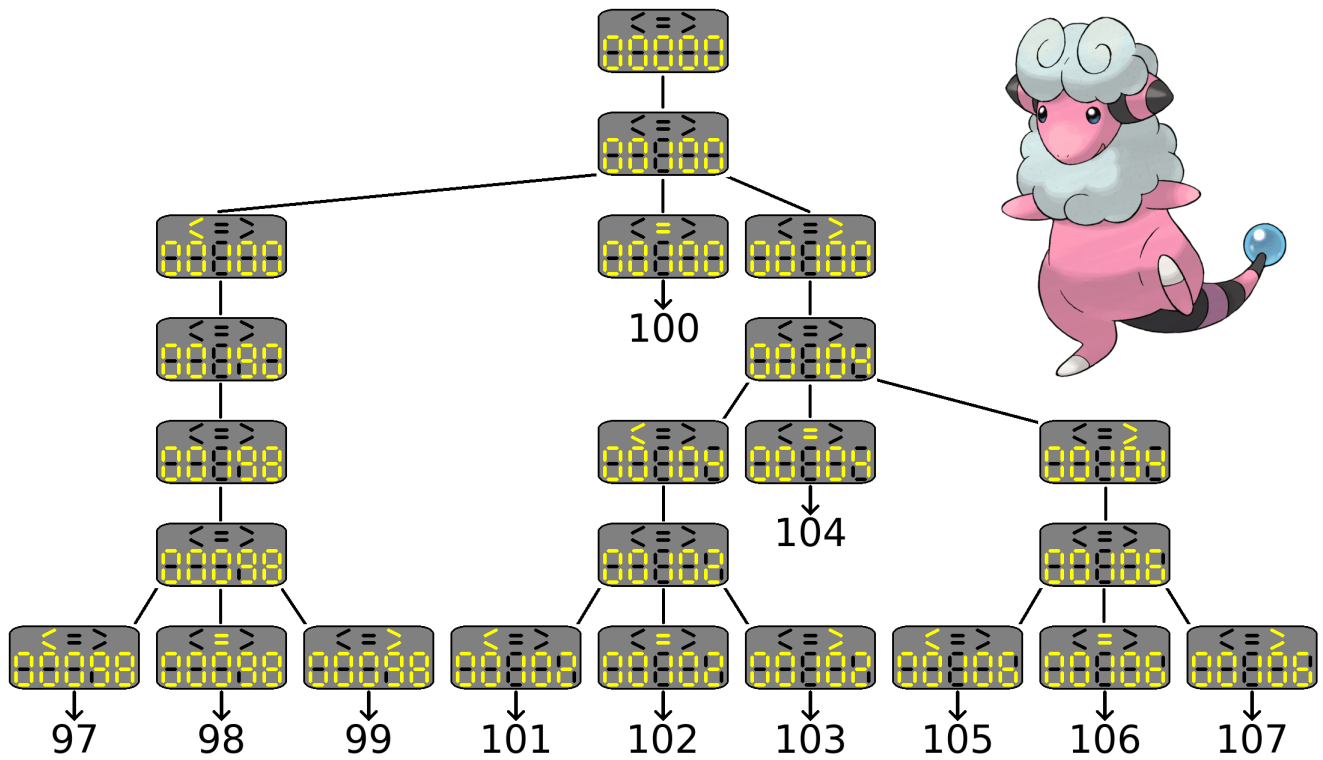
For each testcase, output a single number — the minimum number of shocks Flaaffy needs to produce in order to correctly guess the hidden number.

### Example

standard input	standard output
3	6
97 107	5
12043 12045	7
61 69	

### Note

Here is the decision tree for the first testcase:



Each edge means either changing one digit or comparing  $x$  with the number currently on the display. In the leaves of the tree Flaaffy is already sure what the hidden number is.

## Problem G. Gurdurr

Input file:            `standard input`  
Output file:          `standard output`  
Time limit:           5 seconds  
Memory limit:        512 megabytes

Gurdurr are often helping people on construction sites. Don't worry, Gurdurr are treated very well at work — they're given free food and are allowed frequent breaks from the work. During these breaks, they like to play their favorite game — Jenga. As they are really strong, they play Jenga with heavy steel blocks — the things that they never part with. Their Jenga has the following rules:

There is a tower consisting of  $n$  layers of blocks. Each layer consists of three long steel blocks. The blocks in each layer lie parallel to each other. The blocks in two neighboring layers are perpendicular to each other. Some blocks might be missing at the start of the game. Two players make their moves alternately. During one move a player must choose a block and remove it from the tower provided that the tower remains stable afterwards. The tower is stable if all the following conditions are met:

- Each layer contains at least one block.
- If a layer contains exactly one block, it's the middle one.
- There are no two neighboring layers such that both of them contains only one block.

A player who is unable to make any move loses. Gurdurr are quite good at this game, so you can assume that they play optimally. You are given a starting state of the tower, possibly with some blocks already removed. It's guaranteed that the initial tower is stable. Your task is to determine which player will win this game.

*Please note that in this version of Jenga, players don't put the blocks on the top of the tower.*

### Input

The first line contains one integer  $t$  ( $1 \leq t \leq 30\,000$ ) — the number of testcases. Each of the next  $t$  blocks describes one testcase.

Each block starts with a line with a single integer  $n$  ( $1 \leq n \leq 20$ ) — the number of layers in the tower. Then there are  $n$  lines, the  $i$ -th of them describes the  $i$ -th layer from the top of the tower.

Each of these lines contains a string which consists of exactly three characters. Each of them is either 'I' or '.'. If the  $j$ -th character in the  $i$ -th line is 'I', then the  $j$ -th block in the  $i$ -th layer from the top hasn't been removed. If this character is '.', then this block has been removed. We assume the natural numbering of the blocks in each layer, so the second block is the middle one and the first and the third blocks are on the exterior of a layer.

It's guaranteed that all the described towers are stable.

### Output

For each testcase output one line with one string which is either **First** or **Second**, depending on which player will win.

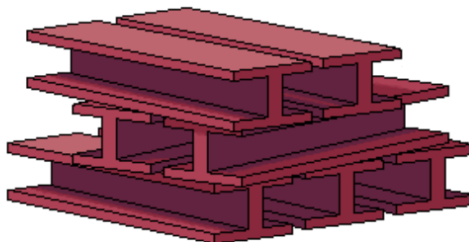
## Examples

standard input	standard output
5 1 III 1 I.I 1 .I. 1 .II 2 III III	First Second Second First First
1 3 II. .II III	Second

## Note

Note that in the second and third testcase from the first sample test it isn't possible to do any move, so the first player loses immediately.

Here is the tower from the second sample test together with two Gurdurr with their own metal beams (which are not taken from the tower):



## Problem H. Hypno

Input file: `standard input`  
Output file: `standard output`  
Time limit: 2 seconds  
Memory limit: 512 megabytes

You've just heard that somewhere in your city a new PokéStop has opened. You obviously want to go there and obtain the items available there. The city consists of  $n$  intersections connected by  $m$  bidirectional roads. The intersections are numbered with integers from 1 to  $n$ . You are currently at the first intersection and the PokéStop is at the  $n$ -th one. It takes one minute to cross each road. How fast can you reach the PokéStop?!

Uh, not so fast. On each road, there is a single Hypno, ready to use its psychic powers on you as soon as you reach the midpoint of the road. As soon as it happens, you fall into a very short sleep. Hypno then makes you sleepwalk to the random endpoint of the road you're on. Therefore, after one minute, you'll be at the opposite end of the road with  $\frac{1}{2}$  probability, or back at the same intersection otherwise.

Here's a thing, though: you're immune to each individual Hypno with  $\frac{1}{2}$  probability. If a road you're on contains a Hypno you're immune to, you'll always safely cross the road in one minute. If a road contains a Hypno you're susceptible to, you'll fall asleep on each crossing of that road. You don't initially know which roads contain which Hypno, but after making an attempt to cross some road you know at which endpoint you currently are. What is the minimum expected time in which you can reach the PokéStop?

### Input

The first line contains two integers  $n, m$  ( $2 \leq n \leq 200\,000$ ,  $1 \leq m \leq 200\,000$ ) — the number of intersections in your city and the number of bidirectional roads connecting them. Each of the following  $m$  lines contains two integers  $a, b$  ( $1 \leq a, b \leq n$ ,  $a \neq b$ ) — the indices of the intersections connected by the road.

No two intersections are connected by multiple roads. The city is connected, that is, you can reach each intersection from any other intersection, possibly using multiple roads.

### Output

Print a single real number — the minimum expected time you need to reach the PokéStop at intersection  $n$ . Your answer is considered correct if its absolute or relative error does not exceed  $10^{-9}$ .

### Examples

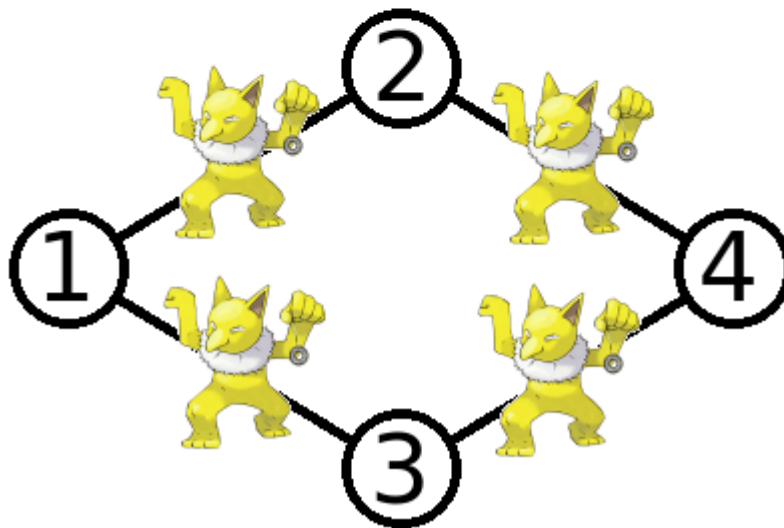
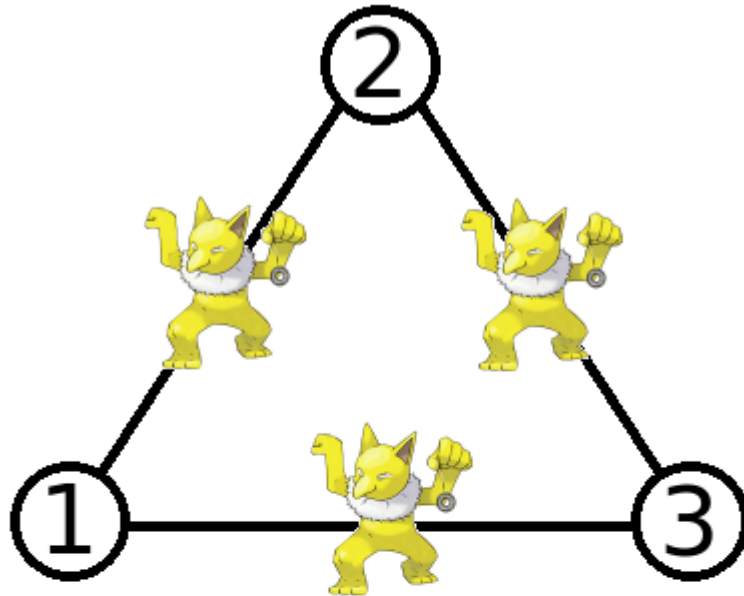
standard input	standard output
3 3 1 2 1 3 2 3	1.500000000000
4 4 1 2 2 4 4 3 3 1	2.875000000000

### Note

In the first sample test, the optimal strategy is to repeatedly try to go directly to the destination using the second road. With  $\frac{1}{2}$  probability, you are immune to Hypno which is standing on that road and you can reach the destination in time 1. On the other hand, you are susceptible with  $\frac{1}{2}$  probability, and we can show that in this case you'll reach the destination in expected time  $2 = \frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 3 + \frac{1}{16} \cdot 4 + \dots$ . Therefore, the overall expected time is  $\frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 2 = \frac{3}{2}$ .

In the second sample test, if you repeatedly tried to go through the second intersection, your expected time would be equal to 3 minutes. The same if you only tried to go through the third intersection. There exists a strategy to achieve a better expected time.

Here are the illustrations of both sample tests:





## Problem I. Infernape

Input file:            **standard input**  
Output file:          **standard output**  
Time limit:           **7 seconds**  
Memory limit:        **512 megabytes**

Monkeys live on the trees, right? Infernape probably too. There is a tree with  $n$  vertices (a tree is a connected undirected graph without cycles) and  $q$  independent queries. Vertices are numbered with integers from 1 to  $n$ .

In each query, there are  $k$  Infernape in the vertices of the tree ( $k$  may be different for different queries). The  $i$ -th of them sits in the vertex  $v_i$  and has power  $r_i$ . Infernape heats all vertices which are in the distance less than or equal to its power from  $v_i$ . The distance between two vertices is the number of edges on the shortest path between them. The powers are non-negative, so each Infernape always heats its own vertex. Your task is to answer how many vertices are heated by at least  $k - 1$  Infernape.

### Input

The first line contains one integer  $n$  ( $2 \leq n \leq 100\,000$ ) — the number of vertices in the tree.

The  $i$ -th of the next  $n - 1$  lines describes the  $i$ -th edge of the tree and contains two integers  $a_i$  and  $b_i$  ( $1 \leq a_i, b_i \leq n$ ) — the endpoints of this edge.

It's guaranteed that the edges describe a correct tree.

The next line contains one integer  $q$  ( $1 \leq q$ ) — the number of queries.

Each of the following  $q$  blocks describes one query.

Each block starts with a line with a single integer  $k$  ( $2 \leq k \leq 300\,000$ ) — the number of Infernape in the current query.

Nextly, each block contains  $k$  lines. The  $i$ -th of them contains two integers  $v_i$  and  $r_i$  ( $1 \leq v_i \leq n$ ,  $0 \leq r_i \leq n - 1$ ) — the index of the vertex at which the  $i$ -th Infernape sits and the power of this Infernape.

The sum of  $k$  over all queries in one test doesn't exceed 300 000.

### Output

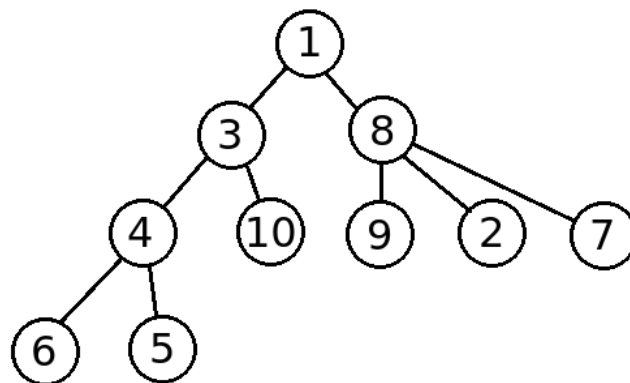
Output  $q$  lines. The  $i$ -th of them should contain the number of vertices heated by at least all but one Infernape in the  $i$ -th query.

## Example

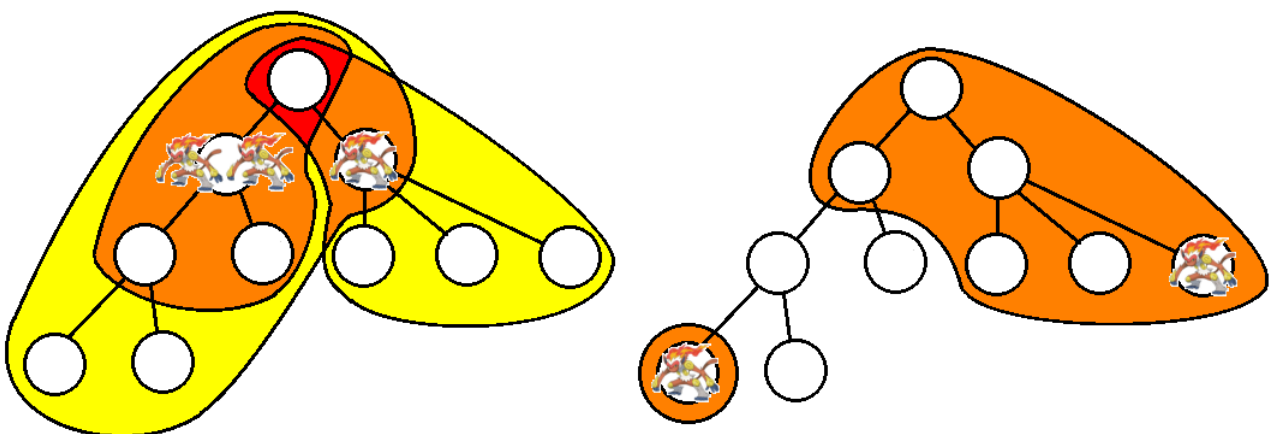
standard input	standard output
10	5
1 3	7
6 4	
9 8	
1 8	
3 4	
2 8	
10 3	
4 5	
8 7	
2	
3	
8 1	
3 1	
3 2	
2	
7 3	
6 0	

## Note

Here's how the tree in the sample test looks like:



And here is how the queries look like:



The red area is heated by all Infernape while the orange one is heated by all but one.

## Problem J. Jigglypuff

Input file: `standard input`  
Output file: `standard output`  
Time limit: 1 second  
Memory limit: 512 megabytes

Your ACM team has gathered in a picturesque city of Petrozavodsk for a team-building camp. You've heard that just outside of the city there is a Jigglypuff field that might help you make stronger bonds with your teammates, so you're definitely giving it a try.

The field is a rectangle partitioned into  $n \cdot m$  squares grouped into  $n$  rows and  $m$  columns. Each square contains a single Jigglypuff, which produces a note when a team member steps on its cell. Each note can be described by a single lowercase English character.

You and your two teammates will stand in the top-left corner of the field. Each of you will then go to the bottom-right corner, moving only right and down. Each of you has to pick a different route through the field.

Jigglypuff have psychic abilities. Therefore, if each of you hears exactly the same sequence of notes when passing through the field, your brains will perfectly synchronize and the team-building exercise will be complete. Is it possible to do so?

### Input

The first line contains two integers  $n, m$  ( $2 \leq n, m \leq 3000$ ) — the height and the width of the field. The following  $n$  lines describe the field. Each of these lines contains a single string of length  $m$  consisting of lowercase English characters. The first character in the first line is the top-left corner of the field.

### Output

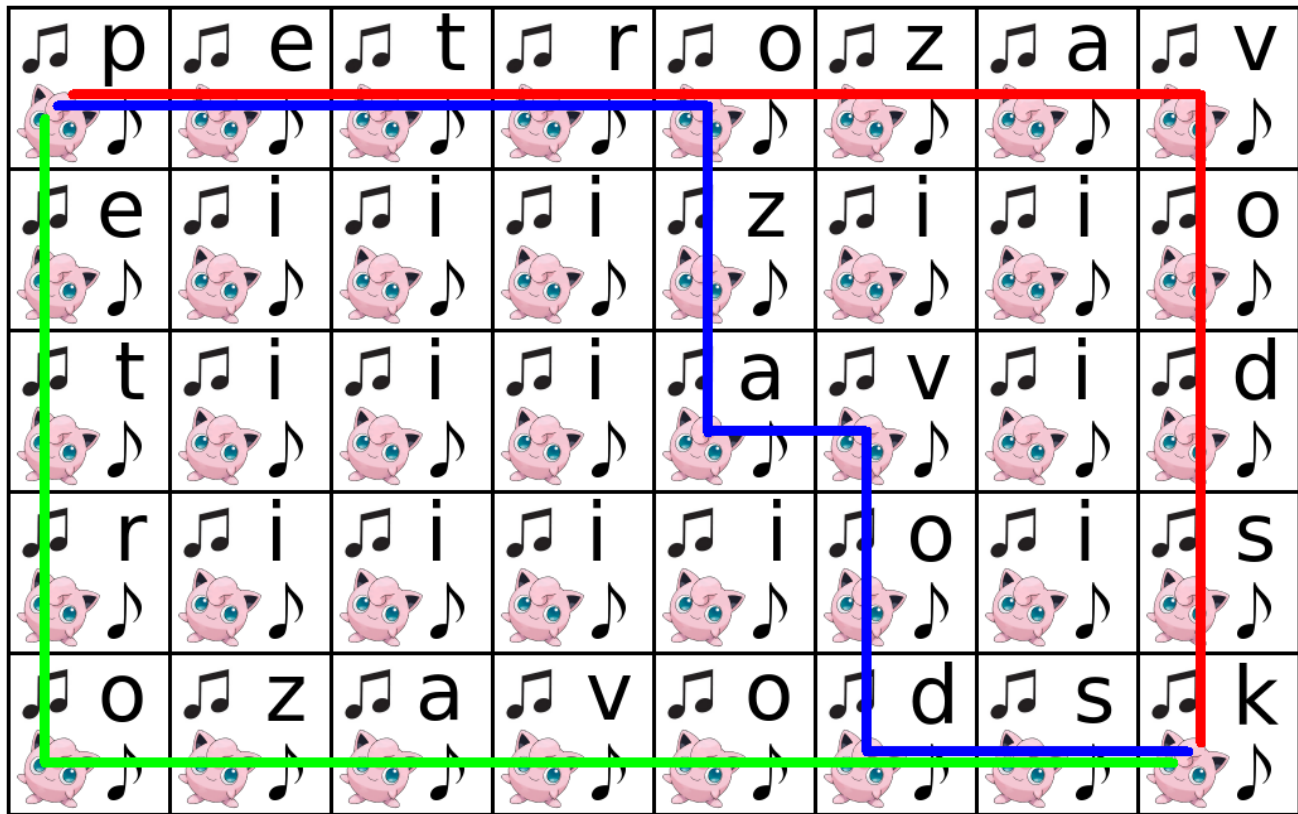
Output YES if it's possible to hear the same sequence of notes on at least three different routes through the grid, or NO otherwise. Each character can be printed in any case (either uppercase or lowercase).

### Examples

standard input	standard output
5 8 petrozav eiiiizio tiiiavid riiiiiois ozavodsk	YES
5 5 abcde fghij klmno pqrst uvwxy	NO

### Note

The following picture shows the first example test and three paths generating the same sequence of notes — petrozavodsk:



## Problem K. Kecleon

Input file: `standard input`  
Output file: `standard output`  
Time limit: 4 seconds  
Memory limit: 512 megabytes

Do you know how Kecleon looks? Similar to chameleons, its skin can be colored in various ways and this is what this task is about. For the purposes of this problem, let's assume that the body of Kecleon is colored in one out of 26 different colors, each denoted by a distinct lowercase English character.

Multiple Kecleon want to form a line, which is initially empty. You have to process two types of queries:

- In the first type, a Kecleon of a specified color comes to the right end of the line and remains there forever.
- In the second type, you're given an integer  $k$ . For each contiguous interval of  $k$  specimens in the line, consider the sequence of colors of Kecleon in this interval, from left to right. You have to say how many of these sequences are exactly the same as the sequence of colors of the first  $k$  Kecleon in the line.

Unfortunately, Kecleon still don't know the order in which they will join the line. Therefore, you need to process the queries online.

### Input

The input is encoded. In order to decode the queries, you need to maintain a variable `last` which is initially equal to 0. After each query of the second type, the value of `last` is set to the most recently computed answer.

The first line contains one integer  $q$  ( $2 \leq q \leq 300\,000$ ) — the number of queries in the input.

Each of the next  $q$  lines contains a description of one query. Each is in one of the following formats:

- **add**  $c'_i$  ( $c'_i$  is a lowercase English letter) — Let  $c_i = \text{char}((\text{asc}(c'_i) + \text{last}) \bmod 26)$  where  $\text{asc}(x)$  denotes the 0-based index of  $x$  in the English alphabet, and  $\text{char}(x)$  means the  $x$ -th character in the English alphabet (0-based as well). This operation adds a Kecleon of color  $c_i$  to the right end of the line.
- **get**  $k'_i$  ( $1 \leq k'_i \leq n$ ) — Let  $n$  be equal to the number of Kecleon currently in the line. Now, let  $k_i = ((k'_i - 1 + \text{last}) \bmod n) + 1$ . You need to find out how many intervals of length  $k_i$  have the same sequence of colors as the first  $k_i$  Kecleon in the line.

It's guaranteed that the first query has type **add** and that there is at least one query of type **get**.

### Output




For each query of type **get** output one line containing one integer described in the statement.

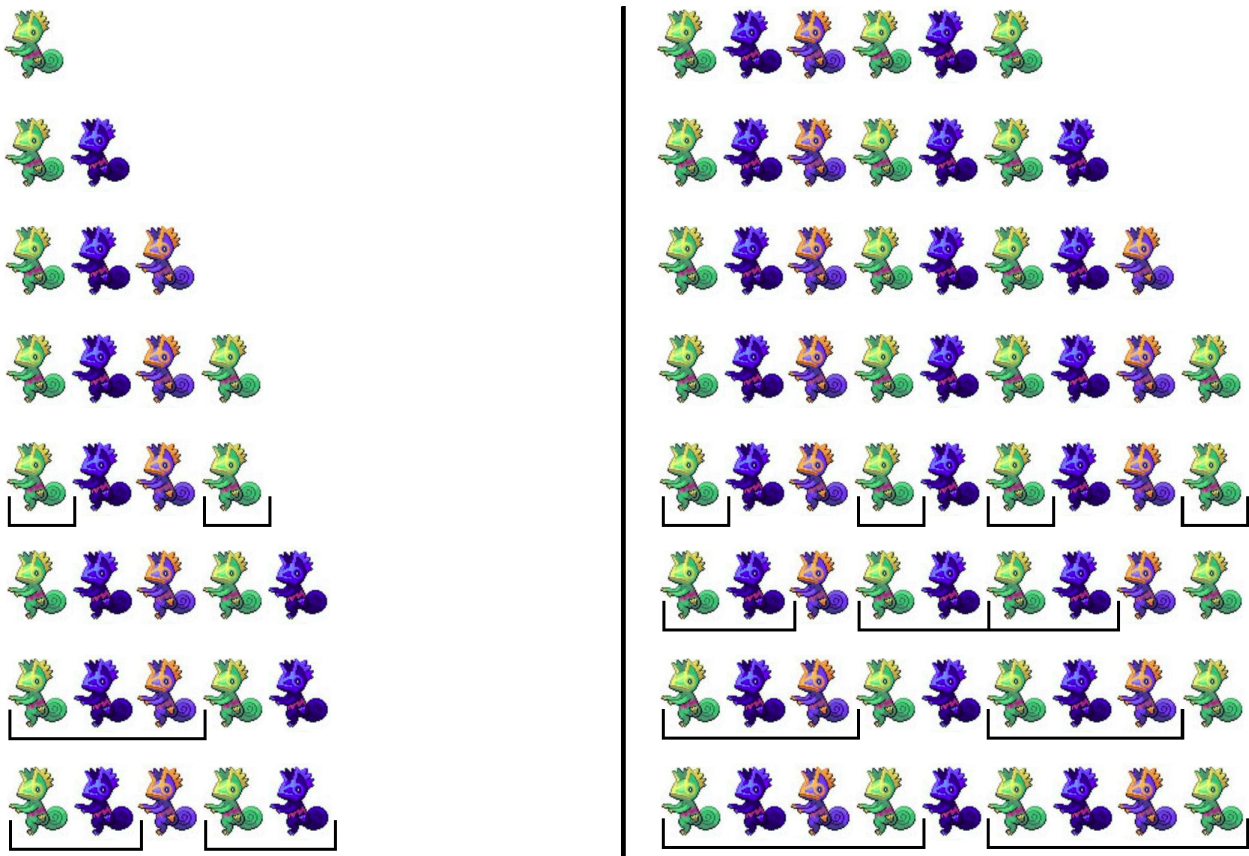
## Example

standard input	standard output
16	2
add a	1
add b	2
add c	4
add a	3
get 1	2
add z	2
get 1	
get 1	
add y	
add z	
add a	
add y	
get 8	
get 7	
get 9	
get 2	

## Note

In this sample, Kecleon come to the line in the following order: **abcababca**. The decoded values  $k_i$  are: 1, 3, 2, 1, 2, 3, and 4.

Assume that 'a' = , 'b' =  and 'c' = . Here is the exact order of queries:



## Problem L. Lati@s

Input file:            **standard input**  
Output file:          **standard output**  
Time limit:           **2 seconds**  
Memory limit:        **512 megabytes**

Latias and Latios are usually living together in peace, but recently they started arguing which of them is actually better. In order to resolve this issue, they agreed to play the following game.

The state of the game will contain a multiset of tuples. Each of them will contain exactly  $n$  non-negative integers. In one move a player must choose any of these tuples, as long as it doesn't contain any zero. Let's call this tuple  $A$ . The player now performs the following move:

Firstly, choose some other tuple  $B$  (the multiset doesn't have to necessarily contain any copy of  $B$ ), such that  $B$  also contains  $n$  non-negative integers and each element of  $B$  is strictly smaller than the **corresponding** element of  $A$ ; that is,  $B_i < A_i$  for each  $i = 1, 2, \dots, n$ . Next, a single copy of  $A$  is removed from the multiset. Then, for each **non-empty** subset  $X$  of integers from 1 to  $n$ , we add  $C_X$  to the multiset.  $C_X$  is a tuple such that  $(C_X)_i = B_i$  if  $i \in X$ , or  $(C_X)_i = A_i$  otherwise. For example, if  $A = (3, 7)$  and  $B = (0, 2)$ , then the tuples  $(0, 7)$ ,  $(3, 2)$  and  $(0, 2)$  will be added to the multiset. Notice that  $2^n - 1$  distinct tuples are always added in this step.

The player which is unable to make a move loses.

It wasn't easy for Latias and Latios to decide what multiset should be the starting one. As they happened to have an  $n \times n$  matrix  $M$  consisting of integers, they agreed to create a multiset containing  $n!$  tuples. For each permutation  $\sigma$  of the integers from 1 to  $n$ , the tuple  $(M_{1,\sigma(1)}, M_{2,\sigma(2)}, \dots, M_{n,\sigma(n)})$  is added to the multiset.

Latias goes first and then the players keep moving alternately. We can prove that the described game is finite, so it's always possible to determine the winner. Your task is to decide who will win assuming that both players play optimally.

### Input

The first line contains one integer  $n$  ( $1 \leq n \leq 150$ ).

Then come  $n$  lines, each consisting of  $n$  integers. The  $j$ -th integer in the  $i$ -th of these lines equals  $M_{i,j}$  ( $0 \leq M_{i,j} < 2^{64}$ ).

Please note that the numbers may not fit in the standard 64-bit **signed** type.

### Output

Output **Latias** or **First** if the first player (Latias) will win the game. Otherwise output **Latios** or **Second**. If you decide the winner correctly, both possible words will be accepted.

### Examples

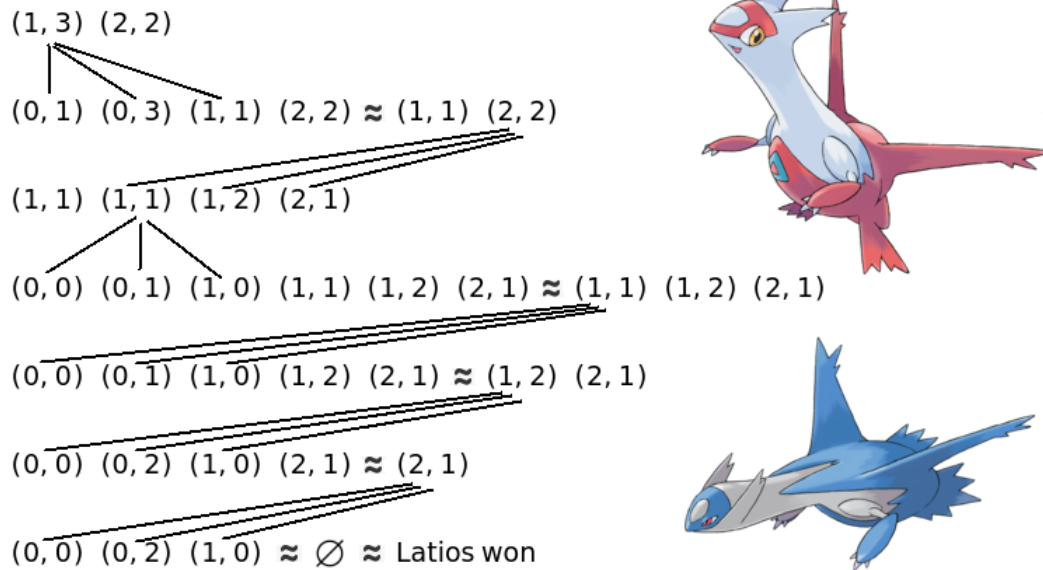
standard input	standard output
3 0 1 2 1 2 3 1 2 1	First
2 1 2 2 3	Second

### Note

**Latias** is also a correct answer for the first sample test. Similarly, **Latios** is a correct answer for the

second sample test. Sorry, but Lati@s is never considered correct.

Here is how the game in the second sample test may look:



As it's not possible to do any move from a tuple containing any zero, such tuples are omitted. The strategy above is optimal for Latios, i.e. it never gives Latias a chance to win.