Problem A. Robots on a grid

Input file:	`robots.in`
Output file:	`robots.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

You have recently made a grid traversing robot that can find its way from the top left corner of a grid to the bottom right corner. However, you had forgotten all your AI programming skills, so you only programmed your robot to go rightwards and downwards (that's after all where the goal is). You have placed your robot on a grid with some obstacles, and you sit and observe. However, after a while you get tired of observing it getting stuck, and ask yourself "How many paths are there from the start position to the goal position?", and "If there are none, could the robot have made it to the goal if it could walk upwards and leftwards?"

So you decide to write a program that, given a grid of size $n \times n$ with some obstacles marked on it where the robot cannot walk, counts the different ways the robot could go from the top left corner s to the bottom right t, and if none, tests if it were possible if it could walk up and left as well. However, your program does not handle very large numbers, so the answer should be given modulo $2^{31} - 1$.

Input

On the first line is one integer, $1 \leq n \leq 1000$. Then follows n lines, each with n characters, where each character is one of "`.`" and "`#`", where "`.`" is to be interpreted as a walkable tile and "`#`" as a non-walkable tile. There will never be a wall at s, and there will never be a wall at t.

Output

Output one line with the number of different paths starting in s and ending in t (modulo $2^{31} - 1$) or "`THE GAME IS A LIE`" if you cannot go from s to t going only rightwards and downwards but you can if you are allowed to go left and up as well, or "`INCONCEIVABLE`" if there simply is no path from s to t.

Example

robots.in	robots.out

5
`.....`
`#..#.`
`#..#.`
`...#.`
`.....`
	6

7
`......#`
`####...`
`.#.....`
`.#...#.`
`.#.....`
`.#..###`
`.#.....`
	THE GAME IS A LIE

Problem B. Mega Inversions

Input file:	`megainversions.in`
Output file:	`megainversions.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

The n^2 upper bound for any sorting algorithm is easy to obtain: just take two elements that are misplaced with respect to each other and swap them. Conrad conceived an algorithm that proceeds by taking not two, but three misplaced elements. That is, take three elements $a_i > a_j > a_k$ with $i < j < k$ and place them in order a_k, a_j, a_i. Now if for the original algorithm the steps are bounded by the maximum number of inversions $\frac{n(n-1)}{2}$, Conrad is at his wits' end as to the upper bound for such triples in a given sequence. He asks you to write a program that counts the number of such triples.

Input

The first line of the input is the length of the sequence, $1 \leq n \leq 10^5$. The next line contains the integer sequence a_1, a_2, …, a_n. You can assume that all $a_i \in [1, n]$.

Output

Output the number of inverted triples.

Example

megainversions.in	megainversions.out
3 1 2 3	0
4 3 3 2 1	2

Problem C. Death Knight Hero

Input file:	`deathknight.in`
Output file:	`deathknight.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

There once was a champion of *WoW*
Arthasdk the name he was bestowed
He *Death Gripped* you to his side
His *Chains of Ice* stopped your stride
And *Obliterates* made you say "OWW!"

But one day our hero got puzzled
His *Death Grip* totally fizzled
In his darkest despair
He could barely hear
"OMG NOOB u *Chains of Iced* than u *Death Gripped*"

Input

You are given a recording of the abilities our hero used in his battles. The first line of input will contain a single integer n ($1 \leq n \leq 100$), the number of battles our hero played. Then follow n lines each with a sequence of k_i ($1 \leq k_i \leq 1000$) characters, each of which are either "`C`", "`D`" or "`O`". These denote the sequence of abilities used by our hero in the i-th battle. "`C`" is *Chains of Ice*, "`D`" is *Death Grip* and "`O`" is *Obliterate*.

Output

Output the number of battles our hero won, assuming he won each battle where he did not *Chains of Ice* immediately followed by *Death Grip*.

Example

deathknight.in	deathknight.out
3 DCOOO DODOCD COD	2

Problem D. Elevator

Input file:	`elevator.in`
Output file:	`elevator.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

You are on your way to your first job interview as a program tester, and you are already late. The interview is in a skyscraper and you are currently in floor s, where you see an elevator. Upon entering the elevator, you learn that it has only two buttons, marked "UP u" and "DOWN d". You conclude that the UP-button takes the elevator u floors up (if there aren't enough floors, pressing the UP-botton does nothing, or at least so you assume), whereas the DOWN-button takes you d stories down (or none if there aren't enough). Knowing that the interview is at floor g, and that there are only f floors in the building, you quickly decide to write a program that gives you the amount of button pushes you need to perform. If you simply cannot reach the correct floor, your program halts with the message "use the stairs".

Given input f, s, g, u and d (floors, start, goal, up, down), find the shortest sequence of button presses you must press in order to get from s to g, given a building of f floors, or output "`use the stairs`" if you cannot get from s to g by the given elevator.

Input

The input will consist of one line, namely f s g u d, where $1 \le s, g \le f \le 10^6$ and $0 \le u, d \le 10^6$. The floors are one-indexed, i.e. if there are 10 stories, s and g be in $[1, 10]$.

Output

You must reply with the minimum numbers of pushes you must make in order to get from s to g, or output "`use the stairs`" if it is impossible given the configuration of the elevator.

Example

elevator.in	elevator.out
`10 1 10 2 1`	`6`
`100 2 1 1 0`	`use the stairs`

Problem E. ls

Input file:	`ls.in`
Output file:	`ls.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

You are implementing an operating system, and now need to write a program to list files in a directory: "ls". You want the user to be able to list only files that match a given pattern that can include wildcards (`*`), for example `*.c`. A wildcard matches zero or more characters of any kind.

Input

The first line contains a string P, containing 1–100 characters "`a`"–"`z`", "`*`" and "`.`". This is the pattern. The second line contains an integer N, $1 \le N \le 100$, which is the number of files in the directory. Then follows N lines containing the names of the files in the directory. Each line is a string containing 1–100 characters "`a`"–"`z`" and "`.`".

Output

The output shall consist of the filenames that match the pattern, P, each on its own line, in the same order that they were given as input.

Example

ls.in	ls.out

`*.*`
`4`
`main.c`
`a.out`
`readme`
`yacc`
	`main.c`
	`a.out`

`*a*a*a`
`4`
`aaa`
`aaaaa`
`aaaaax`
`abababa`
	`aaa`
	`aaaaa`
	`abababa`

Problem F. Knigs of the Forest

Input file:	`moose.in`
Output file:	`moose.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

All moose are knigs of the forest, but your latest moose-friend, Karl-Älgtav, is more interesting than most. In part because of his fondness of fermented blueberries, and in part because of the tribe he lives in. Each year his tribe holds a tournament to determine that year's alpha-moose. The winner gets to mate with all the moose-chicks, and then permanently leaves the tribe. The pool of contenders stays constant over the years, apart from the old alpha-moose being replaced by a newcomer in each tournament.

Karl-Älgtav has recently begun to wonder when it will be his turn to win all the chicks, and has asked you to help him determine this. He has supplied a list of the strength of each of the other male moose in his tribe that will compete during the next $n - 1$ years, along with their time of entry into the tournament. Assuming that the winner each year is the moose with greatest strength, determine when Karl-Älgtav becomes the alpha-moose.

Input

The first line of input contains two space separated integers k $(1 \le k \le 10^5)$ and n $(1 \le n \le 10^5)$, denoting the size of the tournament pool and the number of years for which you have been supplied sufficient information.

Next is a single line describing Karl-Älgtav, containing the two integers y $(2011 \le y \le 2011 + n - 1)$ and p $(0 \le p \le 2^{31} - 1)$. These are his year of entry into the tournament and his strength, respectively.

Then follow $n + k - 2$ lines describing each of the other moose, in the same format as for Karl-Älgtav.

Note that exactly k of the moose will have 2011 as their year of entry, and that the remaining $n - 1$ moose will have unique years of entry. You may assume that the strength of each moose is unique.

Output

The year Karl-Älgtav wins the tournament, or "`unknown`" if the given data is insufficient for determining this.

Example

moose.in	moose.out

2 4
2013 2
2011 1
2011 3
2014 4
2012 6
	2013

2 4
2011 1
2013 2
2012 4
2011 5
2014 3
	unknown

Problem G. Car Trouble

Input file:	`cartrouble.in`
Output file:	`cartrouble.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

The city center of an unnamed Nordic university town consists of what was once a medieval city with narrow winding streets completely surrounded by a high wall protecting the city against Swedish invaders and other unwanted elements. The wall has since been removed and replaced by a system of interconnecting roads completely circumscribing the old part of the town. The roads inside still remains more or less the same as it was in the middle ages, which of course comes in conflict with modern requirements for accessibility by car, resulting in a maze of twisty little one-way streets, all alike, mixed with slightly wider two-way streets.

Making changes to the traffic routes in such a city can easily cause unexpected side-effects if you do not plan carefully ahead. The story goes that a prominent member of the city council once submitted a proposal to the council regarding extensive changes to how the traffic should be organized in the city center. The proposal did have the merit that it would be very easy to drive in to the central square, but it would unfortunately also be impossible to drive out again. The council member in question later went on to become minister of justice in the country under the parole that society should be harder on criminals – "it should be easy to go to jail, but difficult to get out again".

To avoid mistakes as the one above, the city planners need you to develop a tool that can help them discover any traffic problems in the planning stage. The planners need to be alerted of two different situations. The first situation is that a street exists in the city center from which you cannot reach the surrounding, circular, system of roads, i.e., you are trapped inside the city. The second situation is that a street exists in the city that cannot be reached from the surrounding system of roads, i.e., it is unreachable.

Input

The input consists of a description of how streets connect to each other and the surrounding circular road system. Each street (or a segment of a street) within the city center is represented by an arbitrary integer id number ($0 < id < 1000$). The surrounding circular road system is represented by the special id number 0.

First line: An integer giving the number of streets (including the surrounding road system, $0 < streets \leq 1000$).

The following lines: One line for each street (no particular order required and the surrounding road system is included) consisting of a number of integers. First, an integer giving the id number of the street. Second, the number of (other) streets that can be reached from this street. Third, a sequence of street id numbers indicating which streets can be reached from this street.

Output

One line for each street on which you would be trapped within the city consisting of the text "`TRAPPED` X" where X is replaced by street id number in question.

Then, one line for each street within the city that is unreachable from the surrounding system of roads consisting of the text "`UNREACHABLE` X" where X should be replaced by the street id in question.

If no problems are found, i.e., you are not trapped in any street and every street is reachable, you should print a single line containing the text "`NO PROBLEMS`".

If multiple streets cause you to get trapped – or are unreachable – you should list them in the same order they were entered in the input (within respective category).

Example

cartrouble.in	cartrouble.out

6
0 1 1
1 1 2
2 3 1 3 0
3 0
4 2 5 0
5 1 4
	TRAPPED 3
	UNREACHABLE 4
	UNREACHABLE 5

2
1 1 0
0 1 1
	NO PROBLEMS

Problem H. Private Space

Input file:	`space.in`
Output file:	`space.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

People are going to the movies in groups (or alone), but normally only care to socialize within that group. Being Scandinavian, each group of people would like to sit at least one space apart from any other group of people to ensure their privacy, unless of course they sit at the end of a row. The number of seats per row in the cinema starts at X and decreases with one seat per row (down to a number of 1 seat per row). The number of groups of varying sizes is given as a vector $(N_1, \ldots, N_n)$, where N_1 is the number of people going alone, N_2 is the number of people going as a pair etc. Calculate the seat-width, X, of the widest row, which will create a solution that seats all (groups of) visitors using as few rows of seats as possible. The cinema also has a limited capacity, so the widest row may not exceed 12 seats.

Input

The first line of input contains a single integer n ($1 \leq n \leq 12$), giving the size of the largest group in the test case.

Then follows a line with n integers, the i-th integer (1-indexed) denoting the number of groups of i persons who need to be seated.

Output

A single number: the size of the smallest widest row that will accommodate all the guests. If this number is greater than 12, output "`impossible`" instead.

Example

space.in	space.out
3 0 1 1	3
3 2 1 1	4

Problem I. Prime Time

Input file:	primetime.in
Output file:	primetime.out
Time limit:	2 seconds
Memory limit:	256 megabytes

Odd and Even have have had their share of fun times playing the good old prime game. They start with an arbitrary natural number, and take turns either adding 1 or dividing by a prime (assuming the result is still a natural number), and the one to reach 1 is the winner.

However, now that they have a new friend, Ingmariay, they have decided to expand the rules of the game to allow for three-player action: Instead of determining a winner for each round of play, they instead score points; the lowest number each of them has claimed during the round is the amount of points they get. (If any of them did not have the opportunity to claim any numbers, the starting number will be their score for that round.) At the end of the day, the player with the fewest points wins. And to avoid bad blood between themselves, they have all agreed that each of them only will focus on minimizing their own scores, and that whenever a player can choose different numbers that will result in the same score, that player will choose the lowest of those numbers. They have also agreed on a fixed order of play: Odd → Even → Ingmariay → . . . , but they alternate who gets to start.

You recently missed one of their exciting evenings of play, because you had to make problems for the NCPC event. Fortunately for you, they had recorded the numbers and starting players for each round, and told you that since they always play optimally, you could use this to simulate the event for yourself. Oh joy!

As an example round, assume that Even is chosen as the starting player, and with the starting number 15. Then Even claims 16, Ingmariay 8, Odd 4, Even 2 and Ingmariay 1. Odd gets 4 points, Even 2 and Ingmariay 1.

Input

The first line of input contains a single integer n ($1 \le n \le 1000$), the number of rounds they played that evening.

Then follow n lines each beginning with the first character of the name of the starting player (either "O", "E" or "I"), followed by a space and then the starting number for that round, in the range $[1, 10000]$.

Note that, if the starting number is 1, all players receive 0 points for that round.

Output

Output a single line with the score at the end of the day for each of the three contestants, in the order "Odd", "Even", "Ingmariay".

Example

primetime.in	primetime.out
1 O 4	2 1 4
3 O 13 I 14 E 15	6 29 16

Problem J. Enemy Division

Input file:	enemydivision.in
Output file:	enemydivision.out
Time limit:	2 seconds
Memory limit:	256 megabytes

Captain Keram has to make a difficult decision. It is year 2147 and there is a big war in the world. His soldiers have been together since the war started, two years ago, and some of them have become enemies. Luckily, each soldier has at most three enemies.

They need to attack another country soon, and Keram is worried that soldiers who are enemies might not cooperate well during the battle. He has decided to divide them into groups such that every soldier has at most one enemy in his group. He also wants to make it simple, so he wants to use as few groups as possible. Can you divide the soldiers into groups for him?

Input

On the first line there are two integers n and m, $2 \le n \le 100\,000$, $0 \le m \le 3n/2$, where n is the number of soldiers and m is the number of enemy pairs. Then follow m lines, each containing two space separated integers a_i, b_i, denoting that soldiers a_i and b_i are enemies, where $1 \le a_i < b_i \le n$. You can assume that all soldiers have at most three enemies.

Output

The first line of output contains the minimal number of groups of soldiers k. Each of the next k lines contains a space separated list of a soldiers in a unique group.

Example

enemydivision.in	enemydivision.out

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