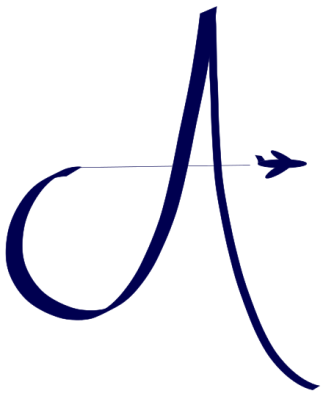
Problem A. Amanda Lounges

Source file name:	amanda.c, amanda.cpp, amanda.java
Input:	`Standard`
Output:	`Standard`

Amanda Air has routes between many different airports, and has asked their most important frequent flyers, members of the AA Frequent Flyer program, which routes they most often fly. Based on this survey, Amanda, the CEO and owner, has concluded that Amanda Air will place lounges at some of the airports at which they operate.
However, since there are so many routes going between a wide variety of airports, she has hired you to determine how many lounges she needs to build, if at all possible, given the constraints set by her. This calculation is to be provided by you, before any lounges are built.
Her requirements specifies that for some routes, there must be lounges at both airports, for other routes, there must be lounges at exactly one of the airports, and for some routes, there will be no lounges at the airports.
She is very economically minded and is demanding the absolute minimum number of lounges to be built.



H. A. Hansen, cc-by-sa

Input

The first line contains two non-negative integers $1 \leq n, m \leq 200\,000$, giving the number of airports and routes in the Amanda Catalog respectively. Thereafter follow m lines, each describing a route by three non-negative integers $1 \leq a, b \leq n$ and $c \in \{0, 1, 2\}$, where a and b are the airports the route connects and c is the number of lounges.

No route connects any airport with itself, and for any two airports at most one requirement for that route is given. As one would expect, 0 is a request for no lounge, 1 for a lounge at exactly one of the two airports and 2 for lounges at both airports.

Output

If it is possible to satisfy the requirements, give the minimum number of lounges necessary to do so. If it is not possible, output `impossible`.

Example

Input	Output
4 4 1 2 2 2 3 1 3 4 1 4 1 2	3
5 5 1 2 1 2 3 1 2 4 1 2 5 1 4 5 1	impossible
4 5 1 2 1 2 3 0 2 4 1 3 1 1 3 4 1	2

Problem B. What does the fox say?

Source file name:	basin.c, basin.cpp, basin.java
Input:	`standard`
Output:	`standard`

Determined to discover the ancient mystery - the sound that the fox makes - you went into the forest, armed with a very good digital audio recorder. The forest is, however, full of animals- voices, and on your recording, many different sounds can be heard. But you are well prepared for your task: you know exactly all the sounds which other animals make. Therefore the rest of the recording - all the unidentified noises - must have been made by the fox.

Input

The first line of input contains the number of test cases T. The descriptions of the test cases follow:

The first line of each test case contains the recording - words over lower case English alphabet, separated by spaces. Each contains at most 100 letters and there are no more than 100 words. The next few lines are your pre-gathered information about other animals, in the format <**animal**> **goes** <**sound**>. There are no more than 100 animals, their names are not longer than 100 letters each and are actual names of animals in English. There is no **fox goes ...** among these lines.

The last line of the test case is exactly the question you are supposed to answer: *what does the fox say?*

Output

For each test case, output one line containing the sounds made by the fox, in the order from the recording. You may assume that the fox was not silent (contrary to popular belief, foxes do not communicate by Morse code).

Example

Input
1 toot woof wa ow ow ow pa blub blub pa toot pa blub pa pa ow pow toot dog goes woof fish goes blub elephant goes toot seal goes ow what does the fox say?
Output
wa pa pa pa pa pa pow

Problem C. Magical GCD

Source file name:	magical.c, magical.cpp, magical.java
Input:	`standard`
Output:	`standard`

The Magical GCD of a nonempty sequence of positive integers is defined as the product of its length and the greatest common divisor of all its elements. Given a sequence $(a_1, \ldots, a_n)$, find the largest possible Magical GCD of its connected subsequence.

Input

The first line of input contains the number of test cases T. The descriptions of the test cases follow: The description of each test case starts with a line containing a single integer $n, 1 \leq n \leq 100000$. The next line contains the sequence $a_1, a_2, \ldots, a_n$, $1 \leq a_i \leq 10^{12}$.

Output

For each test case output one line containing a single integer: the largest Magical GCD of a connected subsequence of the input sequence.

Example

Input	Output
1 5 30 60 20 20 20	80

Problem D. Subway

Source file name:	subway.c, subway.cpp, subway.java
Input:	`standard`
Output:	`standard`

Johny is going to visit his friend Michelle. His dad allowed him to go there on his own by subway. Johny loves traveling by subway and would gladly use this opportunity to spend half a day underground, but his dad obliged him to make as few line changes as possible. There are a lot of stations in the city, and several subway lines connecting them. All trains are perfectly synchronized - the travel between two consecutive stations on every line takes exactly one minute, and changing lines at any station takes no time at all. Given the subway map, help Johny to plan his trip so that he can travel for as long as possible, while still following his dad's order.

Input

First line of input contains the number of test cases T . The descriptions of the test cases follow:

The description of each test case starts with an empty line. The next two lines begin with the strings **Stops:** and **Lines:**, and contain the names (separated by a comma and a space) of all subway stops and lines, respectively. A single line for each subway line follows (in no particular order), beginning with <**line-name**> **route:** and listing the names of the stops along this particular line. The final two lines specify the names of the (different) stations nearby Johny's and Michelle's homes.

In each test case, there are at most 300 000 stations and 100 000 lines, whose total length does not exceed 1 000 000. The names of lines and stations are between 1 and 50 characters long and can contain letters, digits, hyphens (**-**), apostrophes (**'**) and "and" signs (**&**). All lines are bidirectional (although changing the direction of travel counts as a line change) and there are no self-crossings.

Output

Print the answers to the test cases in the order in which they appear in the input. For each test case, print a single line summarizing the optimal route Johny can take (see example output for exact format). You may assume that such a route always exists.

Example

Some lines in the example test data below were too long and had to be wrapped. You can access full sample tests from the Subway Test clarification.

Input
3 Stops: OxfordCircus, PiccadillyCircus, HydeParkCorner, King'sCross, GreenPark, Arsenal, Victoria, Highbury&Islington, LeicesterSquare Lines: Blue, Cyan Cyan route: Highbury&Islington, King'sCross, OxfordCircus, GreenPark, Victoria Blue route: HydeParkCorner, GreenPark, PiccadillyCircus, LeicesterSquare, King'sCross, Arsenal Johny lives at King'sCross Michelle lives at GreenPark Stops: OxfordCircus, PiccadillyCircus, HydeParkCorner, King'sCross,GreenPark, Arsenal, Victoria, Highbury&Islington, LeicesterSquare Lines: Blue, Cyan Cyan route: Highbury&Islington, King'sCross, OxfordCircus, GreenPark, Victoria Blue route: HydeParkCorner, GreenPark, PiccadillyCircus, LeicesterSquare, King'sCross, Arsenal Johny lives at PiccadillyCircus Michelle lives at LeicesterSquare Stops: OxfordCircus, PiccadillyCircus, HydeParkCorner, King'sCross, GreenPark, Arsenal, Victoria, Highbury&Islington, LeicesterSquare Lines: Blue, Cyan Cyan route: Highbury&Islington, King'sCross, OxfordCircus, GreenPark, Victoria Blue route: HydeParkCorner, GreenPark, PiccadillyCircus, LeicesterSquare, King'sCross, Arsenal Johny lives at Victoria Michelle lives at HydeParkCorner
Output
optimal travel from King'sCross to GreenPark: 1 line, 3 minutes optimal travel from PiccadillyCircus to LeicesterSquare: 1 line, 1 minute optimal travel from Victoria to HydeParkCorner: 2 lines, 7 minutes

Problem E. Catalan Square

Source file name:	catalan.c, catalan.cpp, catalan.java
Input:	`standard`
Output:	`standard`

Last weekend you and your friends went to visit the local farmer's market at the town square. As you were standing around in a circle talking, you couldn't help overhearing two of your friends musing over what sounded like an interesting problem: They were considering the number of ways in which you could all shake hands, such that everyone in the circle simultaneously shaked hands with one other person, but where no arms crossed each other.

Photo by Wikimedia Commons user Rauenstein

After a few seconds' thought you decided to join your two friends, to share (with them) the solution to their problem. "If we are $2n$ persons", you said, "pick any particular person, and let that person shake hands with somebody. That person will have to leave an even number of people on each side of the person with whom he/she shakes hands. Of the remaining $n-1$ pairs of people, he/she can leave zero on the right and $n-1$ pairs on the left, 1 on the right and $n-2$ pairs on the left, and so on. The pairs remaining on the right and left can independently choose any of the possible non-crossing handshake patterns, so the count C_n for n pairs of people is given by:

$$C_n = C_{n-1}C_0 + C_{n-2}C_1 + \ldots + C_1C_{n-2} + C_0C_{n-1},$$

which, together with the fact that $C_0 = C_1 = 1$, is just the definition of the Catalan numbers." By consulting your handy combinatorics book, you find out that there is a much more efficient formula for calculating C_n, namely:

$$C_n = \frac{\binom{2n}{n}}{n+1}.$$

After a collective groan from the group, your particularly cheeky friend Val called out "Well, since we are at the town square, why don't you try to square your Catalan numbers!". This was met with much rejoicing, while you started to think about how to square the *Catalan sequence*...

Now your task: let C_n be the nth Catalan number as defined above. By regarding the sequence $(C_n)_{n\geq 0}$ of Catalan numbers, we can define a sequence $(S_n)_{n\geq 0}$, corresponding to "squaring the Catalan sequence", by considering the Cauchy product, or discrete convolution, of $(C_n)_{n\geq 0}$ with itself, i.e.,

$$S_n = \sum_{k=0}^{n} C_k C_{n-k}.$$

Your task is to write a program for calculating the number S_n.[1]

[1] To see why $(S_n)_{n\geq 0}$ could be said to correspond to the square of the Catalan sequence we could look at Cauchy products of power series. Suppose that $p(x) = \sum_{n=0}^{\infty} a_n x^n$ and $q(x) = \sum_{n=0}^{\infty} b_n x^n$, then $p(x) \cdot q(x) = \sum_{n=0}^{\infty} c_n x^n$ where $c_n = \sum_{k=0}^{n} a_k b_{n-k}$.

Input

The input contains one line containing one non-negative integer: n, with $0 \leq n \leq 5\,000$.

Output

Output a line containing S_n.

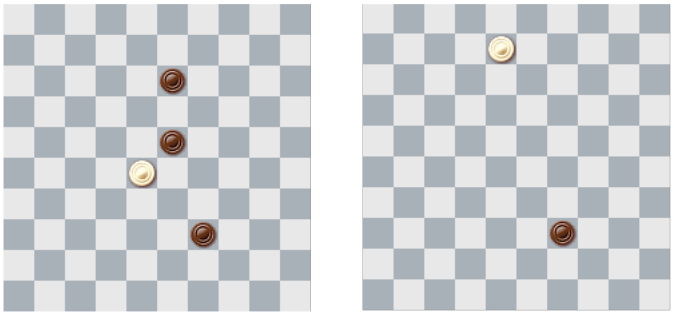
Example

Input	Output
0	1
59	158385096459612004268677277903889 6

Problem F. Draughts

Source file name:	draughts.c, draughts.cpp, draughts.java
Input:	`standard`
Output:	`standard`

Draughts (or *checkers*) is a game played by two opponents, on opposite sides of a 10 x 10 board. The board squares are painted black and white, as on a classic chessboard. One player controls the dark, and the other the light pieces. The pieces can only occupy the black squares. The players make their moves alternately, each moving one of his own pieces. The most interesting type of move is *capturing*: if a diagonally adjacent square contains an opponent's piece, it may be captured (and removed from the game) by jumping over it to the unoccupied square immediately beyond it. It is allowed to make several consecutive captures in one move, if they are all made with a single piece. It is also legal to capture by either forward or backward jumps.



The board before and after a single move with two captures.

You are given a draughts position. It is the light player's turn. Compute the maximal possible number of dark pieces he can capture in his next move.

Input

The first line of input contains the number of test cases T . The descriptions of the test cases follow: Each test case starts with an empty line. The following 10 lines of 10 characters each describe the board squares. The characters # and **.** denote empty black and white squares, **W** denotes a square with a light piece, **B** - a square with a dark piece.

Output

For each test case print a single line containing the maximal possible number of captures. If there is no legal move (for example, there are no light pieces on the board), simply output 0.

Example

Input	Output
2 .#.#.#.#.# #.#.#.#.#. .#.#.B.#.# #.#.#.#.#. .#.#.B.#.# #.#.W.#.#. .#.#.#.#.# #.#.#.B.#. .#.#.#.#.# #.#.#.#.#. .#.#.#.#.# #.#.#.#.#. .#.#.B.#.# #.B.#.B.#. .#.#.B.#.# #.B.W.#.#. .#.B.B.#.# #.#.#.#.#. .#.B.B.#.# #.#.#.#.#.	2 4

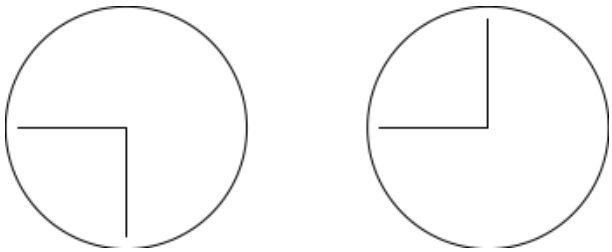
Problem G. Clock Pictures

Source file name:	clockpic.c, clockpic.cpp, clockpic.java
Input:	`standard`
Output:	`standard`

You have two pictures of an unusual kind of clock. The clock has n hands, each having the same length and no kind of marking whatsoever. Also, the numbers on the clock are so faded that you can't even tell anymore what direction is up in the picture. So the only thing that you see on the pictures, are n shades of the n hands, and nothing else.

You'd like to know if both images might have been taken at exactly the same time of the day, possibly with the camera rotated at different angles.

Now your task is: Given the description of the two images, determine whether it is possible that these two pictures could be showing the same clock displaying the same time.



Sample input 2

Input

The first line contains a single integer n ($2 \leq n \leq 200\,000$), the number of hands on the clock.

Each of the next two lines contains n integers a_i ($0 \leq a_i < 360\,000$), representing the angles of the hands of the clock on one of the images, in thousandths of a degree. The first line represents the position of the hands on the first image, whereas the second line corresponds to the second image. The number a_i denotes the angle between the recorded position of some hand and the upward direction in the image, measured clockwise. Angles of the same clock are distinct and are not given in any specific order.

Output

Output one line containing one word: `possible` if the clocks could be showing the same time, `impossible` otherwise

Example

Input	Output
6 1 2 3 4 5 6 7 6 5 4 3 1	impossible
2 0 270000 180000 270000	possible
7 140 130 110 120 125 100 105 235 205 215 220 225 200 240	impossible

Problem H. Chain & Co

Source file name:	chainco.c, chainco.cpp, chainco.java
Input:	`standard`
Output:	`standard`

Chain & Co. specializes in producing infinitely strong chains. Because of their high quality products, they are quickly gaining market share. This leads to new challenges, some of which they could have never imagined before. Like, for example, automatic verification of link endurance with a computer program, which you are supposed to write.

The company produces *links* of equal size. Each link is an infinitely thin square frame in three dimensions (made of four infinitely thin segments).

During tests all links are axis-aligned (Axis-aligned means that all segments are parallel to either X, Y, or Z axis.) and placed so that no two frames touch. To make a proper strength test, two sets of links A and B are forged so that every link of A is inseparable from every link of B (being inseparable means that they cannot be moved apart without breaking one of them).

You stumble upon some links (axis-aligned, pairwise disjoint). Are they in proper testing position? In other words, can they be divided into two non-empty sets A and B with the desired property?

Input

The first line of input contains the number of test cases T . The descriptions of the test cases follow: The description of each test case starts with an empty line. The next line contains an integer $n, 1 \leq n \leq 10^6$ - the number of links in the chain. Each of the next n lines contains 6 space-separated integers x_i, y_i, z_i , x'_i , y'_i, z'_i, all between -10^9 and 10^9 - the coordinates of two opposite corners of the i-th link.

Output

For each test case, print a single line containing the word **YES** if the set is in proper testing position, or **NO** otherwise.

Example

Input	Output
3 2 0 0 0 0 10 10 -5 5 15 5 5 25 5 0 0 0 0 10 10 -5 5 6 5 5 16 -5 5 -6 5 5 4 -5 6 5 5 16 5 -5 -6 5 5 4 5 3 0 0 0 3 0 -3 1 -1 -1 1 2 -4 -1 -2 -2 2 1 -2	NO YES YES

Problem I. Opening Ceremony

Source file name:	opening.c, opening.cpp, opening.java
Input:	`standard`
Output:	`standard`

For the grand opening of the algorithmic games in NlogNsglow, a row of tower blocks is set to be demolished in a grand demonstration of renewal. Originally the plan was to accomplish this with controlled explosions, one for each tower block, but time constraints now require a hastier solution.

To help you remove the blocks more rapidly you have been given the use of a Universal Kinetic / Incandescent Energy Particle Cannon (UKIEPC). On a single charge, this cutting-edge contraption can remove either all of the floors in a single tower block, or all the x-th floors in all the blocks simultaneously, for user's choice of the floor number x. In the latter case, the blocks that are less than x floors high are left untouched, while for blocks having more than x floors, all the floors above the removed x-th one fall down by one level.

Now your task is: Given the number of floors of all towers, output the minimum number of charges needed to eliminate all floors of all blocks.

Input

The first line of input contains the number of blocks n, where $2 \leq n \leq 100\,000$. The second line contains n consecutive block heights h_i for $i = 1, 2, \ldots, n$, where $1 \leq h_i \leq 1\,000\,000$.

Output

Output one line containing one integer: the minimum number of charges needed to tear down all the blocks.

Example

Input	Output
6 2 1 8 8 2 3	5
5 1 1 1 1 10	2

Problem J. Captain Obvious and the Rabbit-Man

Source file name:	captain.c, captain.cpp, captain.java
Input:	`standard`
Output:	`standard`

"It's you, Captain Obvious!" - cried the evil Rabbit-Man - "you came here to foil my evil plans!"

"Yes, it's me." - said Captain Obvious.

"But... how did you know that I would be here, on 625 Sunflower Street?! Did you crack my evil code?"

"I did. Three days ago, you robbed a bank on 5 Sunflower Street, the next day you blew up 25 Sunflower Street, and yesterday you left quite a mess under number 125. These are all powers of 5. And last year you pulled a similar stunt with powers of 13. You seem to have a knack for Fibonacci numbers, Rabbit-Man."

*"That's not over! I will learn... **arithmetics**!" - Rabbit-Man screamed as he was dragged into custody - "You will **never** know what to expect... Owww! Not my ears, you morons!"*

"Maybe, but right now you are being arrested." - Captain added proudly.

Unfortunately, Rabbit-Man has now indeed learned some more advanced arithmetics. To understand it, let us define the sequence F_n (being not completely unlike the Fibonacci sequence):

$F_1 = 1$, $F_2 = 2$, $F_n = F_{n-1} + F_{n-2}$ for $n \geq 3$.

Rabbit-Man has combined all his previous evil ideas into one master plan. On the i-th day, he does a malicious act on the spot number $p(i)$, defined as follows:

$p(i) = a_1 \cdot F_1^i + a_2 \cdot F_2^i + \ldots + a_k \cdot F_k^i$.

The number k and the integer coefficients a_1 , ..., a_k are fixed. Captain Obvious learned k, but does not know the coefficients. Given $p(1), p(2), \ldots, p(k)$, help him to determine $p(k+1)$. To avoid overwhelmingly large numbers, do all the calculations modulo a fixed prime number M. You may assume that F_1 , F_2 , ..., F_n are pairwise distinct modulo M. You may also assume that there always exists a unique solution for the given input.

Input

The first line of input contains the number of test cases T. The descriptions of the test cases follow: The first line of each test case contains two integers k and M , $1 \leq k \leq 4000$, $3 \leq M \leq 10^9$. The second line contains k space-separated integers - the values of $p(1), p(2), \ldots, p(k)$ modulo M.

Output

Print the answers to the test cases in the order in which they appear in the input. For each test case print a single line containing one integer: the value of $p(k+1)$ modulo M.

Example

Input	Output
2 4 619 5 25 125 6 3 101 5 11 29	30 83

Explication

Clarification of the example: the first sequence is simply 5^i mod 619, therefore the next element is 5^5 mod 619 = 30. The second sequence is $2 \cdot 1^i + 33^i$ mod 101.

Problem K. Outing

Source file name:	outing.c, outing.cpp, outing.java
Input:	`standard`
Output:	`standard`

Organising a group trip for the elderly can be a daunting task... Not least because of the fussy participants, each of whom will only make the trip on condition that some other participant also comes.

After some effort, you have taken from each of your participants a number, indicating that this participant will refuse to join the excursion unless the participant with that number also joins– the less choosy simply give their own number. This would be easy enough to resolve (just send all of them) but the bus you are going to use during the trip has only a fixed number of places.

Your task is given the preferences of all participants, find the maximum number of participants that can join.

Input

The first line of input contains two integers n and k ($1 \leq k \leq n \leq 1\,000$), where n denotes the total number of participants and k denotes the number of places on the bus.

The second line contains n integers x_i for $i = 1, 2, \ldots, n$, where $1 \leq x_i \leq n$. The meaning of x_i is that the i-th participant will refuse to join the excursion unless the x_i-th participant also joins.

Output

Output one integer: the maximum number of participants that can join the excursion, so that all the participants' preferences are obeyed and the capacity of the bus is not exceeded.

Example

Input	Output
4 4 1 2 3 4	4
12 3 2 3 4 5 6 7 4 7 8 8 12 12	2
5 4 2 3 1 5 4	3

Problem L. Bus

Source file name:	bus.c, bus.cpp, bus.java
Input:	`standard`
Output:	`standard`

A bus with n passengers opens its door at the bus stop. Exactly half of its passengers and an additional half of a passenger get out. On the next stop, again, half of the passengers plus half of a passenger leave the bus. This goes on for k stops in total. Knowing that the bus leaves the last stop empty, and that no one was hurt during the trip, determine the initial number n of people in the bus.

Input

The first line of input contains the number of test cases T. The descriptions of the test cases follow: The only line of each test case contains the number of stops k, $1 \leq k \leq 30$.

Output

For each test case, output a single line containing a single integer - the initial number of bus passengers.

Example

Input	Output
2 1 3	1 7