Problem A. Kolonie

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

In 1996, the Czech Technical University already organized the 2nd year of the programming contest. The problem set contained a very interesting problem about n-dimensional hyperex hypergons. This single problem gave a uniting theme to the whole 1996 CTU Open contest: all problems involved the launch of a fictional hyper-cosmic spaceship Nostromo. This quickly became a tradition and since then, CTU Open Contest problem sets have common themes.

We wanted to present the Hyperex Hypergons to you today but we finally decided that they are a little bit too difficult (no team did even *try* to solve it back in 1996), so we only recommend the problem to your attention — maybe you would like to give it a try and solve it after this contest is over? We will publish the problem statement on our web page. For this competition, we instead present you an English translation of another problem from the 1996 set.

An important task of the hyper-cosmic spaceship Nostromo is to establish a permanent base on the orbit of the planet MX8-26B in the Centaurus constellation. The base is built from complexes composed of identical hexagonal cubicles. Each of the six sides of each cubicle contains a hole serving either as a passage to a neighboring cubicle or as a window in case when no cubicle of the finished base is neighboring at this side. The complexes can be composed of the cubicles in many different ways.

The travelers have a given number of complexes of several shapes at their disposition. Every cubicle in the finished base accommodates as many people as the number of windows it has.

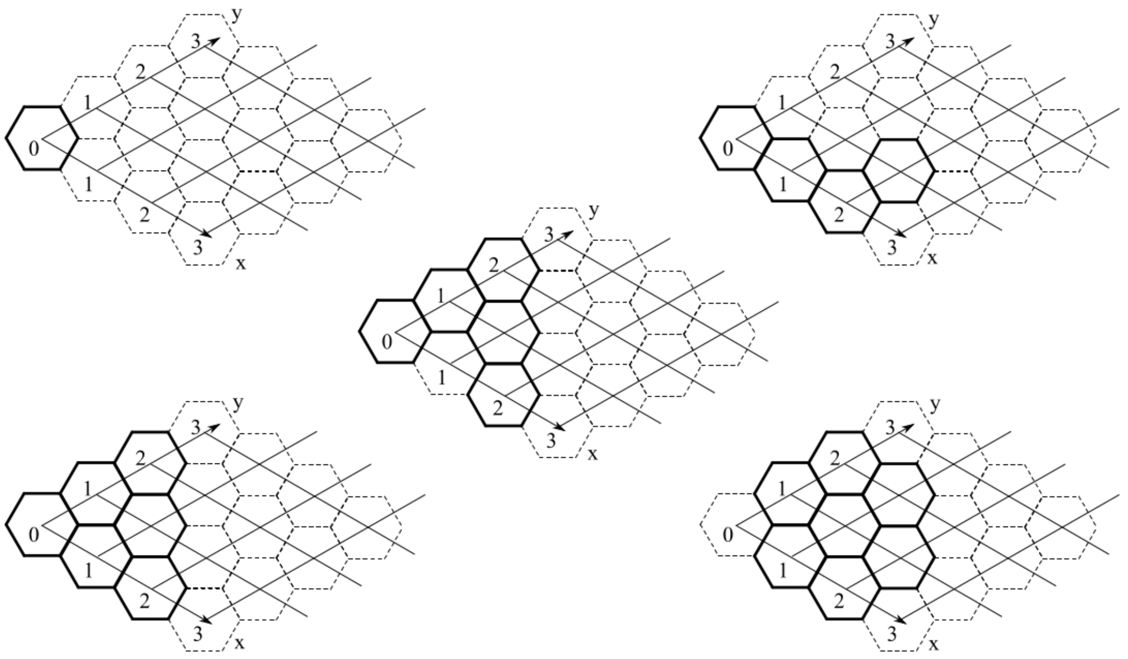
Write a program to determine whether it is possible to build a base for the prescribed number of people given the descriptions of available complexes.

Input

The first line of the input contains the number of test cases N . The first line of each test case contains two space-separated positive integers $P \leq 1\ 000\ 000$ and $T \leq 1000$, where P is the number of inhabitants for whom the base is to be built and T is the number of shapes of available complexes.

Each of the following T lines contains integers separated by spaces describing one shape of a complex. The first two numbers on each of these lines are integers C and S, where C $(0 \leq C \leq 1000)$ is the number of available complexes of this shape and S $(1 \leq S \leq 1000)$ is the number of cubicles that make up the complex. It is known that every complex is connected and the hexagonal bottom bases of its cubicles lie in one plane.

The following S pairs of integers are the x and y coordinates of the centers of the hexagonal bases of the cubicles in a hexagonal coordinate system. The coordinates satisfy $-10\ 000\ 000 \leq x, y \leq 10\ 000\ 000$. The angle between the x-axis of the hexagonal coordinate system and the x-axis of the Cartesian coordinate system is $-30°$. The angle between the y-axis of the hexagonal coordinate system and the x-axis of the Cartesian coordinate system is $+30°$. See the following figure with drawings of five shapes from the first test case of the sample input.



Output

For each test case, print exactly one line. If it is possible to build the base, the line says **"Je treba X celku."**, where X is the minimal possible number of complexes when the shapes and their connections are chosen optimally. Otherwise print **"Kapacita zakladny je pouze X lidi."**, where X is the maximal number of people that can fit inside the optimal base built from all of the available complexes.

Example

Input	Output
`3`	`Je treba 3 celku.`
`50 5`	`Kapacita zakladny je pouze 10 lidi.`
`10 1 0 0`	`Je treba 2 celku.`
`3 4 0 0 1 0 2 0 2 1`	
`4 5 0 0 0 1 0 2 1 1 2 0`	
`6 6 0 0 1 0 2 0 0 1 1 1 0 2`	
`1 7 1 0 2 0 0 1 1 1 2 1 0 2 1 2`	
`11 1`	
`2 1 0 0`	
`10 2`	
`100 1 1 1`	
`0 2 0 0 1 0`	

Problem B. Most

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

In 1997, the programming contest was quickly gaining popularity. For the second time, the Czech Technical University held two programming contests in one year: one for the Faculty of Electrical Engineering (with several guest teams) and, of course, the 1997 CTU Open Contest. By the way, in that year, the winning team of the Charles University then won the Regional Contest and later also the World Finals.

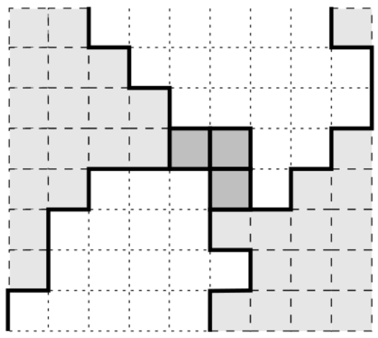
Today you can try to solve one of the problems from 1997, to get an impression.

Several years ago, the African tribe of Hooligans dug up a hatchet. The tribe's military uses advanced technology including machine guns and tanks. Such an equipment gives a huge advantage against the enemy,but it also brings complications during transport. The biggest problem are the numerous rivers and lakes present in the area.

Hooligan cartographers invented a precise transformation that maps the landscape in such a way that all boundaries between land and water are either vertical or horizontal and follow precisely the boundaries of unit squares in a square grid. Moreover, the special transformation algorithm causes all rivers to appear to flow vertically (from top to bottom) and all coordinates of the left river boundary are strictly smaller than any coordinate of the right boundary.

Luckily for the ambitious Hooligans, a great inventor named Postolomlatos invented mobile bridges made of easily transportable pieces, called pontoons, precisely of the size of the unit squares of the map. The pontoons can be connected to each other and to the river boundaries only by their whole sides. Due to the economical crisis, the chief Mlask the Great ordered that every river must be crossed by using the minimal number of pontoons needed to build a connected bridge from one side of the river to the other.

See the image with an example of a river with one possible correct solution using three pontoons.



Input

The first line of the input contains the number of test cases N . The first line of each test case contains a positive integer $K \leq 10\ 000\ 000$, which stands for the height of the map. Each of the following K lines describes one horizontal unit strip of the map. Each of these lines contains two space-separated non-negative integers A and B specifying the left and the right coordinates of the river boundary in that strip. It will always hold that $A < B \leq 1\ 000\ 000$.

Output

For each test case, print exactly one line containing the text **"K prechodu reky je treba X pontonu."** where X is the smallest number of pontoons needed to build a bridge from one side of the river to the other.

Example

Input	Output

2
8
2 8
3 9
4 9
4 8
2 7
1 5
1 6
0 5
5
1 7
2 7
4 8
5 6
0 6
	K prechodu reky je treba 3 pontonu.
	K prechodu reky je treba 1 pontonu.

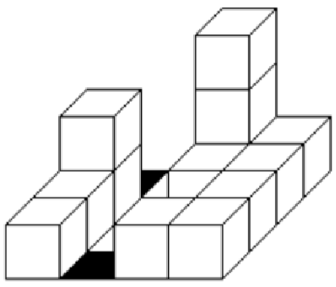
Problem C. Stavitel

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

1998 was the first year when the Central Europe Regional Contest was held in Prague, at the Czech Technical University. This meant having three programming contests in one year. It was also the year when a new evaluation system (PCSS 3) was implemented from scratch. This piece of software then served with only small improvements (and bug fixes) for the next 14 years, until 2011. Can you imagine that? In 1998, a typical contest had 20 teams and 6 problems. Our best computer had 64MB of memory, so PCSS had to impose many limits on data structures. In contrast, there were about 100 teams and 11 problems in 2011, so you can imagine many of the mentioned limits were exceeded several times during those 14 years.

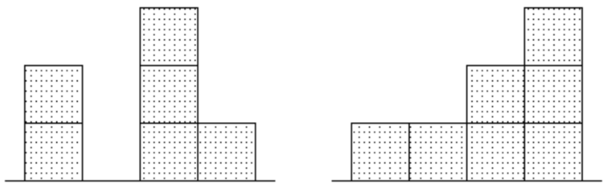
Now you have a unique opportunity to visit the past and try to solve one of the 1998 problems.

Little Matthew always wanted to be an architect and since his childhood he was spending all his free time building architectonic masterpieces. Due to his limited resources, he built his buildings from wooden unit cubes that he put on top of each other. The columns of the cubes were always placed on a checkerboard with $K \times K$ unit squares. Matthew always placed the cubes so that every column covered exactly one checkerboard square. In the following picture, you can admire one of his creations on a board of size 4×4.



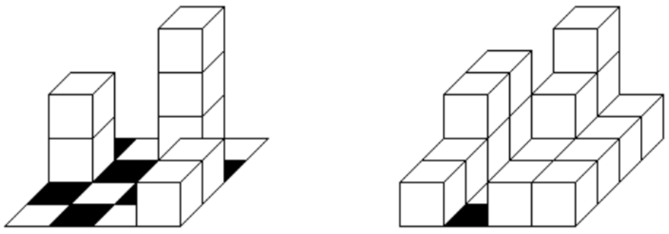
Matthew was proud of his pieces of art. Their order and perfection were unprecedented. However, nobody else shared his enthusiasm. As is usual with most masterpieces, their real value will be understood much later. To preserve his work, Matthew decided to draw the buildings on paper and keep it in a safe place. Because he was unable to draw 3D buildings, he made two 2D drawings: one from the front showing only the front sides of the cubes and one from the right side showing only the right sides of the cubes. In technical terms, his drawings were orthogonal projections.

Have a look at the drawings of the building from the earlier picture:



Matthew believed that these drawings will be enough to reconstruct the buildings later. When he grew up, he realized that he had been wrong. Most of his pairs of drawings could depict many different buildings. After some research, he found out that some buildings may be called *minimal* because they are composed of the minimum number L of cubes among all buildings whose pair of projections match his drawings. Similarly, he called a building *maximal* if it used the maximal number M of cubes among all possible buildings.

The following are examples of a minimal and a maximal building for the above pair of drawings. They use $L = 7$ and $M = 17$ cubes. They are not as perfect as the original building, but they are still worth your attention.



Matthew asked you to write a program that will compute the values L and M for every pair of drawings in his collection.

Input

The first line of the input contains the number of test cases N. Each test case is composed of three lines. The first line of each test case contains a positive integer $K \leq 100$, which stands for the width and the height of the square board. The second and the third line describe the drawing from the front and from the right, respectively. Every drawing is described by K space-separated non-negative integers (not higher than 100 000) describing the heights of the K columns of the cubes projection, in the left-to-right and front-to-back order.

You can assume that it is always possible to build at least one building for every given pair of drawings.

Output

For each test case, print exactly one line containing the sentence **"Minimalni budova obsahuje L kostek, maximalni M kostek."**, where L and M are the minimal and the maximal number of cubes in a building represented by the given pair of drawings.

Example

Input

```
2
4
2 0 3 1
1 1 2 3
1
1
1
```

Output

```
Minimalni budova obsahuje 7 kostek, maximalni 17 kostek.
Minimalni budova obsahuje 1 kostek, maximalni 1 kostek.
```

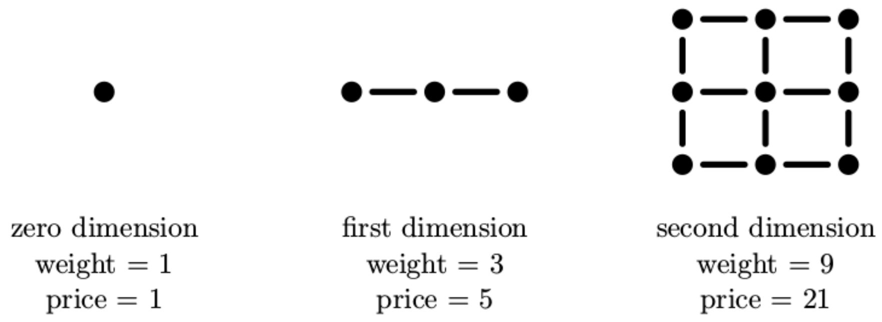
Problem D. Lodě

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

In 1999, there were another three contests at the Czech Technical University: faculty round (FEL++), CTU Open, and Central Europe Regional Contest. The enthusiasm of the "founding fathers" had decreased a little bit and there was no wonder. They had been organizing two or three competitions a year for a period of five years. What had once been fun turned into hard work. Fortunately, many new organizers arose from the former contestants, so the tradition could go on.

The problem Lodě was added to the 1999 contest at the very last moment and it was intended to be an easy one. Now you may find out yourselves how difficult or easy it was.

Juliet reads an interesting sci-fi book. In one chapter, the main character needs to solve a problem of maximizing the utilization of cargo spaceships. The ships transport valuable items that have the form of D-dimensional mesh with the size of 3 nodes in each dimension. The nodes are formed by balls of the same weight. The connections between balls are so light that their weight is negligible compared to the weight of balls. This means that the weight of any item is determined solely by the number of its nodes. On the other hand, the value of such an item is equal to the number of nodes *plus* the number of connections.



Each spaceships has a limited tonnage and we want to maximize the total value of items that may be stored in the ship without exceeding the tonnage. Your task is to decide what items of what dimension should be loaded to maximize their total value, providing that we have an unlimited supply of items of all dimensions.

Input

The first line of the input contains the number of test cases N. Each test case then consists of a single line containing one positive integer number $K < 10\ 000\ 000$ giving the ship cargo capacity.

Output

For each test case, print one line containing space-separated non-negative numbers $X_m X_{m-1} \ldots X_1 X_0$, where $X_m > 0$ and X_i $(0 \leq i \leq m)$ is the number of items of the i-th dimension that we need to store to maximize their total value.

Example

Input	Output

4
1
100
175
9841
	1
	1 0 2 0 1
	2 0 1 1 1
	1 1 1 1 1 1 1 1 1

Problem E. Direct Visibility

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

In 2000, the contest activities sort-of culminated at the Czech Technical University. There were another three competitions again (for the last time, probably), including the third Central Europe Regional Contest in a row. The region then moved to Warsaw and FEL++ was temporarily suspended.

We will end our small historical excursion here and present you the fifth and last problem that originates from our rich archives. It appeared in the 2000 Central Europe Regional Contest.

Building the GSM network is a very expensive and complex task. Moreover, after the *Base Transceiver Stations (BTS)* are built and working, we need to perform many various measurements to determine the state of the network, and propose effective improvements to be made.

The ACM technicians have a special equipment for measuring the strength of electro-magnetic fields, the transceiver power and signal quality. This equipment is packed into a huge knapsack and the technician must move with it from one BTS to another. Unfortunately, the knapsack has not enough memory for storing all of the measured values. It has a small cache only, that can store values for several seconds. Then the values must be transmitted to the BTS by an infrared connection (IRDA). The IRDA needs direct visibility between the technician and the BTS.

Your task is to find the path between two neighboring BTSes such that at least one them is always visible. For simplicity, a town is modelled as a rectangular grid of $P \times Q$ square fields. Each field is exactly 1 meter wide. For each field, a non-negative integer $Z_{i,j}$ is given, representing the height of the terrain in that place, in meters. That means the town model is made of cubes, each of them being either solid or empty.

The technician is moving in steps (*steps* stands for *Standard Technician's Elementary Positional Shift*). Each step is made between two neighboring square fields in North, South, West or East direction, it is not possible to move diagonally. The step between two fields A and B (step from A to B) is allowed only if the height of the terrain in the field B is not very different from the height in the field A. The technician can climb at most 1 meter up or descend at most 3 meters down in a single step.

At the end of each step, at least one of the two BTSes must be visible. However, there can be some point "in the middle of the step" where no BTS is visible (data are handled by the cache). The BTS is considered visible if there is a direct visibility between the unit cube just above the terrain on the BTSes coordinates and the cube just above the terrain on the square field where the technician is. Direct visibility between two cubes means that the line segment connecting the centers of the two cubes does not intersect any solid cube. However, the line can touch any number of solid cubes. In other words, consider both the BTS and the technician being points exactly half meter above the surface and in the center of the appropriate square field.

Note that the IRDA beam can also go between two cubes that touch each other by their edge, although there is no real space between them. It is because such a beam touches both of these two cubes but does not intersect them.

Input

There is a single positive integer T on the first line of input. It stands for the number of test cases to follow. The first line of each test case contains two integer numbers P and Q, separated by a single space, $1 \le P, Q \le 200$. Then there are P lines, each containing Q integer numbers separated by a space. These numbers are $Z_{i,j}$, where $1 \le i \le P, 1 \le j \le Q$ and $0 \le Z_{i,j} \le 5000$.

After the terrain description, there are four numbers R_1, C_1, R_2, C_2 on the last line of each test case. These numbers represent positions of two BTSes, $1 \le R_1, R_2 \le P, 1 \le C_1, C_2 \le Q$. The first coordinate determines the row of the town, the second coordinate determines the column.

Output

You are to find the shortest possible path meeting the above criteria. That is, all steps must be done between neighboring fields, the terrain must not elevate or descend too much, and at the end of each step, at least one BTS must be visible.

For each test case, print one line containing the sentence **"The shortest path is M steps long."**, where M is the number of steps that must be made. If there is no such path, output the sentence **"Mission impossible!"**.

Example

Input	Output
`4`	`The shortest path is 10 steps long.`
`5 5`	`Mission impossible!`
`8 7 6 5 4`	`The shortest path is 14 steps long.`
`2 2 2 2 2`	`The shortest path is 18 steps long.`
`2 2 2 2 2`	
`2 2 2 2 2`	
`2 2 2 2 2`	
`1 1 5 1`	
`5 8`	
`2 2 2 2 2 2 2 2`	
`2 2 2 2 2 2 2 2`	
`2 2 2 2 2 2 2 2`	
`9 9 9 9 9 9 9 2`	
`2 2 2 2 2 2 2 2`	
`1 2 5 1`	
`5 8`	
`2 2 2 2 2 2 2 2`	
`2 2 2 2 2 2 2 2`	
`2 2 2 2 2 2 2 2`	
`9 9 9 9 9 9 9 2`	
`2 2 2 2 2 2 2 2`	
`1 5 5 1`	
`6 12`	
`5 5 5 5 1 5 5 5 5 5 5 5`	
`5 5 5 5 1 5 5 5 5 5 5 5`	
`5 5 5 5 9 5 5 5 5 5 5 5`	
`5 9 1 5 5 5 5 5 5 5 5 5`	
`5 5 9 5 5 5 5 5 5 5 5 5`	
`5 5 9 5 5 5 5 5 5 5 5 5`	
`6 1 3 12`	

Problem F. Karel the robot

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

Karel wants to register his robot Karel for a robot contest. The aim of the contest is to escape from a maze. The maze is a square grid W units wide and H units high where every unit square is either a wall or free terrain. One of the squares with free terrain is designated as an exit.

The design of the robot was inspired by the ancient (it is even much older than CTU Open) educational programming language called Karel. The robot understands three commands: Step, Right, and Left. When executing the Step command, the robot moves to the neighboring square in the forward direction if possible. If the square one unit ahead is a wall or it lies outside the maze, the robot does not move. The Right command turns the robot 90° clockwise and the Left command turns it 90° counterclockwise. The robot has a memory for a program consisting of up to 10 commands. After the robot executes the last command, it continues executing the program from the beginning.

The robot is initially positioned at some free square. We do not know its position but we know it is facing up (towards the neighboring square one row above). Karel wants to write a universal program that will make the robot escape from *any* such initial position. That means, the robot will eventually reach the exit during the program execution.

This sounds like a nice contest problem, doesn't it? We want to give you an idea what it is like to organize a programming contest. Therefore, your task is to write a validator for the problem described above. (You may read more about validators in the *validate* problem.)

Input

The input contains several test cases. The first line of each input contains two space-separated integers: H and W satisfying $1 \leq H, W \leq 100$. Each of the following H lines contains exactly W characters describing the maze. The character "X" means a wall, "." is free terrain, and "E" is free terrain with the exit.

The next line contains a single integer L, $1 \leq L \leq 10$, the length of the program. The last line of the test case contains L characters describing the commands of the program. The character "S" stands for Step, "L" for Left, and "R" for Right.

Output

Print a single line for each test case. The line must contain "OK" if the robot escapes for every initial position using the given program. Otherwise, the line contains the number of initial positions (including the exit), from which the robot escapes successfully.

Example

Input	Output
3 4 E... X.X. .... 3 SSL 3 3 E.. ... ... 3 RSS	5 OK

Problem G. More or Less Accurate

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

During the last 20 years, the Czech Technical University organized not only 20 nation-wide (and later even international) competitions known as CTU Open but also 7 internal competitions of the Faculty of Electrical Engineering (FEL++) and 5 Central Europe Regional Contests (CERC, in 1998, 1999, 2000, 2007, and 2011). As one of the original organizers has pointed out, together there were 0x20 contests during those 20 years (not speaking about hosting the World Finals in 2004). Seven years ago, in CERC 2007, the contestants were given a problem called Weird Numbers dealing with numeric systems using a negative base. The problem assignment said:

A number N written in the system with a positive base R will always appear as a string of digits between 0 and $R-1$, inclusive. A digit at the position P (positions are counted from right to left and starting with zero) represents a value of R^P . This means the value of the digit is multiplied by R^P and values of all positions are summed together. For example, if we use the octal system (radix $R = 8$), a number written as 17024 has the following value:

$$1 \cdot 8^4 + 7 \cdot 8^3 + 0 \cdot 8^2 + 2 \cdot 8^1 + 4 \cdot 8^0 = 1 \cdot 4096 + 7 \cdot 512 + 2 \cdot 8 + 4 \cdot 1 = 7700$$

With a negative radix $-R$, the principle remains the same: each digit will have a value of $(-R)^P$. For example, a negaoctal (radix $-R = -8$) number 17024 counts as:

$$1 \cdot (-8)^4 + 7 \cdot (-8)3 + 0 \cdot (-8)2 + 2 \cdot (-8)1 + 4 \cdot (-8)0 = 1 \cdot 4096 - 7 \cdot 512 - 2 \cdot 8 + 4 \cdot 1 = 500$$

One big advantage of systems with a negative base is that we do not need a minus sign to express negative numbers. A couple of examples for the negabinary system ($-R = -2$):

decimal	negabinary	decimal	negabinary	decimal	negabinary
-10	1010	-3	1101	4	100
-9	1011	-2	10	5	101
-8	1000	-1	11	6	11010
-7	1001	0	0	7	11011
-6	1110	1	1	8	11000
-5	1111	2	110	9	11001
-4	1100	3	111	10	11110

You may notice that the negabinary representation of any integer number is unique, if no "leading zeros" are allowed. The only number that can start with the digit "0" is the zero itself.

Today, we are interested whether there were any contestants' answers in 2007 that were *almost* correct, i.e., their program output was different from the correct answer only by one. Will you help us to find out?

Input

The input contains several test cases. Each test case is given on a single line containing number X written in the negabinary notation. The line contains N ($1 \leq N \leq 10^6$) characters "0" or "1" representing the negabinary bits $a_{N-1} \ldots a_1 a_0$ respectively. Numbers will be given to you without leading zeros, i.e., for each input where $X \neq 0$ it holds that $a_{N-1} = 1$.

Output

For each test case, print a single line with number $(X + 1)$ written in the negabinary notation. Output the number without any leading zeros.

Example

Input	Output
1 0 100 11 10101	110 1 101 0 1101010

Problem H. Self-Intersecting Path

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

Karel wants to register his robot Karel for a robot contest. The aim of the contest is to escape from a maze using a program consisting of N instructions. Each instruction is in the form: "MOVE a_i METERS FORWARD, THEN TURN 90° TO THE RIGHT", where a_i is a positive integer. We can simply encode the whole program for the robot as the sequence of integers $a_1 a_2 a_3 \ldots a_N$ representing the lengths of particular steps.

For example, if the robot starts at coordinates [0, 0] facing north and the encoded program is 1 2 3 4 5, the robot would end up at coordinates [-2, 3] facing east. An important property of any valid program is that the path the robot takes is not allowed to intersect itself at any point.

This sounds like a nice contest problem, doesn't it? We want to give you an idea what it is like to organize a programming contest. Therefore, your task is to write a validator for the problem described above. (You may read more about validators in the *validate* problem.)

Input

The input contains several test cases. Each test case consists of two lines. The first line contains a single integer N $(1 \leq N \leq 10^6)$, the number of instructions that form the robot's program. The second line contains N space-separated integers $a_1 a_2 a_3 \ldots a_N$ $(1 \leq a_i \leq 10^9)$, an encoded program for the robot, as described above.

Output

For each test case, print exactly one line. If the given program describes a path that does not intersect itself, print "OK". Otherwise output a single integer M $(0 \leq M < N)$, the maximum number of instructions from the beginning of the program that describe a valid path. That means the path described by the program consisting of instructions $a_1 a_2 a_3 \ldots a_M$ does not intersect itself and M is maximal with this property.

Example

Input	Output

`7`
`3 1 1 3 2 2 6`
`3`
`2 1 1`
`6`
`2 1 4 4 4 3`
	`3`
	`OK`
	`5`

Problem I. Dice Game

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

Gunnar and Emma play a lot of board games at home, so they own many dice that are not normal 6-sided dice. For example they own a die that has 10 sides with numbers $47, 48, \ldots, 56$ on it.

There has been a big storm in Stockholm, so Gunnar and Emma have been stuck at home without electricity for a couple of hours. They have finished playing all the games they have, so they came up with a new one. Each player has 2 dice which he or she rolls. The player with a bigger sum wins. If both sums are the same, the game ends in a tie.

Your task is: Given the description of Gunnar's and Emma's dice, which player has higher chances of winning?

All of their dice have the following property: each die contains numbers $a, a+1, \ldots, b$, where a and b are the lowest and highest numbers respectively on the die. Each number appears exactly on one side, so the die has $b - a + 1$ sides.

Input

The first line contains four integers a_1, b_1, a_2, b_2 that describe Gunnar's dice. Die number i contains numbers $a_i, a_i + 1, \ldots, b_i$ on its sides. You may assume that $1 \le a_i \le b_i \le 100$. You can further assume that each die has at least four sides, so $a_i + 3 \le b_i$.

The second line contains the description of Emma's dice in the same format.

Output

Output the name of the player that has higher probability of winning. Output "`Tie`" if both players have same probability of winning. q

Example

Input	Output

`1 4 1 4`
`1 6 1 6`
	`Emma`

`1 8 1 8`
`1 10 2 5`
	`Tie`

`2 5 2 7`
`1 5 2 5`
	`Gunnar`

Problem J. Train Passengers

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

The Nordic Company of Passing Carriages is losing money at an alarming rate because most of their trains are empty. However, on some lines the passengers are complaining that they cannot fit in the cars and have to wait for the next train!

The authorities want to fix this situation. They asked their station masters to write down, for a given train, how many people left the train at their station, how many went in, and how many had to wait. Then they hired your company of highly paid consultants to assign properly sized trains to their routes.

You just received the measurements for a train, but before feeding them to your optimisation algorithm you remembered that they were collected on a snowy day, so any sensible station master would have preferred to stay inside their cabin and make up the numbers instead of going outside and counting.

Verify your hunch by checking whether the input is inconsistent, i.e., at every time the number of people in the train did not exceed the capacity nor was below 0 and no passenger waited in vain. The train should start and finish the journey empty, in particular passengers should not wait for the train at the last station. At each station passengers first leave, then others enter.

Input

The first line contains two integers C and n ($1 \le C \le 10^9$, $2 \le n \le 100$), the total capacity and the number of stations the train stops in. The next n lines contain three integers each, the number of people that left the train, entered the train, and had to stay at a station. Lines are given in the same order as the train visits each station. All integers are between 0 and 10^9 inclusive.

Output

One line containing one word: `possible` if the measurements are consistent, `impossible` otherwise.

Example

Input	Output

1 2
0 1 1
1 0 0
	possible

1 2
1 0 0
0 1 0
	impossible

1 2
0 1 0
1 0 1
	impossible

1 2
0 1 1
0 0 0
	impossible