CTU Open Contest 2014

# $>>>$ Celebrating the Twentieth Anniversary $<<<$

Twenty years! Can you imagine the world in 1994? We mean, the world of *computers*? Most people did not know about the Internet yet (for a comparison, Seznam.cz was started in the end of 1995, which means our contest is older). A group of young enthusiastic students at the Czech Technical University discovered the ACM International Programming Contest and persuaded one of the teachers to register their team. And she really did...

The CTU team participated in the 1994 South-Western Europe Regional Contest in Zürich and they liked it so much they decided to organize a similar contest at the Czech Technical University on May 13, 1995. They could not know they just started a long tradition of the CTU Open Contest. Today, October 18, 2014, you have a privilege to participate in the 20th year of this competition. We hope you are going to enjoy it at least as much as the "founding fathers" did back in 1994/5. And, by the way, you may meet some of them at the Closing Ceremony.

In 1995, the programs had to be written in Pascal or C. Today, you can use C, C++, or Java programming languages. The choice is yours but you are fully responsible for the correctness and efficiency of your solutions. We need the correct answer produced in some appropriate time. Nothing else matters. You may choose any algorithm and any programming style.

All programs will read text from the standard input only. The results must be written to the standard output. You are not allowed to use any other files, communicate over network, or create processes. Input and output formats are described in problem statements and must be strictly followed. Do not print anything more than required. Each text line (including the last one) should be terminated by a newline character ("\n"), which is not considered to be a part of that line. If not specified otherwise, all input and output numbers will fit into a signed 32-bit integer.

Good luck in the 20th Czech Technical University Open Contest!
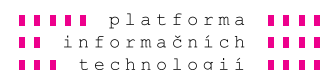
This problem set consists of eleven sheets of paper (including this one) containing ten problems. Please make sure that you have the complete set.

Czech ACM Student Chapter   **Czech Technical University in Prague**

Charles University in Prague   Technical University of Ostrava
Slovak University of Technology   Pavol Jozef Šafárik University in Košice
University of Žilina   Masaryk University
Matej Bel University in Banská Bystrica   University of West Bohemia

CTU Open Contest 2014

Kolonie

`kolonie.c`, `kolonie.cpp`, `Kolonie.java`

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 Specification

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.
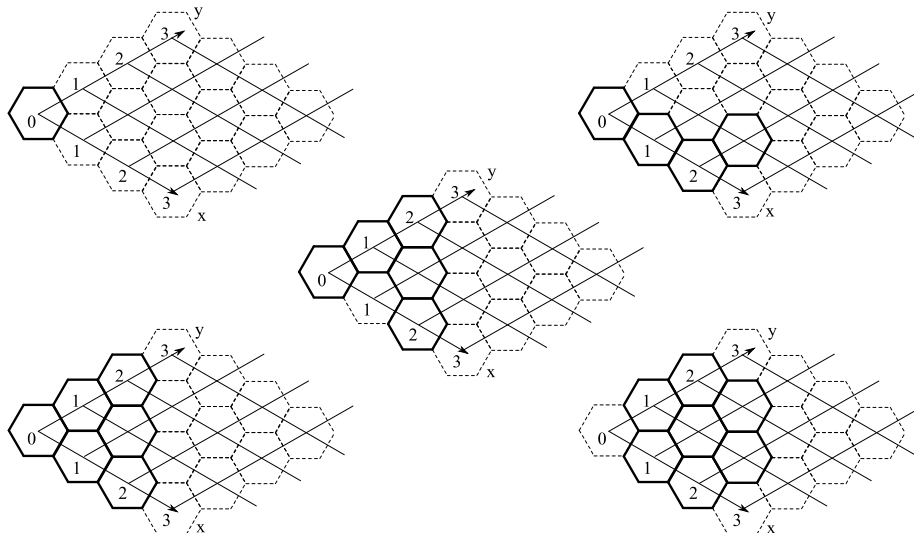
Output Specification

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.

Sample Input

```
3
50 5
10 1 0 0
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
```

Output for Sample Input

```
Je treba 3 celku.
Kapacita zakladny je pouze 10 lidi.
Je treba 2 celku.
```

Czech ACM Student Chapter

Charles University in Prague
Slovak University of Technology
University of Žilina
Matej Bel University in Banská Bystrica

Czech Technical University in Prague

Technical University of Ostrava
Pavol Jozef Šafárik University in Košice
Masaryk University
University of West Bohemia

CTU Open Contest 2014

Most

`most.c`, `most.cpp`, `Most.java`

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.
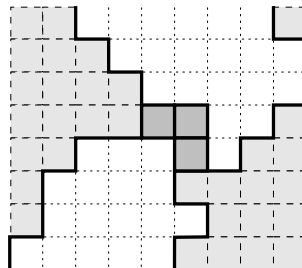
Input Specification

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 Specification

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.

Sample Input

```
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
```

Output for Sample Input

```
K prechodu reky je treba 3 pontonu.
K prechodu reky je treba 1 pontonu.
```

Czech ACM Student Chapter

Charles University in Prague
Slovak University of Technology
University of Žilina
Matej Bel University in Banská Bystrica

Czech Technical University in Prague

Technical University of Ostrava
Pavol Jozef Šafárik University in Košice
Masaryk University
University of West Bohemia

**CTU Open Contest 2014**

Stavitel

`stavitel.c`, `stavitel.cpp`, `Stavitel.java`

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.
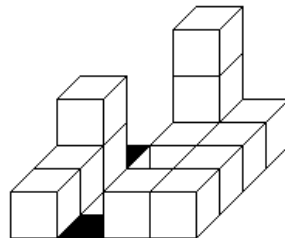
Matthew was proud of his pieces of art. Their order and perfection were unprecedented. However, nobody else shared his enthusiasm. As is usual with most masterpieces, their real value will be understood much later. To preserve his work, Matthew decided to draw the buildings on paper and keep it in a 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:
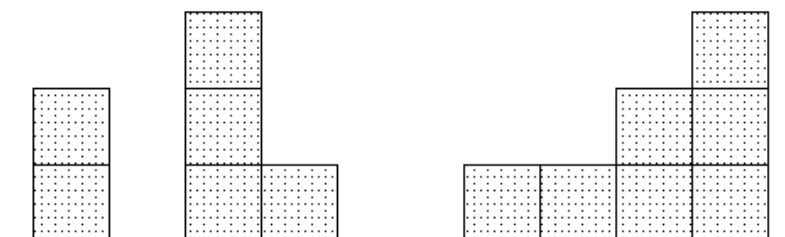
Matthew believed that these drawings will be enough to reconstruct the buildings later. When he grew up, he realized that he had been wrong. Most of his pairs of drawings could depict many different buildings. After some research, he found out that some buildings may be called *minimal* because they 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.
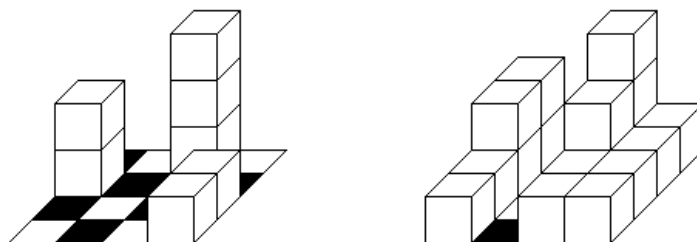
Matthew asked you to write a program that will compute the values L and M for every pair of drawings in his collection.

Input Specification

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 Specification

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.

Sample Input	Output for Sample Input
2	Minimalni budova obsahuje 7 kostek, maximalni 17 kostek.
4	Minimalni budova obsahuje 1 kostek, maximalni 1 kostek.
2 0 3 1	
1 1 2 3	
1	
1	
1	

Czech ACM Student Chapter

Charles University in Prague
Slovak University of Technology
University of Žilina
Matej Bel University in Banská Bystrica

Czech Technical University in Prague

Technical University of Ostrava
Pavol Jozef Šafárik University in Košice
Masaryk University
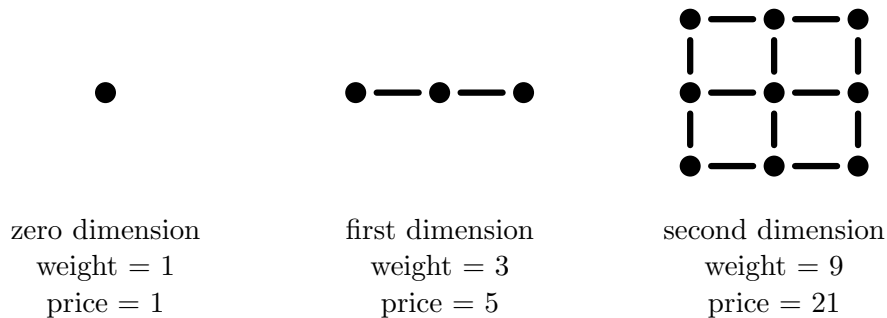University of West Bohemia

CTU Open Contest 2014

Lodě

`lode.c`, `lode.cpp`, `Lode.java`

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.

zero dimension	first dimension	second dimension
weight = 1	weight = 3	weight = 9
price = 1	price = 5	price = 21

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 Specification

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 Specification

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

Sample Input

```
4
1
100
175
9841
```

Output for Sample Input

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

Czech ACM Student Chapter

Charles University in Prague
Slovak University of Technology
University of Žilina
Matej Bel University in Banská Bystrica

Czech Technical University in Prague

Technical University of Ostrava
Pavol Jozef Šafárik University in Košice
Masaryk University
University of West Bohemia

CTU Open Contest 2014

Direct Visibility

`visible.c`, `visible.cpp`, `Visible.java`

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 Specification

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, $1 \leq P, Q \leq 200$. Then there are P lines, each containing Q integer numbers separated by a space. These numbers are $Z_{i,j}$, where $1 \leq i \leq P$, $1 \leq j \leq Q$ and $0 \leq Z_{i,j} \leq 5000$. 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, $1 \leq R_1, R_2 \leq P$, $1 \leq C_1, C_2 \leq Q$. The first coordinate determines the row of the town, the second coordinate determines the column.

Output Specification

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!".

Sample Input

```
4
5 5
8 7 6 5 4
2 2 2 2 2
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
```

Output for Sample Input

```
The shortest path is 10 steps long.
Mission impossible!
The shortest path is 14 steps long.
The shortest path is 18 steps long.
```

Czech ACM Student Chapter   **Czech Technical University in Prague**

Charles University in Prague   Technical University of Ostrava
Slovak University of Technology   Pavol Jozef Šafárik University in Košice
University of Žilina   Masaryk University
Matej Bel University in Banská Bystrica   University of West Bohemia

CTU Open Contest 2014

Karel the Robot

`karel.c`, `karel.cpp`, `Karel.java`

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 Specification

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 Specification

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.

Sample Input

```
3 4
E...
X.X.
....
3
SSL
3 3
E..
...
...
3
RSS
```

Output for Sample Input

```
5
OK
```

Czech ACM Student Chapter   **Czech Technical University in Prague**

Charles University in Prague   Technical University of Ostrava
Slovak University of Technology   Pavol Jozef Šafárik University in Košice
University of Žilina   Masaryk University
Matej Bel University in Banská Bystrica   University of West Bohemia

CTU Open Contest 2014

More or Less Accurate

`more.c`, `more.cpp`, `More.java`

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 Specification

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 $a_{N-1}...a_1a_0$ 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 Specification

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

Sample Input

```
1
0
100
11
10101
```

Output for Sample Input

```
110
1
101
0
1101010
```

Czech ACM Student Chapter

Charles University in Prague
Slovak University of Technology
University of Žilina
Matej Bel University in Banská Bystrica

Czech Technical University in Prague

Technical University of Ostrava
Pavol Jozef Šafárik University in Košice
Masaryk University
University of West Bohemia

CTU Open Contest 2014

Self-Intersecting Path

`self.c`, `self.cpp`, `Self.java`

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 ... 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 Specification

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

Output Specification

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 ($0 \le M < N$), 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 ... a_M does not intersect itself and M is maximal with this property.

Sample Input	Output for Sample Input
7	3
3 1 1 3 2 2 6	OK
3	5
2 1 1	
6	
2 1 4 4 4 3	

Czech ACM Student Chapter

Charles University in Prague
Slovak University of Technology
University of Žilina
Matej Bel University in Banská Bystrica

Czech Technical University in Prague

Technical University of Ostrava
Pavol Jozef Šafárik University in Košice
Masaryk University
University of West Bohemia

CTU Open Contest 2014

Lengthy Traveling Salesman

`lengthy.c`, `lengthy.cpp`, `Lengthy.java`

Karel wants to register his robot Karel for a robot contest. To do so, he needs to obtain a certificate of safety of his robot, fill in various waivers, etc. He needs to start at the entrance of the building of the contest organizers, visit all prescribed offices in a particular order, and return back to the entrance. Fortunately, the building is easy to navigate, as its hallways do not form any cycles, and thus there is only one path between any two offices. However, Karel would like to know in advance how long it will take to visit all the offices.

Given a building layout and an ordering of all offices $v_1, v_2, \ldots, v_n$, the task is to determine the length of the walk going from v_1 to v_2, then from v_2 to v_3, \ldots, and finally from v_n back to v_1.

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. If you want to organize one (please contact us after this contest if you *really* do), writing the textual problem statement and sample solutions is not enough. We also need to prepare the test data that will be used to verify the correctness of submissions. Beside others, we want to eliminate inefficient solutions, so we want to generate a permutation of offices that results in the maximal possible walk. Your task is to find it.
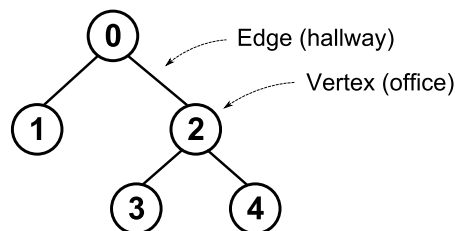
For the purpose of this problem, the building is described as a tree with one office in each of its vertices. For those who are not familiar with graph terms, a short informal definition follows: A *tree* consist of a set of *vertices* connected by *edges* (each edge connects two vertices) in such a way that there is exactly one possible path between any two vertices v_i and v_j. The *length* of the path is the number of edges that must be traversed when traveling from v_i to v_j and it is denoted $dist(v_i, v_j)$.

Input Specification

The input contains several test cases. The first line of each test case contains an integer N, giving the number of the vertices in a tree ($2 \leq N \leq 10\,000$). The vertices are numbered from 0 to $N - 1$. The i-th of the following $N - 1$ lines contains an integer f_i ($0 \leq f_i < i$), indicating that the tree contains an edge between vertices i and f_i.

There is one empty line after each test case.

Output Specification

For each test case, print a single line containing N integers $v_1, \ldots, v_n$ separated by spaces. Each of the numbers $0, 1, \ldots, N-1$ must appear exactly once. The sum

$$\sum_{i=1}^{n-1} dist(v_i, v_{i+1}) \quad + \quad dist(v_n, v_1)$$

should be the maximum possible. If there are multiple solutions (as they usually are), you may print any of them.

Sample Input

```
5
0
0
2
2

2
0
```

Output for Sample Input

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

Czech ACM Student Chapter

Charles University in Prague
Slovak University of Technology
University of Žilina
Matej Bel University in Banská Bystrica

Czech Technical University in Prague

Technical University of Ostrava
Pavol Jozef Šafárik University in Košice
Masaryk University
University of West Bohemia

CTU Open Contest 2014

Picture Validator

`validate.c`, `validate.cpp`, `Validate.java`

Karel wants to register his robot Karel for a robot contest. The aim of the contest is to draw a picture using a program composed of the following instructions:

- L X_{rel} Y_{rel} — draw a line segment between the current robot position and the given relative coordinates. The robot moves to the end point of the segment.

- M X_{rel} Y_{rel} – move the current position by the given relative offset without drawing a line.

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. All contests held at the Czech Technical University (since the very first one on May 13, 1995) always employed the automated judging of submissions. The output of a contestants' program is compared with the correct answer by a computer, without the need of human intervention. This may seem obvious today but we should realize that back in the old years, many programming contests in the world used human judges to issue the final verdict. Manual judging then continued for many years in many other regions.

With some contest problems, there is more than one correct answer. In these cases, a special program (called *validator*) has to be prepared to review the submission output and to issue a final verdict. Do you want to try out what it is like to write such a program?

Your task is to prepare a validator for the problem described above. The validator reads a picture generated by a contestants' submission and decides whether it is equivalent to another picture — the correct answer prepared by judges.

Two pictures are considered equivalent if they are composed of the same visible line segments. Note however that the drawing instructions may differ. For instance, some of the segments may be divided into several shorter segments, or even drawn repeatedly. Also note that pictures with a different scale or rotation are considered different.

Input Specification

The input contains several test cases. Each test case consists of two pictures. Each picture starts with a row containing one positive integer N, the number of drawing instructions to follow ($1 \leq N \leq 5000$). Then there are N lines, each containing one instruction. The instruction consists of an uppercase letter, one space and two integers X_i, Y_i separated with another space ($|X_i|, |Y_i| \leq 1000$). The first letter will either be "L" or "M".

There is one empty line after each test case. You may assume that no real (absolute) coordinate will be further than $30\,000$ in any direction from the picture starting point.

Output Specification

For each test case, print one line containing either "`YES`" if the two pictures are equivalent, or "`NO`" if they differ.

Sample Input

```
2
L 1 1
L 1 1
1
L -2 -2

2
L 1 1
L 1 1
1
L 3 3

5
L 1 0
L 1 0
L 0 5
L 0 -5
L 2 0
3
L 0 5
M -2 -5
L 4 0
```

Output for Sample Input

```
YES
NO
YES
```