Nine Invaders from Outer Space

It is 2091. After many years, the SETI@home project finally brings some results. Unfortunately, these results are not quite exactly what the authors expected. The creatures from space turned out to be hostile aliens. The Earth faces a danger of their invasion. Would you help to save the human race? We need a couple of programs to make the fight easier.

Your programs can be written in C, C++, Pascal, or Java programming languages. The choice is yours but you will be 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 one single text file from the standard input. The results will be written to the standard output. Input and output formats are described in problem statements and must be strictly followed. Each text line (including the last one) should be always terminated by a newline character ("`\n`"), which is not considered to be a part of that line.

You are not allowed to use any other files, communicate over network, or create processes.

A small advice: If not specified otherwise, all *input* numbers and output *results* will fit into a signed 32-bit integer type. Note however, that some intermediate results may exceed 2^{31}, even if it sometimes may not be obvious.

Good luck in the Czech Technical University Open Contest 2011!

And remember, the future is in your hands!

This problem set consists of 11 sheets of paper (including this one) and contains nine problems. Please make sure that you have the complete set.

Analog Clock Display

`analog.c`, `analog.C`, `analog.java`, `analog.p`

Our old clock chimed four o'clock, and with that last sound they dissolved into a pile of sticks. Since the craftsmanship necessary to fix them was forgotten and lost long time ago, we decided to replace them by a computer program.

An easy task, you say? There is one more little thing to mention: Recent studies of extraterrestrial aliens showed that they live in a digital world and therefore they are unable to read the traditional clock face with two moving hands. It is simply something beyond their technical capabilities. Therefore, the Security Board decided that our new clock must use such a traditional "analog" display to protect the time information against non-humans.

Input Specification

The input contains several instances, each of them consisting of one row containing two integers H ($0 \leq H \leq 23$) and M ($0 \leq M \leq 59$) separated by a colon. M is always given with two digits, H has one or two digits, i.e., no leading zero unless $H = 0$.

The last test case is followed by the word "`END`".

Output Specification

For each input instance H:M, draw an "ASCII art" clock face depicting the time of H hours and M minutes according to the following specification.

The face frame is a square, with 51 characters per each side. These characters are uppercase letters "`X`", with the exception of four corners and every tenth character, which is always "`@`". The numbers 3, 6, 9, and 12 are centered at their appropriate sides, with exactly one space between them and the frame. There is one space between digits 1 and 2. The center of the face always contains the asterisk symbol: "`*`". All visually "empty" characters are simple spaces.

Two hands are the only elements of the clock face that depend on the time. In the following description of the placement of the hands, we assume that each character (both space or occupied) is a 1×1 square and the start $(0, 0)$ of the coordinate system is placed in the *center* of the square containing the asterisk symbol, with the first axis pointing to the right, and the second axis upwards.

The hour hand is drawn as a line segment of length 15 starting at $(0, 0)$. The hand points upwards at 12 o'clock and moves uniformly by the same angle each minute. Similarly, the minute hand is drawn as a line segment of length 21 starting at $(0, 0)$. That hand points upwards every hour and also moves uniformly by the same angle every minute in the clockwise direction (what a surprise). The minute hand is considered to be *above* the hour hand, i.e., the characters representing the hour hand may be hidden by parts of the minute hand.

A *line* whose angle from the vertical direction is D degrees should be drawn as follows. If $0 \leq D \leq 45$, one character is printed for each row i at (integer) coordinates (n_i, i) as close as possible to the point (x_i, i) that lies exactly on the geometric line (x_i is a *real* number). If $45 \leq D \leq 90$, there is one character printed for each column i at the (integer) coordinates (i, k_i), the closest possible square to the (real) point (i, y_i) on the line.

The character displayed to draw the line at some position (i, j) depends on the two "neighboring" characters of the line. The character is

- minus symbol "`-`" if there are also characters at both positions $(i-1, j)$ and $(i+1, j)$,
- pipe symbol "`|`" if there are also characters at both positions $(i, j-1)$ and $(i, j+1)$,
- backslash symbol "`\`" if there are characters at positions $(i-1, j+1)$ and $(i+1, j-1)$,
- slash symbol "`/`" if there are also characters at positions $(i-1, j-1)$ and $(i+1, j+1)$,
- lowercase letter "`o`" otherwise.

A line *segment* of length S starting at $(0, 0)$ is drawn by displaying characters in the same way as drawing the corresponding line, but we only print such characters whose distance between the *center* of the square and the origin $(0, 0)$ is *at most* S, $0 < |(0, 0), (i, j)| \leq S$, and only in one direction of the line.

Please see the sample output to resolve any ambiguities in the above description. Print one empty line after each clock face.

Sample Input

```
1:03
14:27
END
```

Output for Sample Input

```
@XXXXXXXXX@XXXXXXXXX@XXXXXXXXX@XXXXXXXXX@XXXXXXXXX@
X                                                 X
X                       1 2                       X
X                                                 X
X                                                 X
X                              o                  X
X                              |                  X
X                              |                  X
X                              o                  X
X                             o                   X
@                             |                   @
X                             o                   X
X                            o                    X
X                            |  o                 X
X                            o  o                 X
X                           o  o                  X
X                           |  o                  X
X                           o /                   X
X                          o o                    X
X                          | o                    X
@                          o/                     @
X                         oo                      X
X                         |o                      X
X                         o                       X
X                        oo                       X
X 9                      *                      3 X
X                                                 X
X                                                 X
X                                                 X
X                                                 X
@                                                 @
X                                                 X
X                                                 X
X                                                 X
X                                                 X
X                                                 X
X                                                 X
X                                                 X
X                                                 X
X                                                 X
@                                                 @
X                                                 X
X                                                 X
X                                                 X
X                                                 X
X                                                 X
X                                                 X
X                                                 X
X                        6                        X
X                                                 X
@XXXXXXXXX@XXXXXXXXX@XXXXXXXXX@XXXXXXXXX@XXXXXXXXX@

@XXXXXXXXX@XXXXXXXXX@XXXXXXXXX@XXXXXXXXX@XXXXXXXXX@
X                                                 X
X                       1 2                       X
X                                                 X
X                                                 X
X                                                 X
X                                                 X
```

```
X                                                 X
X                                                 X
X                                                 X
@                                                 @
X                                                 X
X                                                 X
X                                                 X
X                                                 X
X                                                 X
X                                                 X
X                                                 X
X                                                 X
X                                                 X
@                                                 @
X                                    o--          X
X                                 o-o             X
X                              o-o                X
X                          o--o                   X
X 9                      *o                     3 X
X                        o                        X
X                         o                       X
X                         |                       X
X                         o                       X
@                          o                      @
X                          |                      X
X                          o                      X
X                           o                     X
X                           |                     X
X                           o                     X
X                            o                    X
X                            |                    X
X                            o                    X
X                             o                   X
@                             |                   @
X                             o                   X
X                              o                  X
X                              |                  X
X                              |                  X
X                              o                  X
X                                                 X
X                                                 X
X                        6                        X
X                                                 X
@XXXXXXXXX@XXXXXXXXX@XXXXXXXXX@XXXXXXXXX@XXXXXXXXX@
```

Notice

If you get a *presentation error* for this problem, it will likely mean wrong formatting, usage of wrong characters, bad clock size, missing spaces, etc. On the other hand, *wrong answer* typically means incorrect content of the clock face, although it is technically also a “presentation” issue.

Collatz Conjecture

`collatz.c`, `collatz.C`, `collatz.java`, `collatz.p`

The Collatz Conjecture is an interesting phenomenon. Though its principle is very simple, it still remains among unresolved problems in mathematics, even after many years of study. However, the years of intensive research brought at least some results, which is a huge advantage of the human race against the aliens, because they did not study the conjecture for so many years. We want to keep this advantage.

Imagine a sequence defined recursively as follows: Start with any positive integer x_0 (so-called "starting value"). Then repeat the following:

- if x_i is even, then $x_{i+1} = x_i/2$ ("half ...")
- if x_i is odd, then $x_{i+1} = 3x_i + 1$ ("... or triple plus one")

The Collatz Conjecture says that every such sequence will eventually reach 1. It has still not been proven until today but we already know for sure that this is true for every $x_0 < 2^{58}$. (Never tell this to aliens!)

In this problem, you are given two starting values and your task is to say after how many steps their sequences "meet" for the first time (which means the *first* number that occurs in both sequences) and at which number is it going to happen. For simplicity, we will assume that the sequence does not continue once it has reached the number one. In reality, it would then turn into $1, 4, 2, 1, 4, 2, 1, \ldots$, which quickly becomes boring.

Input Specification

The input contains several test cases. Each test case is described by a single line containing two integer numbers A and B, $1 \leq A, B \leq 1\,000\,000$.

The last test case is followed by a line containing two zeros.

Output Specification

For each test case, output the sentence "A `needs` S_A `steps,` B `needs` S_B `steps, they meet at` C", where S_A and S_B are the number of steps needed in both sequences to reach the same number C. Follow the output format precisely.

Sample Input

```
7 8
27 30
0 0
```

Output for Sample Input

```
7 needs 13 steps, 8 needs 0 steps, they meet at 8
27 needs 95 steps, 30 needs 2 steps, they meet at 46
```

Text Encryption

`encipher.c`, `encipher.C`, `encipher.java`, `encipher.p`

To keep privacy of messages and prevent the aliens from reading them, we may use various encryption algorithms. These algorithms encode a message into the so-called *ciphertext* that is difficult (or impossible) to decode for anyone else than the intended recipient. *Transposition ciphers* are a type of encryption that do not change the letters of the message but only change their order ("shuffle" the letters). Of course, the shuffling must be reversible to allow later decryption.

In this problem, we will consider a simple transposition cipher which shuffles the letters in such a way that the *decryption* algorithm always takes every n-th letter. More specifically: when decrypting, the first letter of the ciphertext is taken first, then the next $n - 1$ letters are (repeatedly) skipped and the next letter taken, and so on until we reach the end of the ciphertext. After that, we repeat the procedure starting with the second letter of the ciphertext, and so on until all letters are used.

Your task is to implement the encryption algorithm for this cipher. For a given message, produce the encrypted text (ciphertext). To make the cipher a little bit stronger, you should convert all letters to uppercase and leave out all spaces between words.

Input Specification

The input contains several messages. Each message is described by two lines. The first line contains one integer number N ($1 \leq N \leq 1000$). The second line contains the message. The message will be at most 10 000 characters long, it will only contain letters and spaces, and there will be at least one letter in each message.

The last message is followed by a line containing zero.

Output Specification

For each message, output the ciphertext that, after using the described decryption algorithm, will result in the original message (with all spaces removed and all letters in uppercase).

Sample Input

```
2
CTU Open Programming Contest
7
This is a secret message that noone should ever see Lets encrypt it
15
text too short
0
```

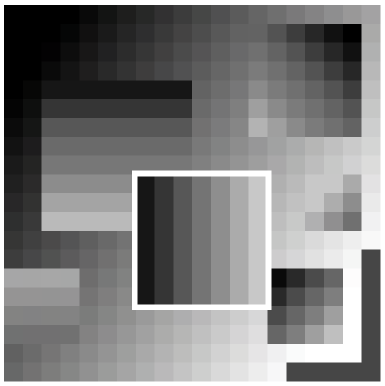
Output for Sample Input

```
CMTMUIONPGECNOPNRTOEGSRTA
TESNUECHCAOLERIRGODLYSEENEEPITTEVTTSMHSESIAEAHRETSSTOSN
TEXTTOOSHORT
```

Mine the Gradient

`gradient.c`, `gradient.C`, `gradient.java`, `gradient.p`

If we need to be prepared for the possible danger of an alien invasion, the first thing to do is to find out where can they come from. One way to determine if a planet is inhabited by intelligent creatures is to study high resolution pictures of planets trying to find typical characteristics of effects of an intelligent life. There are so many habitable planets that a computer program must be written for this task. One distinctive feature of a colonized planets is the presence of surface mines. An alien mine is a square structure with a depth uniformly decreasing from one of the sides of the square towards the opposite.



Your task is to find the largest mine on a provided bitmap image of a planet. The picture is a rectangular grid of numbers between 0 and 65535, representing the shades of grey. Mines will appear as squares of either the same shade or with shade levels gradually and uniformly changing from one side to another. See the above figure for an example. Your program will consider only square mines in some special orientations.

An axis-parallel square is the set of pixels (i, j) such that $c_1 \leq i \leq c_2$ and $r_1 \leq j \leq r_2$ for some c_1, c_2, r_1, r_2, $c_2 - c_1 = r_2 - r_1$.

A vertically (horizontally) oriented mine is an axis-parallel square for which there exist integers S and K such that the shade of every pixel (i, j) of the square is equal to $S + iK$ ($S + jK$ for horizontally oriented mines).

A diagonally oriented mine is an axis-parallel square for which there exist integers $Q \in \{1, -1\}$, S and K such that the shade of every pixel (i, j) of the square is equal to $S + (i + Qj)K$.

Input Specification

The input contains several descriptions of pictures. The first line of each description contains two numbers N and M ($1 \leq N, M \leq 2000$), the height and width of the picture. The following N lines contain M space-separated integers each — there is at least one space character between numbers but there may be more spaces and also additional spaces at the beginning or end of the line are allowed. The j-th number in the i-th row $A_{i,j}$ ($0 \leq A_{i,j} \leq 65535$) describes the shade of grey of the pixel in the ith row and jth column of the picture bitmap.

The last description is followed by a line containing two zeros.

Output Specification

For each picture, output the area (number of pixels inside) of the largest horizontal, vertical, or diagonal mine.

Sample Input

```
4 4
10   1 13 20
18   9 11 13
 5   7  9  6
 6   5  7  7
3 3
10   1    13
18   9    11
 5 1000   9
4 4
10  12 15 20
 5   9 13 10
 5   9 13  6
 5   9 13  7
0 0
```

Output for Sample Input

```
4
1
9
```

Invasion

`invasion.c`, `invasion.C`, `invasion.java`, `invasion.p`

Alien invasion began, and scary man-eating aliens are establishing their bases all over the country. You are only safe on the places that are sufficiently far away from all current alien bases. You need to quickly write a program to help you determine where to move.

Input Specification

The input contains several instances, each of them consisting of several lines. The first line of each instance contains integers N ($1 \leq N \leq 10\,000$), M ($0 \leq M \leq 100\,000$), A ($0 \leq A \leq N$) and K ($1 \leq K \leq 100$) separated by spaces, giving the number of towns in the country, the number of roads between them, the number of bases that the aliens are going to build, and the minimum safe distance from alien bases, respectively. The towns are assigned numbers $1, \ldots, N$.

The following M lines describe the roads; each of them contains integers T_1, T_2 ($1 \leq T_1 < T_2 \leq N$) and D ($1 \leq D \leq 100$) separated by spaces, where D is the length of the road between towns T_1 and T_2. There is at most one direct road between any two towns. All roads can be used in both directions.

The following A lines describe the positions of alien bases; the i-th of them contains the number B_i ($1 \leq B_i \leq N$) of the town where the aliens build their i-th base.

Each instance is followed by one empty line. The empty line after the last instance is followed by a line containing four zeros.

Output Specification

The output for each input instance consists of A lines. On i-th line, write the number of towns that are safe after the aliens build their i-th base. The town is safe if its distance from *any* of the towns B_1, B_2, $\ldots$, B_i is at least K.

Print one empty line after each instance.

Sample Input

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

1 0 1 1
1

0 0 0 0
```

Output for Sample Input

```
2
1
0

0
```

Intergalactic Mortgage

`mortgage.c`, `mortgage.C`, `mortgage.java`, `mortgage.p`

Many people on Earth want to solve their housing. When they have not enough money to buy their house or flat, they get mortgage[*] from some bank and then they pay fixed monthly payments until they redeem the mortgage.

You may think that paying mortgage for 30 years is a long time, but that is quite short time compared to galactic mortgages. Aliens are not buying houses, but whole planets. Since planets are a "little" more expensive, the mortgage periods are longer.

The mortgages work the same way for aliens as for us earthlings. If an alien wants to buy a planet, he comes to GCB (Galactic Central Bank) to borrow an amount of X. Bank offers a mortgage with the *interest rate* $r\%$ p.a. (= "per year"). Interests are computed at the end of each month (1 alien year has 12 months). At the end of every month, the current debt is raised by $(r/12)\%$ and then the alien pays back to bank some fixed amount Y, which is subtracted from the debt.

Because of intergalactic financial crisis, bank rules are quite strict. Every mortgage must start on the first day of a year. If an alien does not pay enough money to cover the principal and interests within first N years, the bank will then confiscate his planet.

On the other hand, galactic employment works quite nice. Once you have a job, you are guaranteed to have it forever. An alien can give the same amount Y at the end of each month for the whole mortgage period.

Your task is to decide whether an alien is able to pay his mortgage or not.

Input Specification

The input contains several test cases. Each test case is described by a line containing numbers X, Y, N, r separated by space. X is principal (the initial amount borrowed), Y is the monthly payment (paid at the end of each month), N is number of years in which the alien is required to pay the mortgage, r is interest rate p.a. in percent.

X, Y are integer numbers ($1 \leq X, Y \leq 1\,000\,000\,000$). N is integer number ($1 \leq N \leq 10\,000$). r is float ($0 \leq r \leq 100$, 2 digits precision). Values X, Y, N, r for each test case were chosen so that even if the alien would not pay anything for the whole time, the resulting debt after N year would be at most 10^{25}. Also, the precision of double should be sufficient for most computations (differences in the rate less then $10^{-8}\%$ will not affect the result).

The last test case is followed by a line containing four zeros.

[*]hypotéka

Output Specification

For each test case output "`YES`" if the alien can pay the mortgage within N years and "`NO`" if his salary is too small to pay the mortgage on time.

Sample Input

```
10000 500 2 5.00
10000 400 2 5.00
10000 245 100 30.00
321321321 2895492 11 3.23
0 0 0 0
```

Output for Sample Input

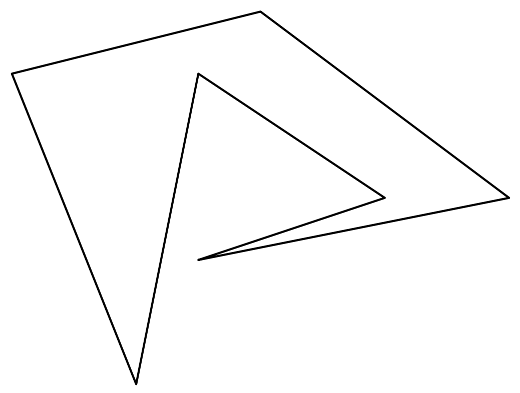
```
YES
NO
NO
NO
```

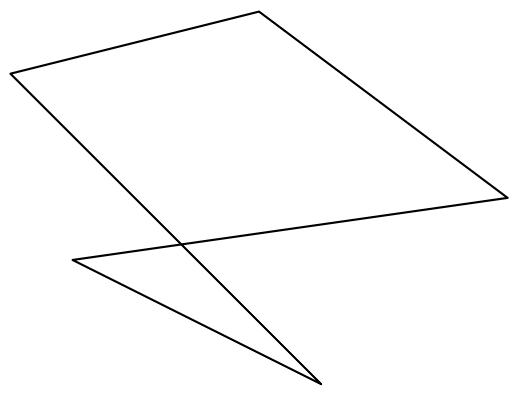
Simple Polygon

`polygon.c`, `polygon.C`, `polygon.java`, `polygon.p`

A *polygon* P determined by points p_1, p_2, …, p_n is a closed chain of line segments (called *edges*) p_1p_2, p_2p_3, …, p_np_1 in the plane. Polygon P is *simple*, if no two edges have any points in common, with the obvious exception of two consecutive segments having one common point (called *vertex*). Note however, that if a vertex is part of any other (third) edge, the polygon is no longer simple.

Any polygon that is not simple is called *self-intersecting*. In two example figures below, the first polygon is simple, the second one is self-intersecting.





Your task is to determine whether a given polygon is simple or self-intersecting.

Input Specification

The input contains several test cases. Each test case corresponds to one polygon. First line of the test case contains N, the number of points ($1 \leq N \leq 40\,000$). Each of the following N lines contains coordinates of point P_i, that is X_i, Y_i separated by space, $1 \leq X_i, Y_i \leq 30\,000$.

The last test case is followed by a line containing zero.

Output Specification

For each test case output either "`YES`" (the polygon is simple) or "`NO`" (the polygon is self-intersecting).

Sample Input

```
5
1 6
5 7
9 4
2 3
6 1
7
1 6
5 7
9 4
4 3
7 4
4 6
3 1
7
1 1
1 4
1 3
2 2
3 1
3 3
2 2
0
```

Output for Sample Input

```
NO
YES
NO
```

Ambiguous Result

`result.c`, `result.C`, `result.java`, `result.p`

The ACM (Advanced Cosmos Monitor) recorded a set of messages transmitted by alien race of Space Invaders. Unfortunately, the antenna used for recording only handles lower frequencies representing numbers and two arithmetical operators in space-invaderian language, while all parentheses (corresponding to a high frequency) were lost.

Since numbers are important for those 8-bit creatures, we really need to know what number ranges these messages belong to — please, write a program that can do this for us!

Input Specification

Input contains several legal arithmetical expressions, each expression on a separate line. Each expression consists only of non-negative integers x_i ($0 \leq x_i \leq 100$) and binary operators "`+`" and "`*`". The expression starts with a number, then the operators and numbers alternate, and the last element is a number. Each expression contains P numbers ($1 \leq P \leq 100$) and $P-1$ operators. There are no parentheses, no other operators, no unary operator, etc.

The last input expression is followed by a line containing the single word "`END`".

Output Specification

For each input line (not counting the final END), output one line containing the minimum and maximum values (separated by a single space) that are achievable by adding parentheses to the input in a way that forms a legal expression and computing the result of that expression.

For example, the minimum value for $2+1*0$ input is achieved by $(2+1)*0$ and the maximum value is achieved by $2+(1*0)$. Therefore, you should print "`0 2`".

It is guaranteed that for *any* placement of parentheses, the value of *each* parenthesis will be less then 2^{63}. This means that also the maximal result will be between 0 and $2^{63}-1$, inclusive.

Sample Input

```
2+1*0
3+2*5+1*7+16
0
END
```

Output for Sample Input

```
0 2
36 690
0 0
```

Domino tiling

`tiling.c`, `tiling.C`, `tiling.java`, `tiling.p`

The aliens want to dominate the whole universe. So it comes as no surprise that their favorite game is played with *domino* tiles. A domino tile is a tile of size 1×2 with a digit 0-9 written on each half. Their game board is a rectangular array with a digit 0-9 written in each unit square. The task is to cover the whole board with the set of provided domino tiles. The tiles can be placed only if the two numbers on the tile equal the two numbers in the unit squares covered by the tile. The tiles can be placed as they are or may be rotated by 90°, 180° or 270°. No two tiles may overlap. You can always use only one piece of each provided tile.

Some tiles are already placed on the board. These must stay where they are.

You are to write a program that will play this game. If you do not succeed, our planet will be *domin*ated!

Input Specification

The input contains several descriptions of game settings. The first line of each description contains three numbers separated by one or more spaces. The first two numbers are the height M and the width N of the board. They satisfy $1 \leq M \leq 20$, $1 \leq N \leq 20$, at least one of them is even, and their product $2 \leq M \cdot N \leq 110$ (guess why). The last number is the number of available tiles K, $1 \leq K \leq M \cdot N/2$.

The next line contains K pairs of space-separated integers describing the pairs of numbers on the available tiles. Each two consecutive numbers on this line are separated by at least one space. In addition, no two tiles are identical, that is, they are different (and stay different even if one is rotated by 180°). The tiles already placed on the board are not among these K tiles.

The following M lines contain N space-separated entries each. For every $i, j, 0 \leq i < M, 0 \leq j < N$, the jth entry in the ith row describes the place in the ith row and jth column of the gameboard. The entry is either the capital letter "`X`" if there is an already-placed tile, or a number $A_{i,j}$ ($0 \leq A_{i,j} \leq 9$) written on the board.

Every description is followed by an empty line and the empty line after the last description is followed by a line containing three zeros.

Output Specification

For each game, find a way to place all tiles onto the board. If there are more solutions, output any of them. Provide the solution by a graphic representation as an $M \times N$ array with "`[`" and "`]`" (square brackets) standing for the left and right half of a horizontally placed domino, and lowercase letters "`n`" and "`u`" for the upper and lower half of a vertically placed domino. The squares covered by a tile in the input are still represented by "`X`". After the M rows, print one line containing the number of other *different* solutions that exist.

Do not output any spaces and start a new line for each row of the gameboard.

If there is no solution, write a single line with the word "`impossible`".

Print one empty line after the each gameboard result.

Sample Input

```
4 5 9
0 0  0 1  1 1  3 3  0 2  1 2    0 3    2 2    2 3
1 2 2 0 X
2 1 0 0 X
2 1 3 3 3
2 3 0 1 0

2 3 3
1 1 2 2 3 3
1 2 3
1 3 2

2 3 3
1 1 2 2 3 3
1 2 3
1 2 3

0 0 0
```

Output for Sample Input

```
[][]X
nn[]X
uun[]
[]u[]
3

impossible

nnn
uuu
0
```

Problem J. Saving the Universe

he urban legend goes that if you go to the Google homepage and search for “Google”, the universe will implode. We have a secret to share... It is true! Please don’t try it, or tell anyone. All right, maybe not. We are just kidding.

The same is not true for a universe far far away. In that universe, if you search on any search engine for that search engine’s name, the universe does implode!

To combat this, people came up with an interesting solution. All queries are pooled together. They are passed to a central system that decides which query goes to which search engine. The central system sends a series of queries to one search engine, and can switch to another at any time. Queries must be processed in the order they’re received. The central system must never send a query to a search engine whose name matches the query. In order to reduce costs, the number of switches should be minimized.

Your task is to tell us how many times the central system will have to switch between search engines, assuming that we program it optimally.

Input

The first line of the input file contains the number of cases, N ($1 \le N \le 20$). N test cases follow.

Each case starts with the number S ($2 \le S \le 100$) – the number of search engines. The next S lines each contain the name of a search engine. Each search engine name is no more than one hundred characters long and contains only uppercase letters, lowercase letters, spaces, and numbers. There will not be two search engines with the same name.

The following line contains a number Q ($0 \le Q \le 1000$) – the number of incoming queries. The next Q lines will each contain a query. Each query will be the name of a search engine in the case.

Output

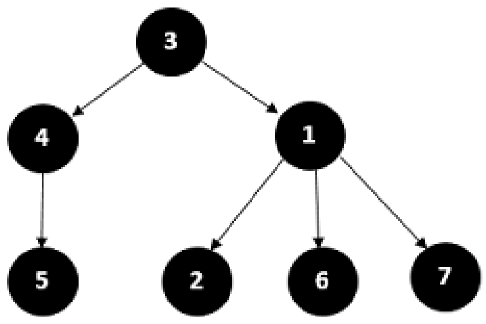
For each input case, you should output Y, where Y is the number of search engine switches. Do not count the initial choice of a search engine as a switch.

Examples

stdin	stdout
2 5 Yeehaw NSM Dont Ask B9 Googol 10 Yeehaw Yeehaw Googol B9 Googol NSM B9 NSM Dont Ask Googol 5 Yeehaw NSM Dont Ask B9 Googol 7 Googol Dont Ask NSM NSM Yeehaw Yeehaw Googol	1 0

Problem K. Who is who’s boss?

well-known company G has n employees. For the sake of simplicity let’s assume that they are numbered from 1 to n. The company has a tree-like structure: each employee, except the president, has exactly one direct boss. One possible structure of a company with 7 employees ($n = 7$) is shown in the picture.



Here, employee 3 is the president and employees 1 and 4 are her direct subordinates. Employee 1 is a boss for employees

2, 6, and 7. Employee 4 has only one subordinate — employee 5.

The structure of the company is kept a secret. However, by carefully asking some of the employees and studying public sources you have found out the following. For each pair of employees (x, y) you know their First Common Boss (letвЂ™s denote this person as $FCB(x, y)$). The first common boss for employee x and employee y is defined as the lowest employee in the hierarchy who is a boss both for employee x and for employee y. As a special case, if employee x is in fact a boss of employee y, $FCB(x, y)$ is equal to x. Similarly, if y is a boss of x, then $FCB(x, y) = y$. For the uniformity, letвЂ™s assume that $F(x, x)$ is equal to x.

In the example above, $FCB(2, 7) = 1$, $FCB(5, 6) = 3$, $FCB(7, 1) = 1$, $FCB(1, 4) = 3$, $FCB(5, 5) = 5$.

It turns out that from this information you can restore the structure of the tree! Write a program that, given FCB for each pair of employees, finds the direct boss for each employee.

Input

The first line contains an integer number n — the number of employees ($1 \le n \le 150$). Each of the next n lines contains n space-separated values. These values give the FCB table: the j-th number in the i-th line is $FCB(i, j)$.

Output

For each of the employees (in the order from 1 to n), print the number of his or her direct boss. For the president of the company print 0 as the number of a boss.

Examples

stdin	stdout

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

`7`
`1 1 3 3 3 1 1`
`1 2 3 3 3 1 1`
`3 3 3 3 3 3 3`
`3 3 3 4 4 3 3`
`3 3 3 4 5 3 3`
`1 1 3 3 3 6 1`
`1 1 3 3 3 1 7`
	`3 1 0 3 4 1 1`