Problem A. Cleaning Robot *(Division 2)*

Input file:	`cleaning-robot.in`
Output file:	`cleaning-robot.out`
Time limit:	2 seconds (*3 seconds for Java*)
Memory limit:	256 mebibytes

In this problem, you have to output a program for the cleaning robot.

A cleaning robot controlled by a program walks along a square grid. Each cell of the grid is either free or occupied by a wall. The robot can occupy any free cell. The position of the robot on the grid is characterized by the coordinates of the cell containing the robot and the direction the robot is facing: it may be one of the four directions parallel to grid lines.

The robot is able to execute the following commands:

- "`f`" (forward) — try to move to the neighboring cell in the current direction,
- "`l`" (left) — turn 90 degrees to the left,
- "`r`" (right) — turn 90 degrees to the right.

If the robot tries to move into a cell occupied by a wall, the command is considered executed, but the position of the robot does not change.

Unfortunately, the cleaning robot does not have any sensors to perceive the world around it. To compensate for that, the robot has a memory module which contains recent history: the last ten executed commands. The commands in the history are ordered chronologically: the first one is the oldest, and the last one is the command which was just executed. After execution of each command, the history is updated according to it: the oldest stored command disappears, the next nine shift by one position, and the newly executed command becomes the last command in the history. When the robot is turned on, the memory can contain any possible history.

The robot is fully automated: after it is turned on, it starts executing the program which must be prepared beforehand. A program for the cleaning robot consists of 3^{10} commands which define the next command for each possible sequence of the last ten executed commands. The program is defined as a sequence of 3^{10} lowercase English letters corresponding to the commands. The first letter is the command executed if the history is "`ffffffffff`", the second one is the command to execute when the history is "`fffffffffl`", the third one is the next command for the history "`fffffffffr`", then goes the next command for the history "`ffffffflf`" and so on in lexicographical order of strings corresponding to the history. The last letter is the command which is executed when the history contains the sequence of commands "`rrrrrrrrrr`".

The cleaning robot has to learn how to clean a room. In order to clean a room, the robot must visit each cell of the room at least once. The first task for the robot is to learn how to clean an empty room of size $w \times h$ cells for different values of w and h. An empty room is a rectangle consisting of free cells which is surrounded by walls from all sides. The task is compilcated by the fact that the robot must solve it with the same program regardless of the robot's initial position in the room and the commands which appear in its memory after it is turned on.

Given w and h, write a program for the cleaning robot which will help it visit each cell of the room at least once. Initially, only one cell is considered visited: the cell where the robot starts executing the program.

Input

The first line of input contains two integers w and h: the dimensions of the room ($1 \le w, h \le 20$).

Output

Print a string without spaces consisting of 3^{10} lowercase English letters from the set of "`f`", "`l`" and "`r`": the program for the cleaning robot.

Example

cleaning-robot.in	cleaning-robot.out
2 2	rrf *(repeated 19683 times without spaces)*

Explanation

The program is formed so that the next command depends only on the last command in the history. If the previous command is "`f`" or "`l`", the next command will be "`r`". If the previous command is "`r`", the next command will be "`f`". So the robot executes either the program "`rfrfrf...`" or the program "`frfrfr...`" depending on the initial history of commands. One can check that, starting from an arbitrary position in a 2×2 room, the cleaning robot will visit all four cells of the room soon enough.

Problem B. Tree Coloring *(Division 2)*

Input file:	`coloring.in`
Output file:	`coloring.out`
Time limit:	2 seconds (*3 seconds for Java*)
Memory limit:	256 mebibytes

Consider a rooted tree with n vertices. The root of the tree has exactly three children. Each vertex except the root and the leaves has exactly two children.

Each leaf is colored in one of three colors. Find a way to color all the other vertices of the tree using the same three colors so that adjacent vertices are colored differently, or determine that there is no such coloring.

Input

The input consists of one or more test cases. Each test case is given in the following format.

The first line of a test case contains an integer n ($4 \le n \le 300\,000$). The next $n - 1$ lines describe the tree. Each i-th of these lines contains an integer p ($1 \le p \le i$), the number of parent of the vertex number $i + 1$. The root of the tree has number 1, other vertices are numbered from 2 to n.

The next line contains an integer m, the number of leaves in the tree. The next m lines specify colors of tree leaves, one per line. The line corresponding to each leaf consists of its number (an integer from 2 to n) and its color (an integer from 1 to 3).

It is guaranteed that the sum of all n in the input also does not exceed 300 000. There is a blank line after each test case. The input is terminated by a line containing a zero instead of n.

Output

For each test case, print a line containing "`YES`" if the coloring exists or "`NO`" otherwise. If the coloring exists, print n integers on the next line. On this line, i-th number must be the color of the vertex number i. If there are several possible colorings, print any one of them.

Examples

coloring.in	coloring.out
4 1 1 1 3 2 3 3 1 4 2 0	NO

coloring.in	coloring.out
6 1 1 1 2 2 4 5 2 6 3 3 1 4 1 0	YES 2 1 1 1 2 3

Problem C. Where Do Identity Permutations Come From *(Division 2)*

Input file:	`iota.in`
Output file:	`iota.out`
Time limit:	2 seconds (*3 seconds for Java*)
Memory limit:	256 mebibytes

In this problem, you have to find out how to make an identity permutation from a given one using the minimum number of certain transformations.

Dwarves Ernest and Leonid work in the standard library of an esoteric programming language, EXTRACALC. Their job is to make the library function `iota` work. Given a positive integer n, this function produces the sequence 1, 2, ..., n: in other words, the identity permutation of n elements.

Each time the `iota` function is called with parameter n, the dwarves take some permutation of length n from an old junk heap, transform it into an identity permutation and present the result to the caller.

Firstly, the permutation gets to Ernest. The transformation available to Ernest is to swap two adjacent elements of the permutation. This transformation can be applied an arbitrary number of times to arbitrary pairs of adjacent elements. Each application of the transformation costs one unit of energy to the library.

After Ernest processes the permutation, it gets to Leonid. The transformation available to Leonid is to replace an element by its index. In order to obtain an identity permutation, this transformation should be applied once for each element which is not yet equal to its index. Each application of this transformation also costs one unit of energy to the library.

Ernest and Leonid want to spend as little energy in total as possible. Given a permutation, find out what transformations and in what order should Ernest apply so that he and Leonid obtain an identity permutation, and the library spends the minimum possible amount of energy as a result.

Input

The input consists of one or more test cases. Each test case is given on a single line which starts with an integer n followed by a permutation: a sequence of n integers in which each integer from 1 to n occurs exactly once. Adjacent integers on a line are separated by one or more spaces. For each given permutation, the double inequality $1 \le n \le 500$ holds. Additionally, the sum of all n in the input also does not exceed 500. The input is terminated by a line containing 0 instead of n.

Output

For each test case in the input, print one line. This line must start with one integer: the minimum possible amount of energy spent by the library in order to transform the given permutation into an identity permutation. After that, print the number of transformations applied by Ernest, followed by the transformations themselves in the order of their application. Each transformation is described by an integer k ($1 \le k < n$) which means that Ernest swaps the elements of the permutation at indices k and $k + 1$. Separate the consecutive integers on a line with one or more spaces. If there is more than one optimal answer, print any one of them.

Example

iota.in	iota.out	Notes
3 2 1 3 3 2 3 1 5 5 2 3 4 1 4 4 3 2 1 0	1 1 1 2 2 2 1 2 0 3 1 2	$\underline{2\ 1}\ 3 \to 1\ 2\ 3$ $2\ \underline{3\ 1} \to \underline{2\ 1}\ 3 \to 1\ 2\ 3$ $\overline{5}\ 2\ 3\ 4\ \overline{1}$ $4\ \underline{3\ 2}\ 1 \to \overline{4}\ 2\ 3\ \overline{1}$

Explanation

The transformations of the given permutations are shown at the right. The underlined elements are the

ones swapped by Ernest. The elements with a line above them are the ones replaced by Leonid when the permutation gets to him from Ernest.

Problem D. Secret Community Card Game *(Division 2)*

Input file:	`lots.in`
Output file:	`lots.out`
Time limit:	1 second (*2 seconds for Java*)
Memory limit:	256 mebibytes

In a certain community, the following game is fairly popular. Each of the n participants has a card which is a stripe of $1 \times m$ cells. Each cell is either empty, filled, or undecided. Before the start of the game, each undecided cell must be either filled or left empty by the participant. After that, all cards are collected and laid on the table one under another forming an $n \times m$ rectangular table of cells. The cards must be ordered from top to bottom by the number of filled cells: cards with higher number of filled cells must be placed higher. The cards which have the same number of filled cells can be arranged arbitrarily.

The participant whose card is placed exactly in the middle of the table is declared the winner. Thus, following the tradition, the number of participants is always odd so that there is no need to divide the prize.

You are a renowned member of the community and an unbeatable player, so it is your long-time honorable duty to lay the cards on the table in the final stage of the game.

Each year, hundreds of your disciples ask you to kindly suggest them what to do with the undecided cells on their cards. This year, every single one of them asked you for help. Now you have decided to teach the youth a lesson: tell them to fill the cells in such a way that you will be able to display the emblem of the community on the table using their cards, following the rules of the game.

All that remains is to find out how to do it.

Input

The input consists of one or more test cases. The first line of each test case contains two integers n and m ($1 \le n, m \le 50$, n is odd). Then follow n lines describing the cards of the participants. Each such line contains m characters. Each of these characters can be one of the following:

- "`x`" — filled cell,
- "`.`" — empty cell,
- "`?`" — undecided cell for which the participant asks your advice.

There is a blank line after the card descriptions.

The next n lines, each containing m characters, describe the emblem of the community which you wish to display on the table. The emblem consists of characters "`x`" and "`.`".

The sum of all values of $n \cdot m$ in the input does not exceed 30 000. The input is terminated by a line containing two zeroes instead of n and m.

Output

For each test case, print a line containing either «`YES`» or «`NO`» depending on whether the goal can be achieved. If the answer is positive, print n more lines each containing m characters: the decided versions of the participants' cards in the order they were given in the input. If there is more than one possible solution, print any one of them.

Examples

lots.in	lots.out
3 2 x? ?? x. xx x. x. 3 3 xx? .x. ... x.x .x. ... 0 0	YES x. xx x. NO
7 11 ....??xx.x?... ..?..?..... ....xxx.??? .xx?....xx. x?xx.??xx?x ..?x...?x.? xxxx...xxxx .xx.....xx. ..xx...xx.. ...xx.xx... ....xxx.... .....x..... .... 0 0	YES ....xx.xx... .....x..... ....xxx.... .xx.....xx. xxxx...xxxx ..xx...xx..

Problem E. Partition *(Division 2)*

Input file:	`partition.in`
Output file:	`partition.out`
Time limit:	0.5 seconds (*1 second for Java*)
Memory limit:	256 mebibytes

Consider a plane divided into square cells. If we choose a set of cells on the plane, it corresponds to the following graph: the cells are the vertices of the graph, and there is an edge between two cells if they share a common side.

There is a set of n cells selected on the plane. It is guaranteed that the graph which corresponds to this set of cells is connected. Find a way to remove no more than $\lceil \frac{3}{2}\sqrt{n} \rceil$ cells from the set so that in the graph corresponding to the remaining cells of the set, each connected component consists of no more than $\lceil \frac{n}{2} \rceil$ cells.

Input

The first line of input contains one integer n, the number of cells in the set ($1 \le n \le 10\,000$).

Each of the next n lines contains two integers x and y, the coordinates of a cell ($-10^9 \le x, y \le 10^9$). It is guaranteed that all n given cells are distinct.

Output

The first line of output must contain one integer m, the number of cells which you want to remove from the set. Keep in mind that it is *not necessary* to minimize this number, however it must not exceed $\lceil \frac{3}{2}\sqrt{n} \rceil$.

The next m lines must contain coordinates of cells which you waht to remove in the same format as in the input. All printed cells must be distinct and must be present in the input. You can print the cells in any order. If there are several possible answers, print any one of them.

Examples

partition.in	partition.out
16 1 1 1 2 1 3 1 4 2 1 2 2 2 3 2 4 3 1 3 2 3 3 3 4 4 1 4 2 4 3 4 4	4 1 1 2 2 3 3 4 4

partition.in	partition.out
14 1 1 1 2 1 7 1 8 2 1 2 2 2 3 2 4 2 5 2 6 2 7 2 8 3 4 3 5	2 2 3 2 6

Problem F. Matrix Consistency *(Division 2)*

Input file:	`prob.in`
Output file:	`prob.out`
Time limit:	1 second (*2 seconds for Java*)
Memory limit:	256 mebibytes

Consider a matrix A of size $n \times n$ consisting of zeroes and ones. A random row vector x and a random column vector y are chosen: both of them have length n and consist of zeroes and ones, each element of each vector takes value 0 or 1 with probability $\frac{1}{2}$ independently from the others. Calculate the probability of the event that $xAy = 1$. Addition and multiplication are performed modulo 2.

Recall that the product PQ of a matrix P of size $u \times v$ and a matrix Q of size $v \times w$ is a matrix R of size $u \times w$ such that for all $1 \le i \le u$ and $1 \le j \le w$, its element $R_{i,j}$ is defined as follows:

$$R_{i,k} = \sum_{j=1}^{v} P_{i,j} \cdot Q_{j,k}.$$

Remember that in this problem, the sum of products in the above formula is calculated modulo 2. The row vector x is a matrix of size $1 \times n$, the column vector y is a matrix of size $n \times 1$.

Input

The input consists of one or more test cases. The description of each test case starts with an integer n on a separate line ($2 \le n \le 1000$). The next n lines of input contain the rows of the matrix, encoded for brevity in a special way. Firstly, a row is extended by zeroes *to the right* if needed until its length divides evenly by six. After that, the row is split into $\lceil \frac{n}{6} \rceil$ groups with exactly six digits in each group. A group $a_0 a_1 \ldots a_5$ is then encoded by the number $48 + \sum_{i=0}^{5} a_i \cdot 2^i$. In the input, this number is represented by a character with the corresponding ASCII code.

The sum of n in all test cases does not exceed 1000.

Output

For each test case, print one real number on a separate line: the probability of the event that $xAy = 1$. An absolute error of no more than 10^{-9} is allowed.

Example

prob.in	prob.out
2 1 2 2 3 3	0.375 0.25

Explanation

The matrices encoded in the example are $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ and $\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$.

Problem G. MST of Random Points *(Division 2)*

Input file:	`randommst.in`
Output file:	`randommst.out`
Time limit:	3 seconds (*9 seconds for Java*)
Memory limit:	256 mebibytes

Consider n distinct points on a plane. The coordinates of points are integers from 0 to 30 000 inclusive. The points are chosen *at random* in the following sense: consider all possible collections of n distinct points on a plane with the given restrictions on coordinates, and then pick one of such collections at random with uniform probability.

You can draw a straight line segment between any two given points. The length of a segment between points with coordinates (x_1, y_1) and (x_2, y_2) is equal to $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. We will say that points a and b are *connected* if a segment is drawn between them or there is a point d which is connected to both a and b. Your task is to draw some segments of minimum total length so that all points are connected.

Input

The first line of input contains an integer n ($2 \le n \le 50\,000$). The next n lines contain the coordinates of the points. It is guaranteed that all given points are distinct. Additionally, it is guaranteed that in each test except the example, the points are selected at random as specified in the problem statement.

Output

On the first line, print a real number w, the total length of the segments you selected to draw. On the next $(n - 1)$ lines, print the segments, one per line. Each segment must be printed as two integers from 1 to n which specify the numbers of points connected by that segment.

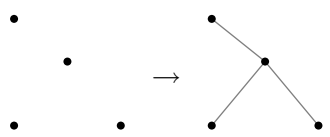
Let the real total length of the printed segments be w^* and the total length of the segments in the optimal answer be w_{opt}. Your answer will be considered correct if

$$\max\left(\left|\frac{w}{w^*} - 1\right|, \left|\frac{w^*}{w_{\text{opt}}} - 1\right|\right) < 10^{-12}.$$

Example

randommst.in	randommst.out
4 0 10 5 6 10 0 0 0	22.02362358924615 1 2 2 3 4 2

Illustration



Problem H. Rolling a Die *(Division 2)*

Input file:	`rolling-dice.in`
Output file:	`rolling-dice.out`
Time limit:	2 seconds (*3 seconds for Java*)
Memory limit:	256 mebibytes

In this problem, you have to find the sum of numbers imprinted on the plane by a cube as it is rolling along a segment.

Consider a plane with Cartesian coordinate system introduced on it. The plane is divided into squares by straight lines of the form $x = c$ and $y = c$ for integers c.

Let us pick two relatively prime positive integers a and b. Consider the straight line segment between points $(0, 0)$ and (a, b) on our plane. Mark the squares which have more than one common point with the segment. Due to relative primality of a and b, these squares form a path from the square with lower left corner $(0, 0)$ to the square with upper right corner (a, b) such that adjacent squares in the path share a side.

Take a die: a unit-sided cube which has different integers from 1 to 6 associated with its different faces. Put the cube on the first square of the path and roll it along the path until it reaches the last square by turning it over the sides. Each time the cube appears on a square, the number on the bottom face of the cube is imprinted on that square.

Given a, b and a description of the cube in its initial position, find the sum of numbers imprinted on the plane.

Input

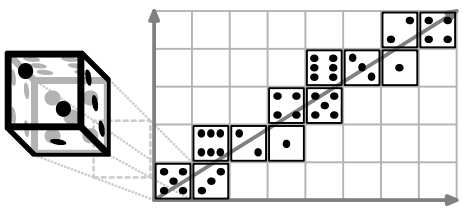
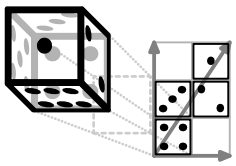
The first line of input contains two integers a and b ($1 \le a, b \le 10\,000$). It is guaranteed that the numbers a and b are relatively prime.

The second line contains six integers: the numbers on the faces of the cube. It is guaranteed that each integer from 1 to 6 appears on exactly one face of the cube. No other properties of the distribution of numbers are guaranteed. The faces are listed in the following order: front, upper, right, left, lower, back.

In the initial position, the cube lies on its bottom face. If we move the cube from its initial position to the adjacent square in positive direction of Ox axis, the cube will land on its right face. If we move the cube from its initial position to the adjacent square in positive direction of Oy axis, the cube will land on its back face.

Output

Print one integer: the sum of numbers imprinted on the plane as the cube is rolling along the given segment.

rolling-dice.in	rolling-dice.out	Notes
8 5 1 2 3 4 5 6	42	
2 3 6 1 2 5 4 3	10	

Explanations

The pictures above correspond to the examples. On the left is the cube in its starting position. On the right is the trace of the cube as it is rolling along the given segment.

Problem I. Statistics *(Division 2)*

Input file:	`stat.in`
Output file:	`stat.out`
Time limit:	1 second
Memory limit:	256 mebibytes

Vasiliy Petrovich plays an infinite game which consists of an infinite number of rounds. In this game, there are n different objects, numbered by n sequential integers, starting from one. During one round, Vasiliy Petrovich visits each of the objects and touches it with probability p.

Consider the first round after which the first object was touched exactly k times. At the moment after that round, what is the expected number of objects which were touched at least once?

Recall that the expected number of object which were touched is the sum

$$\sum_{t=1}^{n} P_t \cdot t,$$

where P_t is the probability of the event that exactly t objects were touched.

Input

The first line contains three numbers n, k and p. Here, n and k are integers from 1 to 10^9 inclusive, and p is a real number from 0.1 to 0.9 inclusive, given with no more than five digits after the decimal point.

Output

Print one real number: the expected number of objects that were touched at the moment after the first such round after which the first object (object, numbered by 1) was touched exactly k times. The absolute or relative error of your answer must not exceed 10^{-12}.

Examples

stat.in	stat.out
10 1 0.5	7.0
10 2 0.1	7.980609418282548