Problem A. Base $i-1$ Notation

Input file:	base-i-1.in
Output file:	base-i-1.out
Time limit:	2 seconds
Memory limit:	256 mebibytes

Gaussian integer is a complex number of the form $a+b\cdot i$, where a and b are integers, and i is the square root of -1.

Let us say that a gaussian integer $a+b\cdot i$ is written in base $i-1$ notation in the form $\overline{c_n c_{n-1}\ldots c_1 c_0}_{i-1}$ if $a+b\cdot i=\sum\limits_{k=0}^{n} c_k\cdot(i-1)^k$, and all $c_k\in\{0,1\}$. Additionally, if $n\neq 0$, then $c_n\neq 0$. An amazing fact is that, for any Gaussian integer, this form exists and is unique.

For example, $1=1_{i-1}$, $-1=11101_{i-1}$, $2i+1=1111_{i-1}$.

You are given two gaussian integers written in base $i-1$ notation. Calculate their sum and print it in base $i-1$ notation.

Input

The first line contains the number a, the second one contains the number b. Both of them are written in base $i-1$ notation, and each consists of no more than 500 000 digits.

Output

Print the sum of the two given numbers, also written in base $i-1$ notation.

Examples

base-i-1.in	base-i-1.out	Notes
101 10	111	a = 1-2*i b = -1+1*i a+b = 0-1*i
10100100100 110101	11111001100001	a = 20-38*i b = 1-6*i a+b = 21-44*i

Problem B. Squaring a Bit

Input file:	`bit-squares.in`
Output file:	`bit-squares.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

You are given an integer n. Consider its binary representation without leading zeroes (for example, $10_{10} = 1010_2$). In how many ways can you rearrange its binary digits so that the resulting number is a perfect square? For example, for $10_{10} = 1010_2$, there is only one way: $1010_2 \to 1001_2 = 9 = 3^2$. It is not allowed to rearrange the bits so that the result contains leading zeroes and that the result exceeds 10^{18}.

Input

The only line of the input contains a single integer n ($1 \le n \le 10^{18}$).

Output

Print an integer: the number of ways to rearrange bits of n in such way that the result is a perfect square.

Examples

bit-squares.in	bit-squares.out
9	1
10	1
2	0
1000000000000000000	12206114

Problem C. Chickens

Input file:	`chickens.in`
Output file:	`chickens.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

Where do baby chickens come from? Probably you have already asked yourself this question. Maybe you have read the answer in a book. But, as the saying goes, one look is worth a thousand words.

In this problem, we are concerned with a specific kind of a matryoshka doll: the broody hen doll. This toy consists of several parts:

- The outer doll is the hen itself.
- n eggs, indexed from 1 to n. Each egg can be opened so one can put something inside.
- n baby chickens, indexed from 1 to n. Ain't no one disassembles a baby chicken. No one.

The doll is considered assembled if each baby chicken is placed inside an egg, and each egg is placed inside the hen. It is not allowed to place an egg inside another egg, place several baby chickens in the same egg, or not to put a baby chicken in any egg at all.

Each baby chicken has a size c_i ($1 \le i \le n$). Similarly, each egg has a size as well e_j ($1 \le j \le n$). One can place a baby chicken i inside an egg j if and only if $c_i \le e_j$.

Your task is to calculate the number of ways to assemble the doll, that is, the number of ways to place each baby chicken inside an egg. Two ways are considered different if there is a baby chicken which is placed in eggs with different numbers in these ways.

Input

The first line of the input contains a single integer n, the number of baby chickens and eggs ($1 \le n \le 12$). The second line contains n integers $c_1, c_2, \ldots, c_n$ separated by single spaces: the sizes of the baby chickens ($1 \le c_i \le 100$). The third line contains n integers $e_1, e_2, \ldots, e_n$ separated by single spaces: the sizes of the eggs ($1 \le e_i \le 100$).

Output

Print a single integer on the only line of the output: the number of ways to put baby chickens into eggs.

Examples

chickens.in	chickens.out
3 1 2 3 3 3 1	2
4 1 1 1 1 100 100 100 100	24
4 100 100 100 100 1 1 1 1	0
10 1 2 3 4 5 6 7 8 9 10 2 3 4 5 6 7 8 9 10 11	512

Problem D. Lights at a Crossing

Input file:	`crossing-lights.in`
Output file:	`crossing-lights.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

Petya the Robot sits in his company's new office and remotely drives several robo-taxis. From time to time, when all taxis stop, he glances through the window at the street crossing by his office. There are n traffic lights at the crossing visible to Petya. The traffic lights are somewhat special. Each can display only two colors: red and green. When the signal is red, the traffic light additionally shows a positive integer: the number of seconds before the signal turns green. When the signal is green, no numbers are shown. The readings on all traffic lights change simultaneously once a second.

Petya the Robot knows that all traffic lights at the crossing are parts of one system. The system has a period of T seconds: if one looks at the lights twice with the time gap of exactly T seconds, all readings are the same. During this period, the signal of each traffic light changes exactly twice: the light i is green for X_i consecutive seconds and then red for $Y_i = T - X_i$ consecutive seconds. Additionally, each second, there is at least one light which is red. Still, Petya does not know the exact value of T, as well as the values X_i and Y_i for each traffic light, he only knows the system's basic functioning principles described above.

Petya the Robot wants to know the period T. Robots generally have good memory, and Petya remembers the results of m observations: each time he glanced through the window, he memorized the readings on all n visible traffic lights. Help him find the exact value of T, or determine that it is impossible with the given information.

Input

The first line of input contains integers n and m: the number of traffic lights at the crossing and the number of observations ($2 \le n \le 20$, $2 \le m \le 250$). The next m lines describe the observations. Each of them contains n simultaneous readings of the traffic lights: the first, the second, ..., the last one. If a traffic light is green, it is denoted as "`X`" (English capital letter "ex"). If a light is red, the readings are expressed as a strictly positive integer: the number of seconds before the light turns green. Consecutive readings on a line are separated by at least one space, and additional spaces may appear before each reading to improve readability.

It is guaranteed that the traffic lights at a crossing are behaving as described in the statement. As for the numbers T, X_i, and Y_i, Petya knows for sure that they are strictly positive integers. Additionally, Petya knows that there is a lower limit on X_i and Y_i which is around 10 seconds, and there is an upper limit on T which is around 200 seconds, but he does not know the exact bounds: they may be actually a few seconds more or less than stated.

Output

If the given information allows to determine the exact value of T, print this number. Otherwise, print the number -1.

Examples

crossing-lights.in	crossing-lights.out
4 5 X 33 X 36 X 4 X 7 42 2 42 5 X 21 X 24 8 X 8 54	83
2 2 X 100 100 X	-1

Explanations

In the first example, Petya the Robot sees four traffic lights. The actual parameters of the system are the following. The period is 83 seconds. The first and the third traffic lights work identically, they turn green for 40 seconds and red for 43 seconds. The second light turns green 43 seconds after the first one turns green and stays green for 36 seconds. The fourth light turns green 46 seconds after the first one and stays green for 29 seconds.

The observations are made 10, 39, 41, 22, and 75 seconds after the moment when the first traffic light turns green. It turns out that these observations allow to uniquely determine the period.

In the second example, there are only two lights and two observations. The observations show that the period is at least 200 seconds. Still, it can actually be any integer greater than 200, and Petya the Robot does not know the exact upper bound on the period. So, the number T can not be uniquely determined.

Problem E. Decimal Form

Input file:	`decimal-form.in`
Output file:	`decimal-form.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

Consider coprime integers a and b ($0 \le a < b \le 10^9$). You are given $\frac{a}{b}$ in the form of decimal fraction, rounded up to precisely 18 digits after the decimal point. In this problem, numbers are always rounded *to the nearest* decimal fractions, and in case of a tie, the number is rounded *up*. Find the integers a and b.

Input

The first line contains an integer n, the number of test cases ($1 \le n \le 10^4$). The next n lines describe test cases. Each of them contains a decimal fraction with exactly 18 digits after the decimal point. It is guaranteed that each decimal fraction was obtained in the way described above.

Output

Output n lines. The i-th line must contain integers a and b ($0 \le a < b \le 10^9$) used to generate the i-th test case.

Example

decimal-form.in	decimal-form.out
2 0.000000000000000000 0.666666666666666667	0 1 2 3

Explanation

In the first test case, $\frac{0}{1} = 0$.

In the second test case, $\frac{2}{3} \approx 0.666\,666\,666\,666\,666\,667$.

Problem F. Martian Maze

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	256 mebibytes

This is an interactive problem.

A research robot on Mars is going to another maze. The maze is a flat rectangular field aligned with the cardinal directions and divided into $h \times w$ squares. Between each pair of squares which share a side, there is either a passage or a wall. A solid wall surrounds the field.

The robot starts at the southwestern corner square of the maze. The robot's goal is to arrive at the northeastern corner square of the maze. Each second, we can issue a command to the robot, so that it moves to the neighboring square in one of the four cardinal directions: "`N`" (north), "`W`" (west), "`S`" (south), or "`E`" (east). Next, the robot either carries out the command if there is a passage in the respective direction, or stays in place if a wall prevents the movement. After that, the robot will send a response: "`yes`" if the move was successful, or "`no`" in the other case.

The communication is carried out via a satellite network and, depending on the positions of Phobos and Deimos, the delay between issuing a command and getting the response may be significant. In the near future, the delay is constant and equal to d seconds. This means that the response to the command number x can be received right after issuing the command number $x + d$. However, a sand storm will start shortly: after $t = 10\,000$ seconds, sending commands will be no longer possible, only the responses to previous commands can be received at that time.

Take control over the robot and help it achieve the goal before the storm makes it impossible.

Interaction Protocol

Firstly, your program is given three integers, h, w, and d, on a separate line: the number of squares from north to south, the number of squares from west to east, and the delay in seconds ($2 \le h, w \le 20$, $0 \le d \le 100$). In each test, the maze is fixed in advance but kept secret.

In the robot's world, the following events happen each second.

1. If the number of already printed commands is less than t, your program must print the next command, "`N`", "`W`", "`S`", or "`E`", on a separate line. You can not skip a command and can not order to stand in place.

   To prevent output buffering, flush the output buffer after each request: this can be done by using, for example, `fflush (stdout)` in C or C++, `System.out.flush ()` in Java, `flush (output)` in Pascal or `sys.stdout.flush ()` in Python.

2. If the number of already printed commands is more than d, your program must read the response to the command issued d seconds ago. Normally, the response will be "`yes`" or "`no`" which states whether the corresponding move was successful. However, if after the move, the robot achieved the goal, instead of "`yes`", the response will be "`success`": in this case, all remaining commands will be ignored, and the program must immediately terminate gracefully. Finally, if after the last possible command, the robot did not achieve the goal, the response will be replaced by "`timeout`", and the solution will get "`Wrong Answer`".

It is guaranteed that the maze in each test is constructed as follows. First we put a solid wall around the field. After that, we consider all possible walls between pairs of neighboring squares of the maze: $h \cdot (w-1)$ walls going from west to east and $(h-1) \cdot w$ walls going from north to south. We then fix a random order of these possible walls and consider them in this order. Each possible wall will be put into the maze if, when the wall is present, each square of the maze is still reachable from all other squares. One can prove that the result will be such a maze that there is a unique path between each pair of squares which does not visit any square twice.

Examples

standard input	standard output	Maze
2 3 0 no yes no yes success	 W E E N E	+-+-+-+ | | | + + + + | | | +-+-+-+
4 3 2 yes yes yes yes success	 N N E N E W W	+-+-+-+ | | | + + +-+ | | | + + + + | | | | + +-+ + | | +-+-+-+

Problem G. Wet Mole

Input file:	`mole.in`
Output file:	`mole.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

Young mole named Friedrich just relocated to the comfortable hole inside Flower Hill. He barely unpacked his belongings, and was about to hang the portrait of his molaunt on the wall, when the sky suddenly filled with dark clouds. Rain will start any second! It will be unfortunate if the portrait of aunt will get wet, so it is necessary to find a place in the hole where rainwater won't get to.

As you may know, moleholes of Flower Hill are digged up in vertical rectangle shapes: every hole is h moleters (mole unit of length) deep and w moleters wide. There is no third dimension in economy-class moleholes, so that neighbouring moleholes don't intersect with each other. Every 1×1 square moleter is either empty of filled with dirt.

When it is raining, water drops from the upper side and fills every empty square in top level of molehole, then flows down and sideways. Formally: if a square has water and a square one moleter below is empty, the water will also fill this empty square. And in case the square below is filled with dirt, but the cells one moleter to the left or to the right are empty (one or both), the water will fill these empty squares, too.

You are given Friedrich's molehole map: an $h \times w$ rectangle of dirt or empty squares. You have to find a square wich will not be filled with water, or complain that there is no such square. Of course, it is only possible to hang the portrait in an empty square, not dig it into the dirt.

Input

The first line of input contains two positive integers: h and w. Each of the next h lines contains w characters: the molehole map from top to bottom. Each character is either "`.`" (empty) or "`#`" (dirt). Molehole is no more than 500 moleters in each dimension. It is guaranteed that the bottom level of the molehole has no empty squares.

Output

On the first line, print "`Yes`" in case you found a suitable square, or "`No`" otherwise. If you found a square, additionaly output h lines, each containing w characters: the map of molehole in the same format as in input, but place character "`X`" in the square suitable for the portrait. If there are multiple suitable squares, mark any one of them.

Examples

mole.in	mole.out
4 6 #.##.# #.#.## #...## ######	Yes #.##.# #.#X## #...## ######
5 6 #.#.## #.#.## #....# ###.## ######	Yes #.#.## #.#.## #...X# ###.## ######
5 5 ..##. .##.. .#..# ...## #####	No
20 30 .....#..############....##.... .###...###########...#.##...## ..###..###########.###....#### #..#.....#########...###....## ##.#.###..#...######..##.##..# ##.#....#####..######.##.###.. ##..###...####........##.####. ###.#####....#####.#####...##. ###......###........######.... ########.########.#........### ########..#######..#..######## #########.########.##....##### #######...########...###..#### #######.############.####..... #######...##########..#######. #########..##########........# ##########.##################.. ##########...################. ############.################. ##############################	Yes .....#..############....##.... .###...###########...#.##...## ..###..###########.###....#### #..#.....#########...###....## ##.#.###..#.X.######..##.##..# ##.#....#####..######.##.###.. ##..###...####........##.####. ###.#####....#####.#####...##. ###......###........######.... ########.########.#........### ########..#######..#..######## #########.########.##....##### #######...########...###..#### #######.############.####..... #######...##########..#######. #########..##########........# ##########.##################.. ##########...################. ############.################. ##############################

Problem H. Oddities

Input file:	`oddities.in`
Output file:	`oddities.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

Jack is arranging a party. He owns the best nightclub in town with n rooms and m bidirectional corridors running between them (a corridor connects two distinct rooms). The nightclub is connected: there is a path via corridors between any two rooms. We will say that G is the corresponding connected graph with vertices as rooms and corridors as edges.

Jack wants the party to be awesome and odd at the same time, so he decides to destroy some of the corridors between rooms. He wants to do that in such a way that, for each i, the oddity (the value modulo 2) of the number of remaining corridors adjacent to room i is equal to a fixed number $a_i \in \{0, 1\}$. You may notice that the word "parity" mathematically means the same thing as "oddity"; the latter just sounds better for the party. So, Jack asked his friend John, who is a computer science student, to find a way to destroy the corridors.

John worked hard for two weeks without any success. He found out that $\left(\sum_{i=1}^{n} a_i\right) \bmod 2 = 1$, and he proved that this implies that there is no way to satisfy Jack's requirements. Jack was very upset when he heard that. He said that John's proof was garbage and that John better found a *hard proof* in five hours. John was wise enough and didn't ask Jack what happens in five hours, so he went straight to you and begged for help.

First of all, you should understand what constitutes a *hard proof* in Jack's terms. Jack wants the proof to be formal, and that means it should involve formulas, more than that, logical formulas in CNF (conjunctive normal form). A *formula in CNF* is a conjunction (logical and) of a finite number of clauses. A *clause* is a disjunction (logical or) of a finite number of *literals*. A *literal* is either a variable (like Y) or a variable's negation (like $\neg Y$). Each variable can have a value of either zero (logical "false") or one (logical "true"). For example, a clause may look like $x \vee y \vee \neg z$. Thus a formula in CNF can look like $(x \vee y) \wedge (\neg y \vee z)$.

Let $L(\varphi)$ be the set of all variables which occur in formula φ and their negations. For example, $L((x \vee z) \wedge \neg y) = \{x, y, z, \neg x, \neg y, \neg z\}$. Now, we call a bijection $\pi : L(\varphi) \to L(\varphi)$ an *automorphism* if two conditions hold:

- For each literal x: $\pi(\neg x) = \neg\pi(x)$ (here we assume that $\neg\neg\alpha = \alpha$).
- If we replace all literals in φ with their images in π simultaneously and obtain a formula $\pi(\varphi)$, then $\pi(\varphi)$ is the same as φ up to the order of clauses and literals inside each clause.

For example:

- x has only one automorphism: the identity function.
- $x \vee \neg y$ has two automorphisms: one is the identity function, and the other maps x to $\neg y$, $\neg x$ to y, y to $\neg x$, and $\neg y$ to x. The latter transforms the formula into $\neg y \vee x$, which is equivalent to $x \vee \neg y$.
- $(x \wedge x) \vee y$ has only one automorphism: the identity function.

Jack calls a formula φ *unsatisfiable* if there is no assignment of values $\{0, 1\}$ to variables such that the value of φ after this assignment is 1. For example, the formula $\varphi = x \wedge \neg x$ is unsatisfiable: $\varphi\big|_{x=0} = \varphi\big|_{x=1} = 0$.

A *hard refutation of a formula φ (in CNF) in Jack's terms* is a sequence of clauses $C_1, \ldots, C_k$ such that $C_k = \square$ ($\square$ stands for the empty clause which is always false) and for each $i \in \{1, \ldots, k\}$, one of the following conditions holds:

1. $C_i \in \varphi$, that is, C_i is one of the clauses of the initial formula.
2. C_i is obtained by resolution rule from two clauses C_j and C_k occuring earlier in the sequence (that is, $j, k < i$). They must have a special form: $C_j = A \vee x$ and $C_k = B \vee \neg x$ for some possibly empty clauses A, B and a variable x. If this is the case, then C_i is obtained as $A \vee B$.

   Note that A and B can intersect (that is, have common literals).

   For example, if we have clauses $C_j = (x \vee y \vee t)$ and $C_k = (x \vee \neg y \vee s))$, we can get $C_i = (x \vee t \vee s)$ by applying the resolution rule for variable y.
3. There is a previous clause C_j ($j < i$) and an automorphism π of φ such that $C_i = \pi(C_j)$. For example, if $\varphi = x \vee \neg y$, the only non-identity π which we can use here would rename x to $\neg y$ and so on (see above).

Let us define a formula ODD_G in CNF which will be satisfiable if and only if it is possible to destroy some corridors in the club to satisfy Jack's requirements. It will have m logical variables $e_1, \ldots, e_m$ which take values from the set $\{0, 1\}$: $e_i = 0$ if i-th corridor is destroyed and $e_i = 1$ if it remains. Now let us encode the fact that i-th room has a correct oddity. We denote the number of corridors from the room i as d_i, and variables corresponding to these corridors as $c_{i,1}, \ldots, c_{i,d_i}$ (for eaxmple, $c_{i,1}$ may be e_4 if there is a corridor from room i with number 4). Now, the room has the correct oddity if and only if it does not have the wrong oddity. This fact can be written as a CNF formula with $2^{d_i - 1}$ clauses, each clause containing d_i literals. In the end, we get the following formula encoding the fact that a specific way of destroying some corridors satisfies Jack's requirements:

$$\mathrm{ODD}_G(e_1, \ldots, e_m) = \bigwedge_{i=1}^{n} \bigwedge_{\substack{s \in \{0,1\}^{d_i} \\ (s_1 + \ldots + s_{d_i}) \bmod 2 \neq a_i}} \underbrace{\left(\bigvee_{\substack{1 \le j \le d_i \\ s_j = 0}} c_{i,j} \vee \bigvee_{\substack{1 \le j \le d_i \\ s_j = 1}} \neg c_{i,j} \right)}_{\text{true iff valiables } c_{i,1} \ldots c_{i,d_i} \text{ do not have values } s_1, \ldots, s_{d_i}}$$

Note that this formula is equivalent to

$$\bigwedge_{i=1}^{n} (c_{i,1} \oplus \ldots \oplus c_{i,d_i} \oplus \neg a_i)$$

where $\oplus$ is exclusive or, but this one is not in CNF, so it can not be used directly in proof.

Find a *hard refutation* of ODD_G containing no more than 1000 clauses.

Input

The first line of the input contains two integers: n and m ($3 \le n \le 73$, $0 \le m \le 492$). The second line contains n integers: $a_1, \ldots, a_n$ ($a_i \in \{0, 1\}$). The i-th of the next m lines contains the description of the i-th corridor: two integers $u, v \in \{1, \ldots, n\}$. Such line means that i-th corridor connects rooms u and v.

Two rooms can not be connected by more that one corridor, and a corridor cannot connect a room to itself. It is guaranteed that it is possible to go via corridors between any two rooms.

Output

Whenever we need to describe a literal in a formula, integer t denotes the variable e_t, and integer $-t$ denotes its negation $\neg e_t$. Edges and corresponding variables are numbered from 1 to m in order of input.

Output must contain no more than 1000 lines. The i-th of these lines must describe clause C_i of the proof and be in one of the following forms:

- `declare` q l_1 ... l_q if C_i will be one of the clauses of ODD_g. Here, q must be the number of literals in said clause, and $l_1, \ldots, l_q$ must be its literals. All literals listed must be distinct.
- `resolve` j k t. Corresponds to an application of the resolution rule. Here, j and k are numbers of clauses to use in rule, and t is the variable used in rule. It must hold that $1 \le j, k < i, t \in \{1, \ldots, m\}$, for some clauses A and B clause C_j must be equal to $A \vee e_t$, and $C_k = B \vee \neg e_t$. C_i will be equal to $A \vee B$.
- `map` j a_1 ... a_m. Corresponds to using the authomorpism rule. a_t must be the literals to which e_t is mapped. The mapping $e_t \mapsto a_t$, $\neg e_t \mapsto \neg a_t$ must be a valid automorphism of ODD_G according to the definition in the problem statement. C_i will be equal to $\pi(C_j)$ where π is the mapping decribed above.

Proof will be considered correct if and only if the last clause is empty.

Example

oddities.in	oddities.out	Notes
3 3 1 0 0 1 2 2 3 1 3	declare 2 -2 1 declare 2 2 -3 resolve 2 1 2 map 3 -3 -2 -1 declare 2 -1 -3 resolve 4 5 1 map 6 -1 -2 -3 resolve 7 6 3	$C_1 = \neg e_2 \vee e_1$ $C_2 = e_2 \vee \neg e_3$ $C_3 = e_1 \vee \neg e_3$ $C_4 = \neg e_3 \vee e_1$ $C_5 = \neg e_1 \vee \neg e_3$ $C_6 = \neg e_3$ $C_7 = e_3$ $C_8 = \square$

Problem I. Sorting on the Plane

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	256 mebibytes

This is an interactive problem.

There are n vectors on the plane, they are all non-zero and pairwise non-collinear. The vector number i goes from the origin to the point (x_i, y_i). But we won't tell you these coordinates.

Instead, you can ask questions of the following format: "Is it true that vectors i and j form a right pair?" Formally, vectors form a right pair if $x_i \cdot y_j > x_j \cdot y_i$. Geometrically, a right pair means that, if we stand at the origin and look straight at the endpoint of vector i, then, in order to turn to the endpoint of vector j as soon as possible, we have to turn in counter-clockwise direction.

You have to ask questions and arrive at one of the two possible outcomes:

1. All vectors lie in one semi-plane with its border passing through the origin. Then you have to sort them: output a sequence of distinct indices $i_1, i_2, \ldots, i_n$ such that for every $p < q$, the vectors i_p and i_q form a right pair.
2. There is no semi-plane such that its border passes through the origin and it contains all the given vectors. Then you have to present a proof: output a sequence of distinct indices $i_1, i_2, \ldots, i_k$, where each vector forms a right pair with the next one, except for the last one (i_k) which forms a right pair with the first (i_1).

Interaction Protocol

Firstly, your program is given the number n on a separate line: the number of vectors ($1 \le n \le 500$). In each test, the vectors are fixed in advance but kept secret.

After that, you can perform the following actions:

1. Ask the jury: "Is it true that vectors i and j form a right pair?"

   In order to do that, your program must output a line in the following format: "`? i j`". The indices must be valid: $1 \le i, j \le n$.

   In response, the jury program will give one number on a separate line: 1 if the answer is "yes" and 0 if the answer is "no".

   To prevent output buffering, flush the output buffer after each question: this can be done by using, for example, `fflush (stdout)` in C or C++, `System.out.flush ()` in Java, `flush (output)` in Pascal or `sys.stdout.flush ()` in Python.

   You can ask no more than 20 000 questions.

2. Output the answer. In this case, your program must output two lines.

   If all vectors lie in one semi-plane, the first line must be "`! YES`", and the second must contain n space-separated distinct integers from 1 to n: the indices of vectors in sorted order.

   If there is no such semi-plane, output "`! NO`" on the first line. The second line must start with an integer k, the number of vectors in the proof, followed by a sequence of k distinct integers from 1 to n: the proof itself. If there are several possible proofs, output any one of them.

   After printing the answer, your program must immediately terminate gracefully.

Examples

standard input	standard output
3 0 1 0	 ? 1 3 ? 3 2 ? 2 1 ! YES 3 1 2
3 1 0 0	 ? 1 2 ? 3 2 ? 1 3 ! NO 3 1 2 3

Problem J. Center of List of Sums

Input file:	sums-center.in
Output file:	sums-center.out
Time limit:	4 seconds
Memory limit:	256 mebibytes

You are given two arrays a and b of same length n.

Consder all n^2 possible pairwise sums $a_i + b_j$ and print them in non-decreasing order. Your task is to find n numbers which are exactly in the middle of this list, that is, n elements of the sorted list with indices from $\frac{n \cdot (n-1)}{2} + 1$ to $\frac{n \cdot (n+1)}{2}$. List elements are numbered staring from 1.

Input

The first line contains an integer n ($n \le 2 \cdot 10^5$).

The second line contains n integers a_1, a_2, ..., a_n ($0 \le a_i \le 10^9$).

The third line contains n integers b_1, b_2, ..., b_n ($0 \le b_i \le 10^9$).

Output

Print n integers: the middle of the sorted list of pairwise sums in their respective order.

Examples

sums-center.in	sums-center.out
2 1 3 2 4	5 5
3 1 2 3 4 6 8	7 8 9

Explanations

In the first example, the sorted list of pairwise sums looks as follows: 3, 5, 5, 7.

In the second example, the list is: 5, 6, 7, 7, 8, 9, 9, 10, 11.

Problem K. Cookies

Input file:	`word-chains.in`
Output file:	`word-chains.out`
Time limit:	4 seconds
Memory limit:	256 mebibytes

There were several cookies lying on a platter in a row. Each cookie was shaped as a lowercase English letter, and together, the cookies formed an English word.

Eve approached the platter and ate one cookie. To keep it in secret, she arranged the remaining cookies in such a way that they formed some other word. No one around noticed the difference. So Eve decided to repeat her venture. And she repeated it several times. At the end, she had no possibility to take one cookie and either arrange others to form a word or leave the platter empty.

Eve did not turn or flip the cookies, so a letter never turned into another letter. Eve consulted the dictionary to be sure that she was using real words.

How could the platter look at different moments of time, assuming that Eve managed to eat the maximum number of cookies?

You will have to solve the problem for several platters.

Input

The first line of input contains the number of test cases n ($1 \le n \le 10^4$). The next n lines contain words, one word per line. Each word defines one initial state of a platter with cookies. The next line contains the size of the dictionary: $m = 173\,554$ words. Then follow m lines with dictionary words, one word per line. The dictionary is the same for all inputs. It is the public domain word list ENABLE for word games in English, slightly edited for this problem (one-letter words were added). The dictionary used in this problem can also be downloaded separately here: `http://acm.math.spbu.ru/171015/words.unix.txt` with Unix line endings or `http://acm.math.spbu.ru/171015/words.windows.txt` with Windows line endings. It is guaranteed that all the initial words are contained in the dictionary. It is not guaranteed that Eve can eat at least one cookie from each platter.

Output

Output n chains, each chain on two lines. The first of these lines must contain the length of the chain, that is, the number of states in it. On the second line, output all the states of the platter in the chain. Output each state either as the respective dictionary word or as a dot ("`.`") if the platter is empty. The states must be separated by the sequence of characters "` -> `". See examples for better understanding of the output format.

In case there exist multiple chains of maximal length for a given initial word, output any one of them.

Example

word-chains.in	word-chains.out
5 university championships open cup cookie 173554 a aa aah ... zyzzyva zyzzyvas	11 university -> intrusive -> neuritis -> unities -> seniti -> nisei -> sine -> sei -> es -> e -> . 2 championships -> championship 5 open -> one -> ne -> e -> . 4 cup -> up -> p -> . 1 cookie

Explanation

As you can notice, the dictionary is not fully present in the statement. To get a correct answer to the example, replace the dictionary text with its complete version.

The output chain for the "`university`" test case is divided into three lines just for readability. In fact, each chain must be printed on a single line.

Problem L. Xor-fair Division

Input file:	xorsep.in
Output file:	xorsep.out
Time limit:	2 seconds
Memory limit:	256 mebibytes

Boys Arthur and Lyoshka found an array of non-negative integers A. As there was only one array, they decided to divide the array elements between themselves so that each element of the array was given to one of the boys and each boy took at least one element.

As the boys are very fond of the bitwise "xor" ($\bigoplus$) operation, they decided to divide the array xor-fairly. The division of the array A into two arrays B_1 and B_2 is xor-fair if

$$\bigoplus_{x \in B_1} x = \bigoplus_{y \in B_2} y.$$

Now, the boys are interested in the following question: in how many ways they can divide the array xor-fairly? Two ways are considered different if there is at least one element that is given to different boys in these two ways.

Input

The first line of the input contains an integer n, the length of the array A ($1 \le n \le 50$). The second line contains n non-negative integers $A_1, A_2, \ldots, A_n$: the elements of the array ($0 \le A_i < 2^{63}$).

Output

Output a single integer: the number of ways for Arthur and Lyoshka to divide the array A xor-fairly.

Examples

xorsep.in	xorsep.out
3 0 1 2	0
2 1 1	2

Explanations

In the second example, there are two ways: Lyoshka can take the first element and Arthur can take the second, or the other way around.