Problem A. Armoring

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

In this problem you are to implement armoring of one side of a tesla-tank. A side of a third generation tesla-tank can be viewed as a rectangle grid with N rows and M cells in each row. Some cells contain communication centers of the tank and should not be armored. So, you need to armor all other cells by tesla-armor blocks. Dealers of VeraNN Company, the supplier of these tesla-tanks, offer armor blocks of sizes $1 \times R$ cells for any R (you may rotate the blocks to get sizes $R \times 1$).

You are going to buy several armor blocks of equal sizes to armor the tank side. In order to do that, you first have to find the maximal integer K such that at least one armor block of size K can be placed at the side aligned to the grid without armoring communication centers (i. e. no armor block of sizes $(K+1)$ or more can be used). Now you should determine whether it is possible to armor the tank using only blocks of size K. Moreover, you are interested in the number of ways to perform armoring. More ways, more profit!

Input

The first line of input file contains two integers N and M — sizes of the tank side ($1 \leqslant N, M \leqslant 1000$). The following N lines describe the cells of the side. Each of these lines contains M symbols '#' and '.' where '#' corresponds a to communication center cell and '.' corresponds a to free cell. It is guaranteed that the value of K for the given board is greater than 1.

Output

On the single line of output file, print the amount of ways for armoring the given tank side or 0 if it is impossible. Two ways are different if there exists at least one cell that is covered by a block $K \times 1$ in the first way and by a block $1 \times K$ in the second way.

Remember that each free cell should be covered by some armor block, communication centers should not be covered, and different armor blocks cannot overlap.

Example

armor.in	armor.out
2 2	2

Problem B. Bomb

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

VeraNN Company, specializing in production tesla-tanks, created a weapon of the New Age — nanochugunium bomb to test their production for endurance. The bomb is a solid polyhedron consisting of a huge amount of chugunium nanocrystals. The bomb developers want to compute its destructive power before the testing. The bomb's destructive power is proportional to its weight, and, respectively, its volume, because the bomb's matter is homogeneous, all nanocrystals are carefully adjusted to each other, and there are no defects.

Now, the main bomb developer asked you to write a program calculating the volume of the bomb. You have only five hours. In case you do not succeed, he threatens to test your endurance with this bomb.

Input

There is one integer n in the first line — the amount of vertices of the bomb-polyhedron ($1 \leqslant n \leqslant 10^5$). After that, there are n lines, i-th of which contains coordinates x_i, y_i and z_i of i-th vertex ($0 \leqslant x_i, y_i, z_i \leqslant 10^6$). The next line contains a single integer m — the amount of facets ($4 \leqslant m \leqslant 10^5$). In the next m lines there are triples of numbers from 1 to n inclusive — numbers of vertices that constitute some facet. All coordinates are integers. It's guaranteed that the bomb's surface is connected, the surface of the bomb contains no self-intersections and no self-touchings.

Output

You should print one number — the volume of the bomb. You answer should have absolute error of no more than 10^{-6}. The answer should be written in the following form: "[+|-]<digits>[.<digits>]" where <digits> is a sequence of one or more digits.

Example

bomb.in	bomb.out
4 0 0 0 0 0 1 0 1 0 1 0 0 4 1 2 3 1 2 4 1 3 4 2 3 4	0.166667

Problem C. Assemble a Cubic

Input file:	`cubic.in`
Output file:	`cubic.out`
Time limit:	3 seconds (*4 seconds for Java*)
Memory limit:	256 mebibytes

The *Cubic* is a modern technology concept introduced by VeraNN Company. It allows tesla-tanks to anticipate the enemy actions. A Cubic consists of nine rectangular blocks 1×3. Each block consists of three cells 1×1. There are five different types of Cubic cells:

1. `hole` — a cell with a round reach-through hole in the middle.
2. `neutral` — an usual cell, flat at both surfaces.
3. `upper tooth` — a cell with half-sphere ledge at the upper surface, flat at the lower surface.
4. `lower tooth` — a cell with half-sphere ledge at the lower surface, flat at the upper surface.
5. `beaver` — a cell with half-sphere ledges at the both surfaces (it contains `lower tooth` and `upper tooth`).

In order to synchronize a Cubic with the AI module of the tank, it is necessary to assemble all the nine blocks together. More specifically, it is necessary to cover a rectangle 3×3 by three layers (lower layer, middle layer and upper layer) of a Cubic using these nine blocks. Each of the blocks can be placed either vertically (as a sub-rectangle 1×3) or horizontally (as a sub-rectangle 3×1). Additionally, there are several natural limitations for covering:

- a `neutral` cell can be placed at the lower layer or if the corresponding cell of the previous layer doesn't have an `upper tooth`.
- a cell containing a `lower tooth` can be placed at the lower layer or if the corresponding cell of the previous layer has an empty `hole`. An empty hole is a hole that is not already used by an `upper tooth`; (two different teeth can't be placed at the same hole).
- a `hole` can be placed without any limitations.

It is allowed to turn blocks (in this case, the first and the third cells will be swapped) and invert blocks (in this case, a `lower tooth` becomes an `upper tooth` and vice versa). So, there are eight ways to place a block in terms of its orientation. Of course, there may be a lot of different ways to assemble Cubic, but for some sets of blocks there is no way to assemble it. The number of ways to assemble Cubic is called the "power" of the set of blocks. Your task is to find the power of given set of nine Cubic blocks. All blocks are labeled by unique identifiers, so any two coverings which differ in at least one block placement and/or orientation should be counted as different.

Input

Input file contains nine lines. Each line consists of three integer numbers between 1 and 5 — types of the corresponding block of the Cubic (in accordance with their enumeration in the problem statement).

Output

The first line should contain a single integer K — the number of ways to assemble the Cubic. If $K \leqslant 9000$, all K possible constructions of Cubic should be printed in lexicographic order. Otherwise, you should just print "`It's over nine thousand!!!`" on the second line.

Each Cubic construction should be described in the following way. First, the lower layer should be described. The first line of the layer description should contain 0 if the layer consists of horizontally oriented blocks, or 1 for vertically oriented blocks. After that, the blocks of the layer should be described in order from left to right (for vertical layer) or from top to bottom (for horizontal layer). Each block is described by three integers — index of the block (blocks are indexed from 1 to 9 in the order they appeared in the input file), rotation (0 if the block is not rotated, 1 if it is rotated relative to the description in the input file), inversion (0 if the block is not inverted, 1 otherwise). Each block should be described on a single line, integers in the description should be separated by exactly one space. After description of the lower layer, the middle and the upper layers should be described in the same format.

Examples

cubic.in	cubic.out
1 1 1 2 2 2 1 1 1 2 2 2 1 1 1 2 2 2 1 1 1 2 2 2 1 1 1	761014517760 It's over nine thousand!!!
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5	0
1 1 2 3 1 2 3 4 4 2 4 4 1 5 2 5 2 5 3 5 1 1 2 3 1 2 3	8192 0 2 0 1 3 0 0 4 0 0 0 5 0 0 1 0 0 8 0 0 1 6 0 0 7 1 0 9 1 0 <... 8190 more constructions ...> 1 9 1 1 7 1 1 6 1 1 0 5 1 1 1 1 1 8 1 1 0 2 1 0 3 1 1 4 1 1

Problem D. Flow

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

VeraNN Company have created a modern tesla-engine for tesla-tanks. The main advantage of this engine is that its power doesn't depend on the amount of fuel received from the fuel supply system. Moreover, this engine can work without any fuel and provide the same power. In order to minimize the fuel expenses, it was decided to upgrade the fuel system so that the amount of fuel consumed by the tesla-engine in a unit of time (the *flow value*) will be as small as possible.

The fuel system can be represented by an undirected graph with N vertices (one of these vertices corresponds to the engine) and M edges between vertices. Each edge has its own capacity. Also, the fuel system has a fuel storage (also represented as a vertex of this graph). The flow value is the maximal flow in this graph from the storage vertex to the engine vertex. To upgrade the fuel system, it is allowed to change endpoints of at most K edges without changing their capacity.

Your task is to find the minimal flow value after upgrading the fuel system.

Input

The first line of input file contains three integers: N, M and K ($3 \leqslant N \leqslant 20$, $0 \leqslant K \leqslant M \leqslant 2000$). Vertices are numbered from 1 to N. Vertex 1 corresponds to the fuel store, vertex N corresponds to the tesla-engine. Each of the following M lines contains three integers U, V and C ($1 \leqslant U, V \leqslant N$, $U \neq V$, $1 \leqslant C \leqslant 10^6$) — the endpoints of corresponding edge and its capacity. The fuel system cannot contain loops and direct edges from store to engine, but it may have multiple edges between some pairs of vertices. After the upgrade, the system should satisfy the same requirements.

Output

On the first line of output file print a single integer — the minimal flow value after upgrading the system.

Example

flow.in	flow.out
4 5 1 1 2 2 1 3 3 2 3 4 2 4 3 3 4 2	2

Problem E. Hexadecimal's Lair

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

As you may know, there is only way to Hexadecimal's Lair. It goes along corridors and portals. Every corridor consists of M halls and they are connected in a circular way: from the first hall, it is possible to go to halls 2 and M, from the second hall — to halls 1 and 3 and so on. There are N corridors in total; one of the halls in corridor k ($1 \leqslant k < N$) contains a bidirectional portal to some hall in corridor $k+1$. The last one (corridor N) contains a portal to Hexadecimal's throne (you shouldn't even think of going there, so we won't give you its location). If a portal from i-th hall of k-th corridor is connected to j-th hall of corridor $k+1$, we will call it a (k, i, j)-portal.

The large number of halls, however, is not the only trouble: from time to time, the portals themselves are changing their positions. More precisely, a (k, i, j)-portal could become a (k, p, q)-portal for any p and q such that $1 \leqslant p, q \leqslant M$.

Your task is to find the shortest way between some pairs of halls in this ever-changing environment. Initially, a portal from corridor k to corridor $k+1$ is always a $(k, 1, 1)$-portal.

Input

First line of input consists of three positive integers N, M and T ($2 \leqslant N \leqslant 10^5$, $1 \leqslant M \leqslant 10^9$, $1 \leqslant T \leqslant 10^5$). Each of the next T lines consists of the name of a query and its arguments. If the name of a query is "`DIST`", it is followed by four integers k_0, i_0, k_1 and i_1 ($1 \leqslant k_0, k_1 \leqslant N$, $1 \leqslant i_0, i_1 \leqslant M$). They represent the start and finish positions, where k_j is the corridor number and i_j is the hall number. If the name of a query is "`MOVE`", it is followed by three integers k, p, and q ($1 \leqslant k < N$, $1 \leqslant p, q \leqslant M$) which mean that the portal between corridors k and $k+1$ becomes a (k, p, q)-portal. It is guaranteed that Hexadecimal's throne won't be used in input.

Output

For each "`DIST`" query, you should write the length of the shortest way between the given pair of halls on a separate line. The length of a way between two halls is the number of halls on the way minus one.

Example

lair.in	lair.out
3 3 9 MOVE 1 1 2 MOVE 2 1 3 DIST 1 3 2 2 MOVE 1 2 3 MOVE 2 2 2 DIST 3 1 3 3 MOVE 1 1 2 DIST 2 2 2 1 DIST 3 3 2 2	2 1 1 2

Problem F. Death Levers

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

> Tesla-bomb is a nanochugunium bomb of 80th generation. The bomb is extremely powerful. It is guaranteed that detonation of one tesla-bomb will destroy the Earth. Usage of tesla-bombs is strictly prohibited by law.
>
> — from Teslapedia.

Student Misha is deeply disappointed about his programming skills. So he decided to destroy the world. Misha has found a tesla-bomb control panel located at the National Tesla Technologies Museum. The panel contains N buttons and each button corresponds to exactly one lever at the bomb (different buttons correspond to different levers). Each lever has two different states: "UP" and "DOWN". Pressing a control panel button changes state of the corresponding lever to the opposite one. To detonate the bomb, it is necessary that at least K different levers will be in state "UP" at the same time. Misha can press buttons to change the states of the levers, but does not have much time and he may perform at most $6N + 6$ pressings.

Unfortunately, Misha doesn't know the initial states of the levers. Also, his programming skills are really poor... You should definitely help him! Your task is to find a sequence of button pressings so that the world will be destroyed after (or during) performing it regardless of initial states of the levers.

Input

Input file contains two integers: N and K ($2 \leqslant N \leqslant 25$, $1 \leqslant K < \frac{2N}{3}$).

Output

The first line of the output file should contain only one integer M — the number of button pressings. The second line should contain M integers — indices of buttons to be pressed (in the order they will be pressed). It is allowed to press a single button several times. Buttons are numbered from 1 to N. M should not be greater than $6N + 6$. If several solutions exist, output any of them. It is guaranteed that at least one solution always exists.

Example

levers.in	levers.out
2 1	2 1 2

Problem G. Medal of Honor

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

After accomplishment of United Aggressive Operation of Tesla-Arms, it was decided to reward the most successful tesla-tank commanders. Of course, destroying civilian aggregates, unreasonable aggression, destruction of monuments and other such things should be awarded. In accordance with this new order, several tesla-tank commanders were given special "Medal of Honor" awards. These medals are triangle plates of the same width.

Two best friends — Vovan and Diman — were rewarded too. But they found that their triangle medals have different forms... It is not known who expressed claims first, but Vovan requested your help to find which of the two medals is heavier. Using your geometry skills, you quickly found out that the medals have equal volumes. But Vovan doesn't believe you... He only agreed that both the plates have the same width. So, you need to cut the medals into several parts (each of them should be a triangle) and construct identical convex polygons from these parts in order to assure Vovan. Your task is to find at least one possible solution for this problem.

Input

The first line of input file consists of three integers — lengths of the sides of the first triangle in clockwise order. On the second line, the second triangle is described in the same format. It is guaranteed that triangles are non-degenerate and have equal areas. All numbers in the input are positive integers not exceeding 100.

Output

In order to describe the cutting and construction, you have to introduce Cartesian coordinates for both triangles and the resulting polygon.

In the first line, you are to describe the initial triangle corresponding to the first medal, in the second line — the triangle corresponding to the second medal. Each triangle should be described by six real numbers on a single line — coordinates of the triangle vertices in clockwise order.

In the third line, the resulting polygon should be described. The first line of the description should contain N — the number of polygon vertices — followed by N pairs of real numbers — coordinates of the vertices in clockwise order.

The next line should contain a single integer K_1 — the number of parts in the partition of the first medal. On the following K_1 pairs of lines, each of the parts should be described: on the first line — where it was before partition in the same coordinate system as the first triangle, on the second line — where it is after construction in the same coordinate system as the polygon. Each of these lines contains six real numbers — the respective coordinates of the part's vertices in clockwise order. It is allowed to start part description on each line from any vertex.

After that, the second medal partition should be described in the same format.

The resulting polygon in the output should be convex, non-degenerate and contain at most ten vertices. It is allowed to move and turn triangles of partition, but not to reflect them. Each of the medals should have at most 1000 parts. The coordinates of polygon and triangles should not exceed 10^5 by absolute value and should be printed with at least six digits after decimal point.

If there are several solutions, you may output any of them. You may assume that at least one solution always exists.

Example

medal.in	medal.out
3 5 4 3 5 4	0 0 0 3 4 0 0 0 0 3 4 0 3 0 0 0 3 4 0 1 0 0 0 3 4 0 0 0 0 3 4 0 1 0 0 0 3 4 0 0 0 0 3 4 0

Problem H. Hexadecimal's Prisoner

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

When Hexadecimal has caught Bob again, she bound him to her throne by a literal string consisting of N symbols. After that, she flew away. Perhaps there will be some destruction in town soon.

Bob found a note with a secret hint to escape. He should divide the string into K non-empty substrings. After division, the resulting substrings are concatenated in reversed order: if the division separates s into substrings s_1, s_2, ..., s_k, the resulting string will be $s_k s_{k-1} \dots s_2 s_1$.

For example, if the string is equal to `ABCAABA` and $K = 4$, Bob can divide it into substrings `AB|CA|A|BA` and get `BA|A|CA|AB` as a result.

To leave Hexadecimal's Lair, Bob needs to divide the string in such a way that the resulting string is lexicographically minimal. He has only one chance. Help Bob, don't fail.

Input

The first line of input file contains numbers K and N ($1 \leqslant K \leqslant N$). The second line consists of N ($1 \leqslant N \leqslant 2 \cdot 10^5$) uppercase Latin letters — the string Bob is bound with.

Output

On the first line of output file, you should write a string of N symbols. It should be the lexicographically minimal string that Bob can obtain by the operation described.

Examples

prisoner.in	prisoner.out
2 7 ABACABA	AABACAB
4 11 ABRACADABRA	AABRACADABR

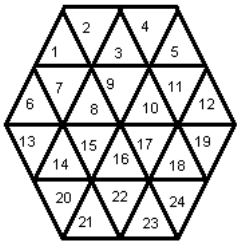
Problem I. Hexadecimal's Puzzle

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

There are lot of people who wants to visit the beautiful programming world — the Mainframe. It's so nice to walk in the Floating Point park with a funny sprite, to dive into the Game Cube with Bob the Guardian...

However, in this brilliant place, there are many dangers. For example, the location called Lost Angles is rather scary. There, the evil chaotic virus Hexadecimal lives. At the risk of losing some parts of your body, you may be a victim of her unusual amusements.

Here is a little example. First, Hexadecimal draws a figure that looks like a regular hexagon divided into K small triangles. Each triangle is assigned a unique number for 1 to K. When the virus finishes numbering the triangles, she tears the hexagon into separate triangles. All triangles are equilateral and have equal size. For each triangle, you know the numbers of its neighbours. Two triangles are called neighbours if they share a common side.



After that, Hexadecimal asks you to compose the triangles to form a regular hexagon. The hexagon should be the lexicographically smallest one, not any other one. Otherwise, she will make some tasty cookies. Of you. Yami-yami.

Input

In the first line there is one number K — the amount of triangles ($1 \leqslant K \leqslant 15\,000$). The next K lines describe the triangles: the first integer n_i on i-th line is the amount of neighbours of i-th triangle, the next n_i integers are their numbers. Triangles are numbered from 1 in the order they are given in the input.

Output

If there is no such hexagon, output "`NO`" and prepare to be eaten. Otherwise, print "`YES`" and then the hexagon, line by line. You answer should be the lexicographically smallest possible hexagon. Trailing and leading dash signs ('`-`') should be added to lines that have less triangles than the longest ones. Consecutive tokens should be separated by a single space. See examples for further clarification.

A regular hexagon A is said to be lexicographically smaller than a regular hexagon B of the same size if after removing all dash signs from the output representation, A is less than B as a sequence of integers.

Examples

puzzle.in	puzzle.out
6 2 2 4 2 1 3 2 2 6 2 5 1 2 4 6 2 5 3	YES 1 2 3 4 5 6
24 2 2 7 2 1 3 3 2 4 9 2 3 5 2 4 11 2 7 13 3 6 8 1 3 7 9 15 3 8 10 3 3 9 11 17 3 10 12 5 2 11 19 2 14 6 3 13 15 20 3 14 16 8 3 15 17 22 3 16 18 10 3 17 19 24 2 18 12 2 21 14 2 20 22 3 21 23 16 2 22 24 2 23 18	YES - 1 2 3 4 5 - 6 7 8 9 10 11 12 13 14 15 16 17 18 19 - 20 21 22 23 24 -

Problem J. Hexadecimal's Theorem

Input file:	`theorem.in`
Output file:	`theorem.out`
Time limit:	3 seconds (*5 seconds for Java*)
Memory limit:	256 mebibytes

Hexadecimal is bored. Again. Sometimes she's throwing fireballs at her pictures, sometimes she's flying above the Lost Angles.

Suddenly, Hexadecimal formulated a theorem. Her theorem says it is possible to find such B and C for every A that $A^2 + B^2 = C^2$.

Play with the virus. Knowing that her the number she likes is A, find such B and C that they will fulfill the condition above.

Input

The only line of input contains a single positive integer A. Its length is no more than 10^5 decimal digits.

Output

Output must consist of two positive integers B and C separated by a space such that $A^2+B^2 = C^2$. If there are no such B and C with length no more than 10^6 decimal digits, you should output "`I'm sorry, Hexy`".

Example

theorem.in	theorem.out
3	4 5

Problem K. Productivity

Input file:	work.in
Output file:	work.out
Time limit:	3 seconds (*4 seconds for Java*)
Memory limit:	256 mebibytes

After analyzing the work efficiency of AI department programmers which develop software for third generation tesla-tanks, the VeraNN Company president has decided to improve their productivity. There are only two ways to increase the productivity: to increase the salaries or to give programmers a lot of beer. However, the long-lasting experience shows that increasing salary doesn't improve the desire for work. On the other side, programmers like beer very much. But beer is expensive too... So, there is a difficult problem — to find how many bottles of beer would be enough for a programmer to improve his productivity to the necessary level.

In school years, the president heard about a strange algorithm named "binary search"... Here is how he remembers it:

```
int l = m, r = n, result = n + 1;
while (l <= r) {
    int c = f(l, r);
    if (enough(c)) {
        result = c;
        r = c - 1;
    }
    else
        l = c + 1;
}
```

where `enough(x)` function returns "`true`" if x bottles are enough and "`false`" otherwise. Function $f(x, y)$ returns an integer from interval $[x, y]$.

Obviously, it is necessary to spend exactly x bottles of beer to find the value of function `enough(x)`. Besides, it is known that $(M - 1)$ bottles are not enough and $(N + 1)$ bottles are enough to satisfy a programmer. Of course, if x bottles of beer are enough for a programmer, then $(x + 1)$ or more bottles are enough too.

Your task is to find how many bottles of beer are necessary in the worst case in order to find the minimal amount of beer bottles necessary for a programmer if function $f(x, y)$ is chosen optimally.

Input

Input file contains two integers M and N ($1 \leqslant M \leqslant N \leqslant 3000$).

Output

Output file should contain a single integer — the answer for the problem.

Examples

work.in	work.out
1 1	1
1 5	9