Problem A. Battle

Input file:	`battle.in`
Output file:	`battle.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

There are n cities on a small island, numbered 1 through n. Some pairs of cities are connected with bidirectional roads. All the cities were independent. All the cities lived in harmony with each other.

One day two cities s and t decided to create their own countries and conquer some of the other independent cities. We'll denote the country started by city s as the first country, and the country started by city t as the second country.

Suppose some day the first country contains cities from $A = \{a_1, \ldots, a_{k_a}\}$ and the second country contains cities from $B = \{b_1, \ldots, b_{k_b}\}$. We define $neigh(A)$ as set of cities connected with some city from A by a road, but not from A itself (so $neigh(A) \cap A = \emptyset$). We also define $popul(i)$ as the population of city i, and $popul(A) = \sum\limits_{i \in A} popul(i)$.

The first country can conquer a set C of independent cities at once iff

$$popul(A \cap neigh(C)) > popul(C) + popul(B \cap neigh(C))$$

In a similar manner the second country can conquer the set C at once iff

$$popul(B \cap neigh(C)) > popul(C) + popul(A \cap neigh(C))$$

.

In the morning of each day the first country can conquer some cities, and in the evening the second country can conquer some cities.

The first country wants to conquer the cities in such a way that in the end $popul(A) - popul(B)$ is maximal possible, and the second country wants to conquer the cities in such a way that in the end $popul(B) - popul(A)$ is maximal possible.

Input

The first line contains three positive integers n, s and t ($3 \le n \le 13$, $1 \le s, t \le n$, $s \ne t$). Next n lines contain n characters each, with the j-th character in the i-th of those lines being '`1`' if cities i and j are connected by a road, and '`0`' otherwise. The next line contains n integers $popul(i)$ ($1 \le popul(i) \le 100\,000$) — the populations of all cities.

Output

Output the final value of $popul(A) - popul(B)$ assuming that both countries act optimally.

See next page for sample input and output.

Sample input and output

battle.in	battle.out

5 1 2
00111
00001
10010
10101
11010
12 100 5 5 7
	-85

3 1 3
010
101
010
5 5 11
	-11

7 1 5
0100011
1010001
0101001
0010100
0001000
1000001
1110010
90 20 20 10 200 85 80
	85

Problem B. Beatiful graph

Input file:	`beautifulgraph.in`
Output file:	`beautifulgraph.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

Misha is currently interested in undirected graphs that contain no two simple cycles sharing exactly one edge. Let's call them *beautiful graphs*. He wants to find the maximal beatiful graph, that is the beautiful graph that contains the most edges among all beautiful graphs with at most n vertices. But Misha is going to leave on vacation, so he asked you to help him with this problem.

Input

The input file contains a single integer n ($1 \le n \le 100$) — the maximum number of vertices your graph can contain.

Output

Output the number of vertices V ($1 \le V \le n$) and the number of edges E of your graph to the first line of the output file, separated with a space. Then output E lines with two integer numbers each, again separated with a space. The two numbers should be the numbers of the vertices connected by the corresponding edge. The vertices of the graph are numbered from 1 to V. You can output edges in any order. If there are several maximal graphs, output any.

Sample input and output

`beautifulgraph.in`	`beautifulgraph.out`
`1`	`1 0`

`2`	`2 1`
	`1 2`

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

Problem C. Control function

Input file:	`control.in`
Output file:	`control.out`
Time limit:	5 seconds
Memory limit:	256 megabytes

A matrix T of non-negative integers with n rows and m columns is called a *control matrix* when its first row is different from all other rows. Formally speaking, $\forall(2 \le i \le n)\ \exists j\ T_{1j} \ne T_{ij}$.

A function f from non-negative integers to non-negative integers is called a *control function* for the given control matrix T when the matrix $f(T)$ obtained by applying f to every element of T is also a control matrix. Formally speaking, $\forall(2 \le i \le n)\ \exists j\ f(T_{1j}) \ne f(T_{ij})$.

Find a control function with all values not exceeding 50 for the given control matrix T.

Input

The first line of the input file contains two integers n and m $(1 \le n, m \le 1000)$. The next n lines contain m integers each, representing the matrix T_{ij} $(0 \le T_{ij} \le 1\,000\,000\,000)$. It is guaranteed that the matrix T is a control matrix.

Output

Output "`Yes`" (without quotes) to the first line of the output file if such a function exists, and "`No`" (without quotes) otherwise. If the answer is positive, then output the function via "`key -> value`" pairs (without quotes). Order keys in increasing order. All different numbers from matrix T must appear as a key exactly once, and no other keys should be printed.

Sample input and output

control.in	control.out
1 5 1 2 3 4 5	Yes 1 -> 0 2 -> 0 3 -> 0 4 -> 0 5 -> 0

2 2
1 2
1 3
	Yes
	1 -> 1
	2 -> 1
	3 -> 0

4 2
0 2
4 5
7 6
3 1
	Yes
	0 -> 1
	1 -> 0
	2 -> 1
	3 -> 0
	4 -> 1
	5 -> 0
	6 -> 1
	7 -> 0

Problem D. Double cyclic

Input file:	`cyclic.in`
Output file:	`cyclic.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

Consider a sequence of numbers $a_1, a_2, \ldots, a_n$. Its *cyclic shift* is any sequence obtained from it by taking some (possibly none) of its starting elements and moving them to the end in the same order. In other words, it is a sequence $a_k, a_{k+1}, \ldots, a_{n-1}, a_n, a_1, a_2, \ldots, a_{k-2}, a_{k-1}$ for some k.

The *smallest cyclic shift* of a given sequence is its cyclic shift that is smallest lexicographically. A sequence is *lexicographically smaller* than another sequence of the same length if and only if it has a smaller number in the first position they differ.

Now, consider a sequence σ of integers $a_1, a_2, \ldots, a_n$ where $\forall i\ 0 \le a_i \le m-1$, where m is some fixed constant. We define $(\sigma + 1) \mod m$ to be the sequence $(a_1 + 1) \mod m, (a_2 + 1) \mod m, \ldots, (a_n + 1) \mod m$.

Given σ, m and an integer k between 1 and n, output m integer numbers: the k-th element of the smallest cyclic shift of σ, the k-th element of the smallest cyclic shift of $(\sigma + 1) \mod m$, …, the k-th element of the smallest cyclic shift of $(\sigma + (m-1)) \mod m$.

Input

The first line of the input file contains three integers n, m and k ($1 \le n, m \le 50\,000$, $1 \le k \le n$). The second line of the input line contains n integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le m-1$).

Output

Output m integer numbers one per line — k-th elements of the smallest cyclic shifts of the sequences explained above.

Sample input and output

cyclic.in	cyclic.out

`5 6 3`
`1 2 1 2 3`
	`1`
	`2`
	`3`
	`5`
	`5`
	`0`

Problem E. Hamiltonian polyhedron

Input file:	`polyhedron.in`
Output file:	`polyhedron.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

You are given a convex polyhedron with the sum of its solid angles less than 4 (in steradians; the full angle is in this case equal to 4π). Find out whether there exists a cycle that passes through all vertices of the polyhedron exactly once. The cycle must have edges of the polyhedron as its edges, and no edge can be used more than once in the cycle. You can leave some of the edges of the polyhedron unused.

Input

The first line of the input file contains one integer number f — the number of facets of the polyhedron. Then all f facets are described. Each description begins with a line with single integer k — the number of vertices of the facet. The next k lines contain 3 real numbers each — the vertex coordinates. The points are given in the order they appear on the border of the facet (either clockwise or counter-clockwise). The numbers are given with exactly 9 digits after the decimal point. If a point is listed in several facet descriptions, its coordinates in all descriptions will be exactly the same.

The overall number of vertices of the polyhedron will be at most 100. The overall number of facets and the number of vertices on any facet will be at most 100 as well.

The coordinates of the points are not the exact coordinates in 3-dimensional space, they are rounded to 9 digits after the decimal point from their true value. Some of the points can be very close, but the distance between any two different points is at least $5 \cdot 10^{-7}$. The absolute values of the coordinates of the points do not exceed 1000.

Output

If there is no such cycle as described in the problem statement, output "`No`" on the first line of the output file, otherwise output "`Yes`". In the latter case, output the description of the cycle: on the second line output n — the number of vertices of the polyhedron; on the next n lines output the coordinates of the vertices in the order they appear along the cycle, exactly as they were given in the input file, with the same 9 digits after the decimal point.

Sample input and output

polyhedron.in	polyhedron.out

4
3
100.146488845 0.000000000 0.000000000
100.145878719 0.349576483 0.000000000
-100.145878719 -0.349576483 0.000000000
3
33.382162948 0.000000000 1.219611194
100.146488845 0.000000000 0.000000000
100.145878719 0.349576483 0.000000000
3
33.382162948 0.000000000 1.219611194
100.145878719 0.349576483 0.000000000
-100.145878719 -0.349576483 0.000000000
3
33.382162948 0.000000000 1.219611194
-100.145878719 -0.349576483 0.000000000
100.146488845 0.000000000 0.000000000
	Yes
	4
	-100.145878719 -0.349576483 0.000000000
	33.382162948 0.000000000 1.219611194
	100.146488845 0.000000000 0.000000000
	100.145878719 0.349576483 0.000000000

Problem F. Rebus

Input file:	`rebus.in`
Output file:	`rebus.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

An *addition rebus* is a puzzle where you get an addition equation, like `ABC+CBA=BDB`, and you're asked to replace each letter with a digit (from 0 through 9) in such a way that:

- Equal letters are replaced with equal digits.
- Different letters are replaced with different digits.
- None of the resulting numbers starts with 0, unless the entire number is just 0.
- The resulting equation holds.

A *proper addition rebus* is an addition rebus with exactly one solution. An *aligned addition rebus* is an addition rebus where all three involved numbers have the same length (number of digits). The above addition rebus is aligned, but isn't proper: `143+341=484` and `341+143=484` are its two possible different solutions (and there are more).

Find 1000 different proper aligned addition rebuses. Two rebuses are *different* if there is no one-to-one mapping from the set of 26 English letters on itself that transfers one of those rebuses into another. For example, `ABC+CBA=BDB` and `XAY+YAX=AZA` are not different, while `ABC+CBA=BDB` and `ABC+DEF=GHI` are different.

Input

The input file will be empty.

Output

Output any 1000 different proper aligned addition rebuses, one per line. Each rebus may only use capital English letters, symbols '`+`' and '`=`'. Each rebus must be at most 100 characters long.

Sample input and output

Note that this example output doesn't contain the required 1000 rebuses, it contains just two.

`rebus.in`	`rebus.out`

	`AABC+AABB=DBCB`
	`RRR+TTR=EDT`

Problem G. Problem Stacks

Input file:	`stacks.in`
Output file:	`stacks.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

Fedor and Sergey are playing a game while preparing for the World Finals. They have chosen a lot of problems to solve, and arranged the problem statements into n heaps, with i-th heap containing a_i problem statements, and put those heaps along one straight line. They make alternating moves, and each move consists of taking some (maybe all) problems from the first or from the last heap (but not from both) and solving them (and thus dumping the corresponding problem statements). When some player takes all problems from the first heap, the next heap is now considered first; when some player takes all problems from the last heap, the previous heap is now considered last. The player who doesn't have any more problems to solve loses.

Obviously, both Fedor and Sergey will play optimally. Fedor makes the first move. Who is going to win?

Input

The first line of the input file contains an integer n — the number of heaps ($1 \le n \le 5$). The second line of the input file contains n integers $a_1, a_2, \ldots, a_n$ ($\forall i\ 1 \le a_i \le 100\,000$) — the amounts of problems in each heap.

Output

Output "`FEDOR`" (without quotes) if Fedor will win, or "`SERGEY`" (without quotes) otherwise.

Sample input and output

stacks.in	stacks.out
3 5 5 5	FEDOR
4 3 1 2 3	SERGEY

Problem H. Unit-distance graph

Input file:	`unit.in`
Output file:	`unit.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

A graph is called *unit-distance graph* when it's possible to map its vertices to points on a plane in such a way that the distance between the points that correspond to vertices that are connected by an edge is exactly 1, and the distance between the points that correspond to vertices that are not connected by an edge is not equal to 1 (but still greater than 0).

Given a graph, find out whether it's a unit-distance graph, and if yes, output the corresponding points on the plane.

Input

The first line of the input file contains two integers n and m — the number of vertices and edges of the graph ($1 \le n \le 7$). The next m lines contain two distinct integers each, describing two vertices connected by an edge (the vertices are numbered starting from 0). Every two vertices can be connected by at most one edge.

Output

If the graph is unit-distance, print "Yes" (without quotes) on the first line of output file, otherwise print "No" (without quotes). In the former case, then print n lines, each containing two floating-point numbers not exceeding 100 by absolute value — the coordinates of the corresponding points in the order the vertices are numbered. Your solution will be considered correct if no two points are closer than 10^{-2}, the distance between points corresponding to connected vertices differs from 1 by no more than 10^{-7}, and the distance between two points corresponding to non-connected vertices differs from 1 by at least 10^{-2}.

Sample input and output

unit.in	unit.out

4 6
0 1
0 2
0 3
2 1
3 1
2 3
	No

5 6
0 1
1 2
2 3
3 0
3 4
4 0
	Yes
	0.0 0.0
	1.0 0.0
	1.0 1.0
	0.0 1.0
	-0.8660254037844386 0.5

Problem I. Wildcards

Input file:	`wildcard.in`
Output file:	`wildcard.out`
Time limit:	7 seconds
Memory limit:	256 megabytes

You are given a list of files in a directory and need to match a specific subset of these files. You've decided to do that with a single string containing wildcards.

More precisely, each file name is a string consisting of lowercase English letters ('a' through 'z'). You want to craft a string consisting of lowercase English letters, asterisks ('*'), and question marks ('?'). An asterisk matches any sequence of letters (including empty). A question mark matches exactly one letter. See examples for further clarification.

Given a list of file names that you want to match and the list of file names that you don't want to match, output a string (which may include wildcards) that will achieve that.

Input

The first line of the input file contains two integers n and m ($1 \le n, m \le 10$) — the number of file names to match and the number of file names not to match, respectively. The next n lines contain file names to match, one per line, each no more than 11 characters long, without any whitespace expect newlines. The next m lines contain file names not to match in the same format. All file names in the input file are different.

Output

Output one string that will do the required matches. The length of that string must not exceed 100. If there're several possible solutions, output any. If there's no solution, output "OOPS" (without quotes) instead.

Sample input and output

`wildcard.in`	`wildcard.out`

`2 2`
`list`
`tablexx`
`dataone`
`datatwo`
	`*l???`

Problem J. XYZX 2009

Input file:	`xyzx2009.in`
Output file:	`xyzx2009.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

The annual XYZX programming competition is well known for its unusual problem statements. The Technical Coordinator of the competition, also known as Snusmumrik, intentionally leaves some freedom in the problem statements, so that the participants have to guess the missing conditions from the sample tests, from the problem preface or even epigraph, or just invent something themselves. This peculiarity revealed itself in the latest XYZX-2009 competition as well.

Misha is competing in a round of that competition — Quarterfinal of Eniseysk, and he has bumped into that obstacle again: he needs to understand why does his perfectly correct program fail on the sample tests. The statements are in English, and he knows that there is only one false sentence in the whole problem statement. Moreover, he knows a set of rules that enables him to restore the original statement that the author had in mind. It is very simple: the word "not" can be inserted after any of the words "can", "may", "must", "should" in the text; also the word "no" can be inserted after any of the phrases "is", "are". Words "not" and "no" must be inserted exactly as written: "Not", "nO" and other variants are not permitted. The other words mentioned, however, are case insensitive, so you can insert "not" after "Can", and you can insert "no" after "ARE". If there is any article ("a" or "the") after "is" or "are" where you're inserting "no", you must also remove that article.

Misha wants to get the list of all the possible reasons that his program doesn't work on the sample tests. A reason here is a sentence where a negation can be applied. Remember that no two negations can be applied simultaneously, neither in one sentence nor in different sentences.

Input

The input file contains the text of the problem statement containing latin letters, single spaces and dots. Every sentence has a dot at the end (which is not considered the part of the last word). Words are separated with a single space. There is no space before any dot. There is exactly one space after each dot unless that dot follows the last sentence; in that case, the dot is followed by a newline, and then the input file ends. When inserting a word "not" or "no" after some word, you must first insert a single space, then the corresponding word immediately. The size of the input file doesn't exceed 1 024 bytes.

Output

The output file must first contain an integer k on a line by itself — the number of ways the problem statement can be understood differently. Then k lines must follow. Each line should contain one sentence from the initial text with exactly one negation applied. You must list all the possibilities (the lines are wrapped in the sample output just for better visibility; you must output one sentence per line). All the variants of negation must go in the natural order: all the negations that can be inserted in the first sentence, then all the negations that can be inserted in the second sentence, etc. Inside one sentence, first goes the negation that is applied after the first word (if applicable), then the one that is applied after the second word, etc.

You must not change the case of the letters in the words that you copy from the initial sentences. Your words must still be separated with single spaces, and the same rules apply to the dots at the ends of the sentences, as they do for the input sentences.

See next page for sample input and output.

Sample input and output

xyzx2009.in

```
There is a field with label K on the field. There is a field with label G on
the field. You must intersect the rectangle with each of the given rectangles
separately. Two different particles can annihilate each other in case of
collision. There must be a blank line after each test case output. Two
different ways to insert a symbol into the expression are considered the same
if the resulting expressions viewed as strings are equal. Misha can find out
whether this problem statement can be understood correctly at all.
```

xyzx2009.out

```
9
There is no field with label K on the field.
There is no field with label G on the field.
You must not intersect the rectangle with each of the given rectangles
separately.
Two different particles can not annihilate each other in case of collision.
There must not be a blank line after each test case output.
Two different ways to insert a symbol into the expression are no considered the
same if the resulting expressions viewed as strings are equal.
Two different ways to insert a symbol into the expression are considered the
same if the resulting expressions viewed as strings are no equal.
Misha can not find out whether this problem statement can be understood
correctly at all.
Misha can find out whether this problem statement can not be understood
correctly at all.
```