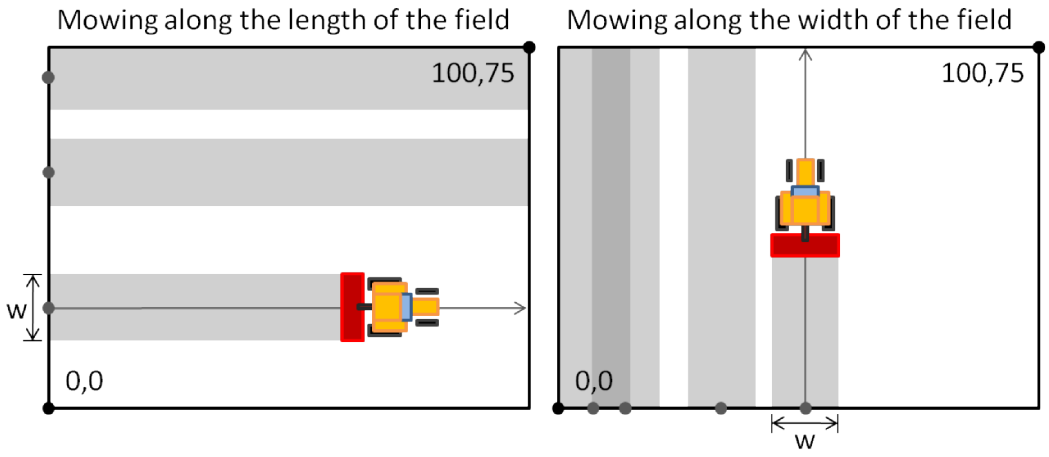
Problem A. Lawn mower

Input file: `a.in`
Output file: `a.out`

The International Collegiate Soccer[1] Competition (ICSC) is famous for its well-kept rectangular stadiums. The grass playing fields in ICSC stadiums are always 100 meters long, and 75 meters wide. The grass is mowed every week with special lawn mowers, always using the same strategy: first, they make a series of passes along the length of the field, and then they do the same along the width of the field. All passes are straight lines, parallel to the sides of the field.



The ICSC has hired a new lawn-mower, Guido. Guido is very chaotic, and instead of covering the field incrementally, he likes to choose random starting positions for each of his passes. But he is afraid of not doing a good job and being fired by the ICSC, so he has asked you to help him. Write a program to make sure that the grass in the field is perfectly cut: all parts of the field have to be mowed at least once when the mower goes from end to end, and again when the mower goes from side to side.

Input

Each test case contains three lines. The first line contains two integers, n_x ($0 < n_x < 1\,000$) and n_y ($0 < n_y < 1\,000$), and a real number w ($0 < w \leq 50$), which represents the width of the cut of that particular lawn mower. The next line describes the end-to-end passes (along the length of the field), and contains n_x real numbers x_i ($0 \leq x_i \leq 75$) describing the starting positions of the mower's center in Guido's end-to-end passes. The last line describes the side-to-side passes, with n_y real numbers y_i ($0 \leq y_i \leq 100$).

The end of the test cases is signalled with a line that contains the numbers `0 0 0.0`. You should generate no output for this line, as it is not a test case.

Real numbers for w, x_i and y_i can have up to seven digits after the decimal point, and any cut will also include its boundaries. For example, if a 2.0-meter wide cut is performed along the 10.0-meter mark, then a strip of grass from 9.0 to 11.0 (including both) will be considered "cut".

Output

Print "`YES`" if Guido has done a good job, or "`NO`" if some part of the field has not been mowed at least once when the mower was travelling along the length of the field, and again when it was travelling along the width.

[1] The ICSC is sponsored by the Association for Sports Machinery (ASM), which started out in the US, so they prefer to use the term "soccer" instead of "football".

Example

a.in

```
8 11 10.0
0.0 10.0 20.0 30.0 40.0 50.0 60.0 70.0
0.0 10.0 20.0 30.0 40.0 50.0 60.0 70.0 80.0 90.0 100.0
8 10 10.0
0.0 10.0 20.0 30.0 40.0 50.0 60.0 70.0
0.0 10.0 30.0 40.0 50.0 60.0 70.0 80.0 90.0 100.0
4 5 20.0
70.0 10.0 30.0 50.0
30.0 10.0 90.0 50.0 70.0
4 5 20.0
60.0 10.0 30.0 50.0
30.0 10.0 90.0 50.0 70.0
0 0 0.0
```

a.out

```
YES
NO
YES
NO
```

Problem B. Periodic points

Input file: `b.in`
Output file: `b.out`

Computing the number of fixed points and, more generally, the number of periodic orbits within a dynamical system is a question attracting interest from different fields of research. However, dynamics may turn out to be very complicated to describe, even in seemingly simple models. In this task you will be asked to compute the number of periodic points of period n of a piecewise linear map f mapping the real interval $[0, m]$ into itself. That is to say, given a map $f : [0, m] \to [0, m]$, you have to calculate the number of solutions to the equation $f^n(x) = x$ for $x \in [0, m]$, where f^n is the result of iterating function f a total of n times, i.e.

$$f^n = \overbrace{f \circ \ldots \circ f \circ f}^{nf's},$$

where $\circ$ stands for the composition of maps, $(g \circ h)(x) = g(h(x))$.

Fortunately, the maps you will have to work with satisfy some particular properties:

- m will be a positive integer and the image of every integer in $[0, m]$ under f is again an integer in $[0, m]$, that is, for every $k \in \{0, 1, \ldots, m\}$ we have that $f(k) \in \{0, 1, \ldots, m\}$.
- For every $k \in \{0, 1, \ldots, m-1\}$, the map f is linear in the interval $[k, k+1]$. This means that for every $x \in [k, k+1]$ its image satisfies $f(x) = (k+1-x)f(k) + (x-k)f(k+1)$, which is equivalent to its graph on $[k, k+1]$ being a straight line segment.

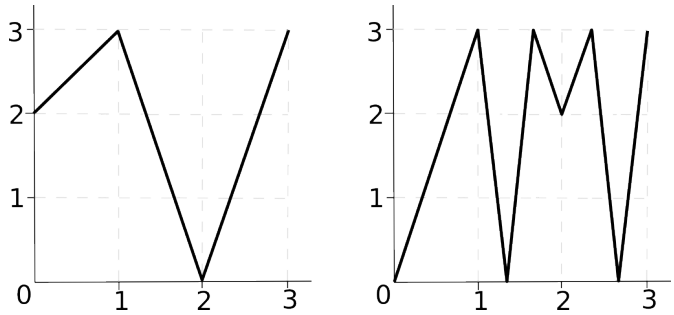


Figure 1: Graphs of the third map in the sample input, f_3 (left), and of its square, f_3^2 (right)

Since there might be many periodic points, you will have to output the result modulo an integer.

Input

The input consists of several test cases, separated by single blank lines. Each test case begins with a line containing the integer m ($1 \leq m \leq 80$). The following line describes the map f; it contains the $m+1$ integers $f(0)$, $f(1)$, ..., $f(m)$, each of them between 0 and m inclusive. The test case ends with a line containing two integers separated by a blank space, n ($1 \leq n \leq 5\,000$) and the modulus used to compute the result, mod ($2 \leq \text{mod} \leq 10\,000$).

The input will finish with a line containing 0.

Output

For each case, your program should output the number of solutions to the equation $f_n(x) = x$ in the interval $[0, m]$ modulo mod. If there are infinitely many solutions, print "`Infinity`" instead.

Example

b.in	b.out
2 2 0 2 2 10 3 0 1 3 2 1 137 3 2 3 0 3 20 10000 0	4 Infinity 9074

Problem C. Comparing answers

Input file: `c.in`
Output file: `c.out`

In a place in Southwestern Europe, the name of which I do not wish to recall, not long ago there were n cities connected by unidirectional roads, with possibly more than one road connecting a city to another one, or even to itself. As a homework assignment for your geography class, you need to calculate the number of paths of length exactly two that were between each pair of cities. However, youve been too busy celebrating the Spanish victory in the World Cup, so now you are copying the answers from your friend. You would like to make sure his answers are correct before handing in your homework.

Input

The input consists of several test cases, separated by single blank lines. Each test case begins with a line containing the integer n ($1 \leq n \leq 1\,000$). The following n lines contain n elements each, with element j of line i being the number of roads from city i to city j (a number between 0 and 10, inclusive). After that, there will be n lines. Each will contain n elements, with element j of line i being the answer from your friend for the number of length-2 paths from city i to city j; it will be an integer between 0 and 100 000 inclusive.

The test cases will finish with a line containing only the number zero (also preceded by a blank line).

Please note that this problem has large input files, so use fast I/O routines.

Output

Output For each case, your program should output a line. The content of this line should be "`YES`" if your classmate's solution to the assignment is right, and "`NO`" otherwise.

Example

c.in	c.out
3 2 0 1 1 0 3 1 1 0 5 1 2 5 3 1 3 0 4 3 2 0 1 1 0 3 1 1 0 5 1 2 5 3 2 3 0 4 0	YES NO

Problem D. Fake scoreboard

Input file: `d.in`
Output file: `d.out`

As you know, after the award ceremony of SWERC it is customary to publish a complete scoreboard with detailed information on the submissions and verdicts received. However, due to the buggy contest management system, most of the relevant data are not being recorded today. Clearly such state of affairs fails to meet the high standards we are committed to, so the judges have resolved to make up the rest of the data based on whatever shred of information left, and hope contestants are unable to tell the difference. To make our lives even simpler, we kindly ask you to provide a solution for us, or else today's scoreboard will remain forever veiled in mystery (even the fake one).

What we will know by the end of the contest is the number T of teams, the number P of problems, and the number of accepted submissions by each team. From the number and colour of balloons floating around on the premises we will also be able to infer how many teams solved each of the problems. Your task is to figure out which teams solved which problems.

Our counting skills are not up to par, so your program should be able to detect when the data we collected must be wrong (see sample input 1). Otherwise you should output a possible solution, represented as a sequence of T strings of P characters each, in the following way. Both problems and teams are assigned with distinct integers, from 1 to P and 1 to T, respectively. For team number i ($1 \le i \le T$), write the string on alphabet $\{\texttt{N}, \texttt{Y}\}$ such that its j-th ($1 \le j \le P$) character is `Y` if the team managed to get problem j accepted, and `N` otherwise. For example, the following three strings form a solution to the second sample case, where the score of each of three teams is 2, 1, 2, and the count of accepted submissions for each of three problems is 1, 2, 2:

```
NYY
NNY
YYN
```

There is at least one other solution, namely

```
NYY
NYN
YNY
```

When several solutions are possible we ask you to supply the one giving rise to the lexicographically smallest string, when each of the T rows are concatenated in order. In the example above we prefer the first solution, since `NYYNNYYYN` comes before `NYYNYNYNY` in lexicographical order. (String S comes before S_0 in lexicographical order if the first different character between the two is `N` in S but `Y` in S_0).

Input

Each input case is described by three lines. The first contains two space-separated integers T (the number of teams) and P (the number of problems), with $1 \le T, P \le 80$. The second contains T space-separated integers between 0 and 90 (inclusive), the i-th of which indicates the number of problems solved by team i. The third (and last) line has P integers between 0 and 90, the j-th of which describes the number of teams successfully solving problem j. Different input cases are separated by a blank line. The last line of the input file will be `0 0`.

Output

If the input data has a solution, print T lines of P characters each, depicting the lexicographically smallest solution as explained above. Otherwise output a single line with the word "`Impossible`". In any case a blank line should separate outputs for different test cases.

Example

d.in	d.out
2 2 1 2 1 1 3 3 2 1 2 1 2 2 3 5 3 3 1 3 1 1 0 2 0 0	Impossible NYY NNY YYN YNYNY YYNNY YNNNN

Problem E. Palindromic DNA

Input file: `e.in`
Output file: `e.out`

A DNA sequence is composed of a series of four possible nucleobases, namely Adenine, Guanine, Thymine and Cytosine; we will refer to each of these bases by their initial. For our purposes, nucleobases have an associated cyclic "order": `A` is followed by `G`, which in turn is followed by `T`, which is followed by `C`, which is followed by `A` again. State-of-the-art research in genomics has revealed the startling fact that many diseases are caused by certain subsequences of bases not forming a palindromic sequence! Your mission as a leading researcher at ICPC laboratories is to take a DNA string S and a series of subsets $P_1, \ldots, P_t$ of indices to characters (nucleobases) in S, and transform S so that each of the restrictions of the resulting string to $P_1, \ldots, P_t$ are palindromic. (The restriction of S to a subset $P = \{i_1, i_2, \ldots, i_k\}$ of indices, where $0 \leq i_1 < i_2 < \ldots < i_k < |S|$, is the string $S_{i_1} S_{i_2} \ldots S_{i_k}$). It is possible to inspect any base of S at will, but only three transformations can be applied to a base:

1. Leave it unaltered.
2. Increase it by 1 in the cyclic order of nucleobases (e.g. turn `C` into `A`).
3. Decrease it by 1 (e.g. turn `T` into `G`).

Moreover, owing to limitations of current technology, it is impossible to modify two bases in consecutive positions of the sequence. Is our goal achievable?

By way of example, consider DNA sequence `AGTAT`. Number positions starting from 0, and suppose we have the three subsets $P_1 = \{1, 4\}$, $P_2 = \{0, 1\}$ and $P_3 = \{0, 2, 4\}$. One solution is to increase the first character and decrease the last, yielding $S_0 =$ `GGTAG`. The restrictions of S_0 to P_1, P_2 and P_3 are `GG`, `GG` and `GTG`, respectively; all of them are palindromic.

One case where no solution is possible is when the string is `CATGC`, and we require the subsequences determined by positions $\{0, 3\}$ and $\{3, 4\}$ be palindromic. Here, characters 3, 0 and 4 would all need to become a `T`. But this entails modifying consecutive characters 3 and 4, which is not allowed.

Input

The first line of each test case has two integers N and T ($1 \leq N \leq 10\,000$, $1 \leq T \leq 6\,000$), the sequence length and number of subsets to consider. The next line contains the DNA sequence of length N, all of whose characters are in $\{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T}\}$. The subsets are described by the following T lines. Each line starts by "`L:`", where L ($0 \leq L \leq N$) is the number of positions in the subset, and is followed by T distinct integers between 0 and $N - 1$ in increasing order. Subsets may overlap partially or totally.

A blank line separates different test cases. The input file is terminated by a line containing `0 0`.

Output

In a single line per test case, print "`YES`" if the task is solvable and "`NO`" otherwise.

Example

e.in	e.out
5 3 AGTAT 2: 1 4 2: 0 1 3: 0 2 4 5 3 CATGC 0: 2: 0 3 2: 3 4 0 0	YES NO

Problem F. Jumping monkey

Input file: `f.in`
Output file: `f.out`

You are a hunter chasing a monkey in the forest, trying to shoot it down with your all-powerful automatic machine gun. The monkey is hiding somewhere behind the branches of one of the trees, out of your sight. You can aim at one of the trees and shoot; your bullets are capable of going through the branches and killing the monkey instantly if it happens to be in that tree. If it isn't, the monkey takes advantage of the time it takes you to reload and takes a leap into a neighbouring tree without you noticing. It never stays in the same place after a shot. You would like to find out whether there is an strategy that allows you to capture the monkey for sure, irrespective of its initial location and subsequent jumps. If so, you need to determine the shortest sequence of shots guaranteeing this.

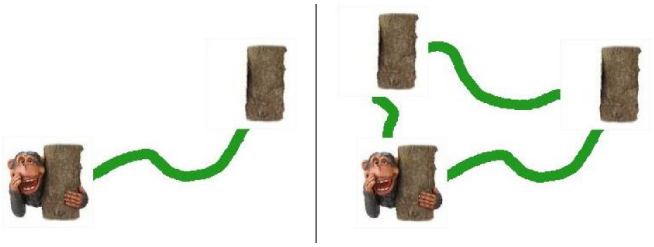


Figure 2: Example

As an example, consider the situation in which there are only two neighboring trees in the forest (left hand side of Figure 2). It is then possible to make sure you capture the monkey by shooting twice at the same tree. Your first shot succeeds if the monkey happened to be there in the first place. Otherwise, the monkey was behind the other tree and it will necessarily have moved when you shoot for the second time. However, depending on the shape of the forest it may not possible for you to ensure victory. One example of this is if there are three trees, all connected to one another (right hand side of Figure 2). No matter where you aim at, there are always two possible locations for the monkey at any given moment. (Note that here we are concerned with the worst-case scenario where the monkey may consistently guess your next target tree).

Input

The input consists of several test cases, separated by single blank lines. Each test case begins with a line containing two integers n and m ($1 \leq n \leq 21$); n is the number of trees in the forest, and m is the number of adjacency relations between trees. Each of the following m lines contains two distinct integers between 0 and $n-1$ (inclusive), the identifiers of the trees in an adjacent pair. The order of both trees within a pair carries no meaning, and no pair appears more than once. You may further assume that no tree is adjacent to itself, and there is always a path between any two trees in the forest. The test cases will finish with a line containing only two zeros (also preceded with a blank line).

Output

Print a line for each test case. The line should contain the single word "`Impossible`" if the task is impossible. Otherwise, it must contain the shortest sequence of shots with the required property, in the format $L : V_1 V_2 \ldots V_L$, where L is the length of the sequence, and $V_1, V_2, \ldots, V_L$ are space-separated integers containing the identifiers of the trees to shoot at in the right order. If several shortest sequences exist, print the lexicographically smallest one. (A sequence is smaller than another in lexicographic order if the first element on which they differ is smaller in the first one).

Example

f.in	f.out
2 1 0 1 3 3 0 1 1 2 2 0 4 3 0 1 2 3 1 3 0 0	2: 0 0 Impossible 4: 1 3 3 1

Problem G. Sensor network

Input file: `g.in`
Output file: `g.out`

In response to a request by the SWERC 2010 problem-setting team, a sensor network has just been installed in the headquarters. The organizing committee has put in the problem-setters the disproportionate fear of suffering a leak of classified information about the problems.

Nevertheless, in the rush they forgot to think about the electricity network needed to make the sensors work. Because of this, the security system is currently not working, but we need to justify the great amount of resources invested in it, so the installation must be ready before the end of the contest. Hence you are now asked to elaborate a program which will help the electrician in his duty.

Since the organizing committee spared no expense, they ordered the sensors from a prestigious company. Every sensor is handcrafted and a number is written on each of them, whose meaning is the recommended voltage that should be applied to it for correct operation. They can be used under higher voltages, with an increasing risk of failure, but never with a voltage below the recommended one. The electrician gazed open-mouthed at the sensors when they were given to him: nearly all of them had different recommended voltages! The sensors were installed in the building in such a way that each of them controls exactly two doors and every door is controlled by at least one sensor. Now is the time for the electrician to supply power to the sensors. He faces the following constraints:

- Fortunately, there is no need to activate all sensors. However, we will require that the subset of sensors chosen to be on satisfy that every door is controlled by, at least, one sensor in the subset.
- Electricity is to be supplied to one of the active sensors, and then distributed to the others with wires. In order not to waste wires, they can only be installed by connecting a pair of neighboring active sensors (by "neighbouring" we mean that some door is controlled by both of them). Since we must supply energy to every active sensor, not every subset of sensors is suitable as the chosen subset of working sensors.
- Because of the above, all of the sensors will be supplied the same voltage, which cannot be below the corresponding recommended voltages.

For simplicity, we will refer to a subset of sensors satisfying the first two constraints above as an admissible subset. The network is designed so that the set of all sensors is always admissible, but we would like to take a possibly smaller subset so as to minimize the *margin*, defined as the maximum of the differences, in absolute value, between the numbers written on each pair of sensors in the subset. (This is to keep the chances of failure to a minimum).

The electrician could not solve the task of finding the admissible subset of the set of sensors for which the margin is minimum. Therefore, the electrical installation is still missing. Today, we ask you to write a program to find out the answer, given a sensor network and the recommended voltage for each of the sensors.

Input

The input consists of several test cases, separated by single blank lines. Each test case begins with a line containing the integer n ($2 \leq n \leq 350$), the number of doors in the building. The following line contains another integer, m ($n - 1 \leq m \leq n(n-1)/2$), the number of sensors in the network. The test case finishes with m lines containing a description of each of the m sensors. The i-th of those lines contains three integers, a ($0 \leq a \leq n-1$), b ($0 \leq b \leq n-1$) and w ($1 \leq w \leq 2^15$), in that order. Integers a and b represent the pair of doors controlled by the i-th sensor, and w its recommended voltage. You can safely assume that there are no two sensors controlling the same two doors.

The input will finish with a line containing 0.

Output

For each case, your program should output a line containing the minimum margin of an admissible subset of the sensors.

Example

g.in	g.out
3 3 0 1 220 1 2 120 2 0 160 4 5 2 3 80 1 3 80 0 1 180 2 1 200 3 0 140 0	40 60

Problem H. Assembly line

Input file: `h.in`
Output file: `h.out`

The last worker in a production line at the factory of Automated Composed Machinery is worried. She knows that her job hangs in the balance unless her productivity increases. Her work consists of assembling a set of pieces in a given sequence, but the time spent on assembling pieces a and b and then c may not the same as that on assembling pieces b and c, and then assembling a with the resulting component. Only two consecutive pieces may be assembled at a time, and once they are assembled they behave as another piece in terms of the time needed for further assembly. In order to aid her, you need to find the optimal way to assemble all components. The input to your program will be a set of symbols representing (types of) pieces, and a so-called assembly table representing the time it takes to assemble them, as well as the type of the resulting component. For instance, we may have two symbols $\{a, b\}$, and the following table:

	a	b
a	$3 - b$	$5 - b$
b	$6 - a$	$2 - b$

This means, for example, that two pieces of type a and a may assembled in 3 minutes, and the result is a component of type b, in that the time required to assemble it again with another piece of, say, type a is 6 minutes, and so on. Note that the table is not symmetric, i.e. assembling b and a may be more time-consuming than a and b. For a sequence of components labelled aba, the two possible solutions are:

- $(ab)a = ba = a$ with time $t(ab) + t(ba) = 5 + 6 = 11$.
- $a(ba) = aa = b$ with time $t(ba) + t(aa) = 6 + 3 = 9$.

So the result for this case would be a piece of type b in 9 minutes (denoted $9 - b$).

Input

The input consists of several test cases. Each test case begins with a line containing a natural number k ($1 \le k \le 26$), followed by a line with k symbols (characters in $[a–z]$) separated by spaces. The following k lines contain the assembly table: the i-th line has k pairs of the form $t–r$, where t (time) is an integer between 0 and 1 000 000 inclusive, and r (result) is a symbol belonging to the preceding set. The j-th pair in the i-th line represents the time to compose pieces of types represented by the i-th and j-th symbols, along with the type of the resulting piece. After the table, a line with an integer n indicates the number of lines that follow, each line being a string of at most 200 symbols. Each of these lines is a sequence of components that need to be assembled together in the right order.

The input will finish with a line containing 0, which should not be processed.

Output

For each test case, print n lines, each with an integer time and a symbol result in the format $t–r$ as above. Each line represents the minimum time and the type of the resulting piece for the corresponding case in the input. In case of a tie among several possible results with the same minimum time, choose from among those the piece whose type letter appears first in the line that contained the k symbols at the beginning of the test case. (For example, if that line was a c b and both c and b can be obtained with minimum cost 5, print $5–c$).

There must be an empty line between the output of different test cases.

Example

h.in	h.out
2 a b 3-b 5-b 6-a 2-b 2 aba bba 2 m e 5-e 4-m 3-e 4-m 1 eme 0	9-b 8-a 7-m

Problem I. Locks and keys

Input file: `i.in`
Output file: `i.out`

A wizard is in a labyrinth where there are V rooms and $V-1$ doors connecting some pairs of rooms in both directions, in such a way that there is always a sequence of doors one can traverse to go from a room to any other room. Additionally, there are C locks and C keys of C different colors (one of each) in some of the doors and rooms of the maze, respectively; each door has at most one lock, and there is at most one key placed in each room. It should be an easy matter for the wizard to bypass the lock system, were it not for the fact that he forgot his spell book, without which his wizardry is utterly useless. The wizard is currently in room X, and he wants to get his spell book, located in room Y, without taking too long. At every step he may go to an adjacent room through one of the doors. Of course, if the door is locked, he needs to be carrying the key of the same colour as the lock (unless, of course, that door has already been unlocked). The wizard can carry only one key at a time and after picking up a key it is not possible for him to drop it somewhere in the maze in order to take it again afterwards. Once a door has been unlocked, the key is thrown away since it is no longer any use.

Given the maze and the positions of the C keys and C locks, determine how to reach Y from X, if possible. Any path whose length does not exceed $4 \cdot (C+1) \cdot V$ is acceptable.

Input

The first line of each case contains four integers: the number V of rooms in the maze ($1 \leq V \leq 1\,500$), the number C of locks ($0 \leq C < V$), and rooms X and Y numbered 0, 1, ..., $V-1$. Then comes a (possibly empty) line with C integers indicating the location of each of the keys, in order of increasing color. The next $V-1$ lines describe the maze: each contains three integers A B L, meaning that there is a door between rooms A and B which can be unlocked with the key of color L, if $0 \leq L < C$; a value of -1 for L indicates that no lock is needed. The last line has V, C, X, Y = 0, 0, 0, 0.

Output

There is one line of output per test case. If there is no solution, output "`Impossible`". Otherwise print the full path corresponding to your solution in the format $L : V_0 \ldots V_L$, where $L \leq 4(C+1)V$ is the length of a path from X to Y , and $X = V_0, V_1, \ldots, V_{L-1}, V_L = Y$ is the sequence of $L+1$ vertices visited in the right order. A single space must precede each vertex in the path; see sample output for clarification.

Example

i.in	i.out
1 0 0 0 3 1 0 2 1 0 1 -1 0 2 0 3 2 0 2 1 2 0 1 1 0 2 0 5 3 0 4 2 0 3 0 1 0 0 2 -1 1 3 1 2 4 2 0 0 0 0	0: 0 3: 0 1 0 2 Impossible 10: 0 2 0 1 0 1 3 1 0 2 4

Problem J. 3-sided dice

Input file: `j.in`
Output file: `j.out`

Just like every fall, the organizers of the Southwestern Europe Dice Simulation Contest are busy again this year. In this edition you have to simulate a 3-sided die that outputs each of three possible outcomes (which will be denoted by 1, 2 and 3) with a given probability, using three dice in a given set. The simulation is performed this way: you choose one of the given dice at random, roll it, and report its outcome. You are free to choose the probabilities of rolling each of the given dice, as long as each probability is strictly greater than zero. Before distributing the materials to the contestants, the organizers have to verify that it is actually possible to solve this task.

For example, in the first test case of the sample input you have to simulate a die that yields outcome 1, 2 and 3 with probabilities $\frac{3}{10}$, $\frac{4}{10}$, $\frac{3}{10}$. We give you three dice, and in this case the i-th of them always yields outcome i, for each $i = 1, 2, 3$. Then it is possible to simulate the given die in the following fashion: roll the first die with probability $\frac{3}{10}$, the second one with probability $\frac{4}{10}$ and the last one with probability $\frac{4}{10}$.

Input

The input consists of several test cases, separated by single blank lines. Each test case consists of four lines: the first three of them describe the three dice you are given and the last one describes the die you have to simulate. Each of the four lines contains 3 space-separated integers between 0 and 10 000 inclusive. These numbers will add up to 10 000, and represent 10 000 times the probability that rolling the die described in that line yields outcome 1, 2 and 3, respectively.

The test cases will finish with a line containing only the number zero repeated three times (also preceded with a blank line).

Output

For each case, your program should output a line with the word "`YES`" if it is feasible to produce the desired die from the given ones, and "`NO`" otherwise.

Example

j.in	j.out
0 0 10000 0 10000 0 10000 0 0 3000 4000 3000 0 0 10000 0 10000 0 3000 4000 3000 10000 0 0 0 0 0	YES NO