Problem A. Graph Coloring

Input file:	`coloring.in`
Output file:	`coloring.out`
Time limit:	0.7 seconds (*1.5 seconds for Java*)
Memory limit:	256 mebibytes

You are given an undirected graph without loops and multiple edges. You have to paint edges of the graph in the minimal amount of colors in such a way that every two edges sharing a vertex have different colors. If the minimal amount of colors is greater than three, you don't need to color the graph.

Input

The first line contains the amount of vertices n ($2 \le n \le 25$) and the amount of edges m ($1 \le m \le \frac{n(n-1)}{2}$). The next m lines contain pairs of vertex numbers a_i, b_i ($1 \le a_i, b_i \le n$): descriptions of edges of the graph. It is guaranteed that there are no loops and multiple edges in the graph.

Output

Let minimal number of colors be c. If $c > 3$, output «`NO`». Otherwise, on the first line output «`YES`», and on the second line output m integers from 1 to c: colors of edges. You should output colors of edges in the order in which edges are given in the input.

If there are several possible solutions, print any one of them.

Examples

coloring.in	coloring.out
4 5 1 2 2 3 3 1 3 4 1 4	YES 1 2 3 1 2
5 4 1 2 3 1 1 4 5 1	NO

Problem B. Decimal Fraction

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

In this problem, you have to find an optimal period for an infinite decimal fraction.

Consider an infinite decimal fraction $x_0.x_1x_2x_3...$ representing some real number x between 0 and 1, inclusive: $x = x_0 + x_1 \cdot 10^{-1} + x_2 \cdot 10^{-2} + x_3 \cdot 10^{-3} + \ldots$. Here, x_i are decimal digits from 0 to 9. In this problem, we pose no restrictions on the fraction other than given above. In particular, that means, for example, both 0.999999... and 1.000000... are valid infinite decimal fractions representing the same real number 1.

A *periodic* decimal fraction is a way to write down an infinite decimal fraction in the form $y_0.y_1y_2y_3...y_r(y_{r+1}y_{r+2}...y_s)$ for some integers $r \geq 0$ and $s > r$. This *expands* into the infinite decimal fraction $y_0.y_1y_2y_3...y_ry_{r+1}y_{r+2}...y_sy_{r+1}y_{r+2}...y_sy_{r+1}y_{r+2}...y_s...$, that is, an infinite fraction starting with $y_0.y_1y_2y_3...y_r$ and then repeating the digits $y_{r+1}y_{r+2}...y_s$ in an infinite loop. We will say that r is the length of *pre-period* and $s - r$ is the length of *period*. Not every infinite decimal fraction could be written down as a periodic one. In fact, such representation is possible if and only if the real number x represented by the fraction is rational.

We are given the first few digits of an infinite decimal fraction, the rest is just truncated (that is, no rounding takes place). Now, we want to write down some periodic decimal fraction which, when expanded, starts with the given finite part. Of such periodic decimal fractions, find one for which the sum of pre-period length and period length is minimal possible.

Input

The first line of input contains some truncated infinite decimal fraction in the form $x_0.x_1x_2x_3...x_n$ ($1 \leq n \leq 1\,000\,000$). Here, x_i are decimal digits from 0 to 9, and the real number x represented by the fraction is between 0 and 1, inclusive.

Output

Print one line containing a periodic decimal fraction in the form $y_0.y_1y_2y_3...y_r(y_{r+1}y_{r+2}...y_s)$ for some integers $r \geq 0$ and $s > r$. Here, y_i must be decimal digits from 0 to 9. The fraction must expand into an infinite decimal fraction starting with $x_0.x_1x_2x_3...x_n$ (the truncated part given in the input), and the sum of pre-period length and period length must be minimal possible. If there are several possible answers, output any one of them. It is guaranteed that at least one answer exists.

Examples

decimal.in	decimal.out
0.9999999	0.(9)
0.63573573	0.6(357)
0.123456789	0.12345(6789)

Explanations

In the first example, the periodic decimal fraction 0.(9) expands into the infinite decimal fraction 0.999... which starts with 0.9999999. Here, pre-period length is 0 and period length is 1. Other answers such as 0.9(99) or even 0.99999998(7) also expand into a fraction which starts with 0.9999999, but they are not optimal. Note that, although 0.9999999... = 1 as a real number, the answer 1.(0) is **not** valid since its expansion does not start with 0.9999999.

In the second example, the answer 0.6(357) expands into 0.6357357357357.... Here, pre-period length is 1 and period length is 3. The first few digits correspond to the given truncated fraction.

In the third example, possible answers are 0.(123456789), 0.1(23456789), ..., 0.12345678(9). Remember that pre-period length must be non-negative, and period length must be positive.

Problem C. Teams of Equal Power

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

In this problem, you have to divide mages into two teams so that the powers of the teams are equal.

There are n mages which must be divided into two teams for a magic game. An integer representing the mage's power is associated with each mage. The power of a team of mages is the sum of powers of all mages in the team plus the maximum power of a mage in the team. How can we divide the mages into two teams of equal power?

Input

The first line of input contains a single integer n: the number of mages ($1 \le n \le 500$). The second line contains n integers a_1, a_2, ..., a_n: the powers of mages ($1 \le a_i \le 500$).

Output

If the mages cannot be divided into two teams of equal power, print the integer -1. Otherwise, on the first line of output, print the integer equal to the power of each team, and on the next four lines, print the teams' composition.

A description of a team consists of two lines. The first of these lines contains one integer k: the number of mages in the team. The second line contains k integers: the numbers of mages in the team. The numbers of mages can be printed in any order.

Each mage must be mentioned exactly once in exactly one of the two teams. The mages are numbered from one in the order their powers are given in the input.

If there are several possible divisions, output any one of them.

Examples

equal-power.in	equal-power.out
5 1 2 3 4 5	12 3 1 3 4 2 2 5
4 8 1 7 6	-1
6 8 6 5 3 8 4	24 4 4 6 3 2 2 1 5

Explanations

In the first example, the power of the first team is calculated as $(1 + 3 + 4) + 4$, and the power of the second team as $(2 + 5) + 5$. Thus the power of each team is equal to 12.

In the second example, it is absolutely impossible to divide the given four mages into two teams of equal power.

In the third example, there is another possible solution: take first, second and fourth mages in the first team to get $(8 + 6 + 3) + 8 = 25$ and take the remaining third, fifth and sixth mages in the second team to get $(5 + 8 + 4) + 8 = 25$.

Problem D. Hexagon

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

Consider the plane and the following lines on it:

1. $y = \frac{\sqrt{3}}{2} \cdot i$,
2. $y = \sqrt{3} \cdot x + \sqrt{3} \cdot i$,
3. $y = -\sqrt{3} \cdot x + \sqrt{3} \cdot i$

for $i = \ldots, -1, 0, 1, 2, 3, \ldots$. The lines divide the plane into regular triangles with sides of length 1.

A hexagon with sides A, B, C, D, E, F is placed on the plane in such a way that all its sides are the segments of the lines given, and all its vertices lie on the points of intersection of the lines. All the interior angles of the hexagon are equal to 120°.

Let's call a *diamond* a figure consisting of two regular triangles with sides of length 1 sharing one side.

Find the number of tilings of a given hexagon by diamonds modulo $p = 2\,147\,483\,647$.

Input

The first line of input contains six integers A, B, C, D, E, F ($1 \le A, B, C, D, E, F \le 15$): hexagon sides in clockwise order. It is garanteed that given sides define the hexagon consistent with the problem statement.

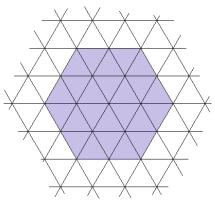
Output

Output one integer: the number of ways to divide the given hexagon into diamonds modulo $p = 2\,147\,483\,647$.

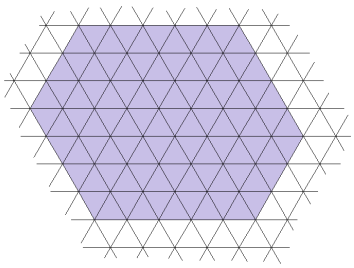
Examples

hexagon.in	hexagon.out
2 2 2 2 2 2	20
3 5 4 3 5 4	116424

Explanations



The hexagon from the first example can be divided into diamonds in 20 different ways modulo p.



The hexagon from the second example can be divided into diamonds in 116 424 different ways modulo p.

Problem E. Maximal Matching

Input file:	matching.in
Output file:	matching.out
Time limit:	3.5 seconds (*5 seconds for Java*)
Memory limit:	256 mebibytes

You are given a bipartite graph. Each vertex has some weight. The weight of an edge is the sum of weights of its ends. The weight of a matching is the sum of weights of edges of the matching. You have to find a matching with maximal weight. Note that the matching may contain any number of edges, the only constraint is that its weight should be maximal possible.

Recall that a matching in a bipartite graph is a collection of edges of the graph such that no two edges share a common vertex.

Input

The first line contains the sizes of parts of the bipartite graph n and m ($1 \le n, m \le 5\,000$) and the number of edges e ($0 \le e \le 10\,000$). The second line contains n integer numbers from 0 to 10 000 — weights of vertices of the first part. The third line contains m integer numbers from 0 to 10 000 — weights of vertices of the second part. Next e lines contain the edges of the graph. Each edge is described by a pair of integer numbers a_i b_i, where $1 \le a_i \le n$ is the number of vertex in the first part and $1 \le b_i \le m$ is the number of vertex in the second part.

Output

On the first line output w — the maximal weight of matching. On the second line output k — the number of edges in a matching with maximal weight. Next line should contain k different numbers from 1 to e — numbers of edges from the matching. If there are several matchings with maximal weight, you may output any one of them.

Examples

matching.in	matching.out
4 3 3 2 0 9 9 1 0 9 1 2 2 1 1 1	3 1 3
3 2 4 1 2 3 1 2 1 1 2 1 2 2 3 2	8 2 4 2

Problem F. Right Turn Only

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

In this problem, you have to find an optimal drive route in a city where driving rules only allow to turn right.

We are driving in a city. The map of the city can be viewed as an infinite rectangular grid: streets correspond to lines $y = c$ for every integer c and avenues correspond to lines $x = c$ for every integer c.

The traffic is heavy, so the driving rules in the city are rather strict. When a car arrives at a crossing, it can either stop, drive straight, or turn right, changing the direction by 90 degrees. So, for example, if a car arrives at crossing $(0, 1)$ from crossing $(0, 0)$ and is going to drive further, it can drive to $(0, 2)$ or turn right and drive to $(1, 1)$, but there is no way to immediately drive towards $(-1, 1)$.

We start at crossing $(0, 0)$, ready to drive towards crossing $(0, 1)$. Find the minimal distance we have to travel to reach crossing (x, y). The direction from which we arrive to (x, y) is not important.

Input

The first line of input contains a single integer t: the number of test cases ($1 \le t \le 10\,000$). Each of the next t lines describes one test case. Each description consists of two integers x and y separated by a space: the coordinates of the destination ($|x|, |y| \le 10^9$).

Output

For each test case, print one line containing one integer: the minimal travel distance to the destination according to the rules. Output the answers in the order the test cases are given in the input.

Example

right-turn-only.in	right-turn-only.out
0 0 3 2 0 -1	0 5 5

Explanation

In the first test case, we are already at the destination.

In the second test case, we can drive from $(0, 0)$ to $(0, 2)$, turn right, and then drive from $(0, 2)$ to $(3, 2)$.

In the third test case, we can drive as follows:

$$(0, 0) \to (0, 1) \xrightarrow{\text{turn}} (1, 1) \xrightarrow{\text{turn}} (1, 0) \to (1, -1) \xrightarrow{\text{turn}} (0, -1).$$

Note that we can not turn before we move from $(0, 0)$, and later, we can not turn twice in a row while we are at $(0, 1)$.

Problem G. Similar Strings

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

You have n different strings of the same length k. All strings consist of characters with ASCII codes from 33 to 126. The distance between two strings of equal length is the amount of positions at which the strings differ. For each of n strings you have to find the nearest one among n given strings (but not the same one).

Input

On the first line there are two integers n ($2 \le n \le 50\,000$) and k ($1 \le k \le 6$). Each of next n lines contains exactly k characters.

Output

Output n lines, on each line print two numbers: i-th line should contain distance from i-th string to the nearest one, and the number of the nearest string. Strings are numbered by integers from 1 to n in the order they are given in the input. If there are several strings with minimal distance, you may output any one of them.

Examples

similar.in	similar.out
5 3 aaa aab ccc aac cab	1 2 1 1 2 4 1 2 1 2
2 6 abcdef fabcde	6 2 6 1

Problem H. Traffic Lights

Input file:	`traffic.in`
Output file:	`traffic.out`
Time limit:	2 seconds (*4 seconds for Java*)
Memory limit:	256 mebibytes

The city Nsk looks like a rectangular table of $n \times m$ quarters. If you look at the map of the city, you will see that there are exactly $n+1$ vertical streets and exactly $m+1$ horizontal streets. All quarters are squares of the same size. All streets have the same width.

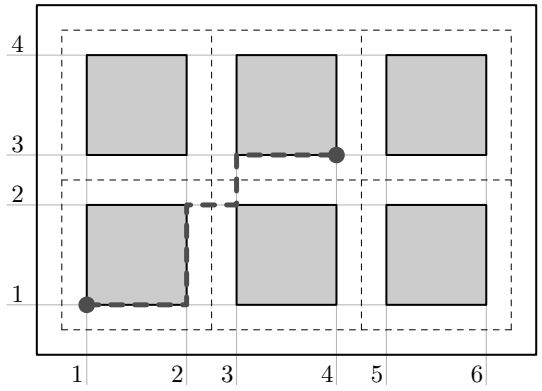
Vasya is a pedestrian. Pedestrians may cross streets only in special places with traffic lights. In Nsk, traffic lights stand on all crossroads. To cross a road, Vasya should primarily come to a corner of a quarter. He may cross a road only strictly orthogonally to the road. After crossing the road, he will be exactly in the corner of a quarter which is opposite to the original one. Vasya may move along the borders of quarters. Vasya can not move inside a quarter: he does not know what awaits him there and does not want to risk. So, going through the city, in each moment Vasya may do one of two actions:

- If Vasya stays next to a traffic light, and that traffic light is green, Vasya may cross the street (even if the traffic light changes its color to red before Vasya will finish crossing the street). It takes exactly s seconds to cross the street. If the traffic light is red, Vasya may wait for a green light.
- Vasya may move along a border of a quarter. Vasya passes the distance between two adjacent corners of a quarter in d seconds.

Vasya lives in a house located exactly in the corner of a quarter of Nsk. Vasya is going to visit his friend Kolya who lives in another house which is also located in the corner of a quarter of Nsk. Moving down the stairs, Vasya thought: and **what is the expected amount of time he spends moving to Kolya's house, if Vasya will move optimally?**

Traffic lights are located on crossroads. Traffic lights in the center of the city regulate 4 crosswalks. Traffic lights on the border of the city regulate one crosswalk. A traffic light which regulates 4 crosswalks in each moment of time is either green for both horizontal crosswalks and red for both vertical crosswalks or vice versa. Vasya is not so young and knows some interesting facts about traffic lights. Firstly, period of each traffic light is $2t$. Secondly, each traffic light is green for t seconds, then it is red for t seconds, then it is again green for t seconds and so on. But for each traffic light, the shifts over the period (time moments modulo $2t$ when traffic lights become green) are independent real uniformly distributed random values from 0 to $2t$. When Vasya moves down the stairs and thinks about the expected value, he doesn't know any shifts yet. Traffic lights in Nsk are rather modern, each traffic light has an infinitely precise timer which shows after what time the traffic light will change its color. Vasya is an experienced programmer, so his eyesight allows to see digits on the timer only in the moment when Vasya already stays next to the crosswalk which the traffic light regulates. Also it is known that $d > t + s$ and $t > s$. You have to help Vasya to calculate the expected amount of time he spends moving to Kolya's house assuming that Vasya will move optimally.

The following picture describes in detail the city and possible Vasya's path. Circles on the picture are Vasya's house and Kolya's house. Vasya thinks all the time, so after reaching some traffic light and receiving new information (shift over the period of the traffic light), he may decide how to proceed.



The picture describes the fourth sample of the statement, $n = 3$, $m = 2$.

Input

The first line contains five integer numbers n, m, d, t and s ($2 \le n, m \le 1000$, $0 < t < d \le 1000$, $0 < s < t$, $s + t < d$). On the picture, you may see not only $n + 1$ and $m + 1$ streets but also $2n$ and $2m$ numbered *pedestrian streets*. Each corner of each quarter is uniquely described by the intersection of two *pedestrian streets*. The second line contains the position of the corner of quarter where Vasya's house is situated. The position is described by a pair of numbers of *pedestrian streets*: x, y ($1 \le x \le 2n$, $1 \le y \le 2m$). The third line contains the position of the corner of quarter where Kolya's house is situated. The position is given in the same format.

Output

Print one real number: the expected amount of time Vasya spends moving from his house to Kolya's house. The answer will be accepted if absolute or relative error will be no more than 10^{-9}.

Examples

traffic.in	traffic.out
1 3 100 10 1 1 1 1 6	307.0
2 2 100 10 1 3 3 2 2	3.9375
2 2 100 10 1 2 1 3 4	203.4310302734
3 2 100 10 1 1 1 4 3	303.2595461871

Problem I. Triple Connections

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

In this problem, you are given the record made by one robot, and you have to output the record made by the other robot at the same moment of time.

Let us fix an integer n and mark $3n$ points on a circle: vertices of a regular $3n$-gon. We will then inscribe non-degenerate triangles with vertices in marked points into the circle. A *triple connection* is a set of n triangles such that no two of them share a common point (vertices cannot be shared, too).

We will use the following notation for triple connections. First, let us number the marked points with integers from 1 to $3n$ in some order.

A triangle is denoted as a triple of numbers of its vertices in ascending order. Triangles are compared by their notations as sequences of three numbers.

A notation of a triple connection is a sequence of $3n$ numbers which results from writing one after another the notations of n triangles in the connection. Here, triangles are arranged in ascending order. For convenience, we will break this sequence into n lines, one line per triangle. Triple connections are compared by their notations as sequences of numbers. Two triple connections are considered different if and only if their notations are different.

Two robots drew themselves copies of the circle with $3n$ marked points, one circle per robot. Then each of them numbered the points on his circle in his own way (however, they may have numbered the points in the same way). After that, robots simultaneously started to write down all triple connections with the same speed of one connection per second. Each of the robots writes down triple connections on his circle in lexicographical order.

Given the integer n, both numberings and the notation of a triple connection which was just written down by the first robot, find the notation of a triple connection which was just written down by the second robot.

Recall that a sequence $a_1, a_2, \ldots, a_p$ is *lexicographically smaller* than a sequence $b_1, b_2, \ldots, b_q$ if:

- either $p = 0$ and $q > 0$,
- or $a_1 < b_1$,
- or $a_1 = b_1$ and the sequence $a_2, \ldots, a_p$ is lexicographically smaller than the sequence $b_2, \ldots, b_q$.

Input

The first line of input contains a single integer n ($1 \le n \le 25$).

The second line contains $3n$ integers from 1 to $3n$, each of them exactly once: the numbering of points on the circle which is used by the first robot. The numbers of points are given in the order of traversing the circle in clockwise direction starting from some point.

The next n lines contain three integers each and together constitute the notation of a triple connection which was just written down by the first robot. These n lines also contain each of the integers from 1 to $3n$ exactly once. It is guaranteed that these lines give a valid notation of a triple connection with the numbering of points used by the first robot.

Finally, the last line of input contains $3n$ integers from 1 to $3n$, each of them exactly once: the numbering of points on the circle which is used by the second robot. The numbers of points are once again given in the order of traversing the circle in clockwise direction starting from some point.

Output

Print n lines with three integers on each line: the notation of a triple connection which was just written down by the second robot. These n lines should contain each integer from 1 to $3n$ exactly once.

Examples

triconnect.in	triconnect.out
2 1 2 3 4 5 6 1 2 6 3 4 5 2 4 6 1 3 5	1 3 6 2 4 5
2 1 2 3 4 5 6 1 2 3 4 5 6 3 4 5 6 1 2	1 2 3 4 5 6

Explanations

In both examples, $n = 2$, so there are six marked points on the circle. With any numbering of points, we can construct only three different triple connections when $n = 2$. We should pick some three sequential points on the circle and construct the first triangle on these points and the second one on the other three points. There are six ways to pick three sequential points, and they can be combined into pairs so that the ways of one pair differ only in which triangle was constructed first — and which was the second one.

In the first example, the points on the first robot's circle are numbered by consecutive natural numbers in the order of clockwise traversal. The first robot writes down triple connections in the following order:

1 2 3 4 5 6	1 2 6 3 4 5	1 5 6 2 3 4

The numbering of points is different for the second robot. He writes down triple connections in the following order:

1 3 5 2 4 6	1 3 6 2 4 5	1 4 6 2 3 5

In the second example, the numbering of points for the second robot can be obtained by shifting the numbering for the first robot in a circular way. This results in records of robots being equal at every possible moment of time.