Problem A. Soda Surpler

Input file:	`soda.in`
Output file:	`soda.out`
Time limit:	2 seconds
Memory limit:	64 Mebibytes

Tim is an absolutely obsessive soda drinker, he simply cannot get enough. Most annoyingly though, he almost never has any money, so his only obvious legal way to obtain more soda is to take the money he gets when he recycles empty soda bottles to buy new ones. In addition to the empty bottles resulting from his own consumption he sometimes find empty bottles in the street. One day he was extra thirsty, so he actually drank sodas until he couldn't afford a new one.

Input

Three non-negative integers e, f, c, where $e < 1\,000$ equals the number of empty soda bottles in Tim's possession at the start of the day, $f < 1\,000$ the number of empty soda bottles found during the day, and $1 < c < 2\,000$ the number of empty bottles required to buy a new soda.

Output

How many sodas did Tim drink on his extra thirsty day?

Example

soda.in	soda.out
9 0 3	4
5 5 2	9

Problem B. Money Matters

Input file:	`money.in`
Output file:	`money.out`
Time limit:	2 seconds
Memory limit:	64 Mebibytes

Our sad tale begins with a tight clique of friends. Together they went on a trip to the picturesque country of Molvania. During their stay, various events which are too horrible to mention occurred. The net result was that the last evening of the trip ended with a momentous exchange of "I never want to see you again!"s. A quick calculation tells you it may have been said almost 50 million times!

Back home in Scandinavia, our group of ex-friends realize that they haven't split the costs incurred during the trip evenly. Some people may be out several thousand crowns. Settling the debts turns out to be a bit more problematic than it ought to be, as many in the group no longer wish to speak to one another, and even less to give each other money.

Naturally, you want to help out, so you ask each person to tell you how much money she owes or is owed, and whom she is still friends with. Given this information, you're sure you can figure out if it's possible for everyone to get even, and with money only being given between persons who are still friends.

Input

The first line contains two integers, n ($2 \le n \le 10\,000$), and m ($0 \le m \le 50\,000$), the number of friends and the number of remaining friendships. Then n lines follow, each containing an integer o ($-10\,000 \le o \le 10\,000$) indicating how much each person owes (or is owed if $o < 0$). The sum of these values is zero. After this comes m lines giving the remaining friendships, each line containing two integers x, y ($0 \le x < y \le n-1$) indicating that persons x and y are still friends.

Output

Your output should consist of a single line saying "`POSSIBLE`" or "`IMPOSSIBLE`".

Example

money.in	money.out

5 3
100
-75
-25
-42
42
0 1
1 2
3 4
	POSSIBLE

4 2
15
20
-10
-25
0 2
1 3
	IMPOSSIBLE

Problem C. Allergy Test

Input file:	`allergy.in`
Output file:	`allergy.out`
Time limit:	2 seconds
Memory limit:	64 Mebibytes

A test for allergy is conducted over the course of several days, and consists of exposing you to different substances (so called allergens). The goal is to decide exactly which of the allergens you are allergic to. Each allergen has a live duration D measured in whole days, indicating exactly how many days you will suffer from an allergic reaction if you are allergic to that particular substance. An allergic reaction starts to show almost immediately after you have been exposed to an allergen which you are allergic to. The test scheme has two action points per day:

1. At 8 o'clock each morning, at most one of the allergens is applied to your body.
2. At 8 o'clock each evening, you are examined for allergic reactions.

Thus an allergen with live duration D will act exactly D allergic reaction examinations.

Of course, if you have two or more active allergens in your body at the time of an observed reaction, you cannot tell from that information only, which of the substances you are allergic to.

You want to find the shortest possible test scheme given the durations of the allergens you want to test. Furthermore, to allow simple large scale application the test scheme must be non-adaptive, i.e. the scheme should be fixed in advance. Thus you may not choose when to apply an allergen based on the outcome of previous allergic reaction examinations.

Input

The first line of the input contains a single integer k ($1 \le k \le 20$) specifying the number of allergens being tested for. Then follow k lines each containing an integer D ($1 \le D \le 7$) specifying the live duration of each allergen.

Output

The number of days of the shortest conclusive non-adaptive test scheme. *A scheme ends the morning when you no longer have active allergens in your body, thus a test scheme for a single allergen with live duration D takes D days.*

Example

allergy.in	allergy.out
3 2 2 2	5

allergy.in	allergy.out
5 1 4 2 5 2	10

Problem D. Rain Fall

Input file:	`rain.in`
Output file:	`rain.out`
Time limit:	2 seconds
Memory limit:	64 Mebibytes

Rainfall is measured in millimeters. The rain is collected in a vertical transparent tube with millimeter markings, and once the rain has stopped falling, one can check the height of the water in the tube.

In our problem, the tube unfortunately has a leak at height L millimeters (mm). If the water level is above the leak then water drains from the tube at a rate of K millimeters per hour (mm/h). We want to figure out how much rain fell during a particular rainfall. *We assume that the tube is high enough that it does not overflow. We also assume that rain falls at an (unknown) uniform rate during a rainfall, and that water does not evaporate from the tube. The height of the leak itself is also negligible.*

Input

The input is a line with five positive numbers: L K T_1 T_2 H where

L is where the leak is (mm)

K is the rate at which water leaks (mm/h)

T_1 is the duration of the rainfall (h)

T_2 is the time between the end of the rainfall and the observation of the water level (h)

H is the water level in the tube when we observe it (mm)

Each number is at least 0.01 and at most 1 000.00, and each is given with two decimals.

Output

One line with two floating point numbers F_1 F_2 where F_1 is the smallest rainfall in millimeters that would result in the given observation, and F_2 is the largest rainfall in millimeters that would result in the given observation. Values with either absolute or relative error smaller than 10^{-6} are acceptable.

Example

rain.in	rain.out
80.00 0.50 2.00 1.50 80.00	80.000000 80.759403
150.00 1.00 100.00 150.00 100.00	100.000000 100.000000

Problem E. Speedy Escape

Input file:	`escape.in`
Output file:	`escape.out`
Time limit:	2 seconds
Memory limit:	64 Mebibytes

The Newton brothers are planning to rob a bank in the city of Alviso and want to figure out a way to escape the city's only police car. They know that their car is faster than the police car so if they could just reach one of the highways exiting the city they will be able to speed away from the police.

The police car has a maximum speed of 160 km/h. Luckily, the brothers know where the police car will start (it's parked at the police station). To be on the safe side they assume that the police car will start moving as soon as they leave the bank and start their car (this is when the alarm goes off).

The brothers want to find a fixed route that ensures that they are able to leave the city no matter what route the police car take and at what speed it drives. However, since the brothers are not very confident drivers they don't want to drive faster than necessary. Luckily they have recently invested in a new hi-tech in-car police escape system that you have constructed. This system will tell them what the minimal top speed needed to escape is (and probably other useful things like what route to take).

Let's turn the clock back a bit to the time when you were constructing the escape system and focused on finding the minimal required speed. Can you get it right?

You may treat all roads as infinitesimally narrow and both cars as point objects. If the brothers ever end up at the same point (on any road or intersection) at the same time as the police car they will be caught and by Murphy's law if there is any possibility of this happening it will happen. The two cars start simultaneously and can accelerate/decelerate instantaneously at any time to any speed below or equal to its maximum speed. They can also change roads at intersections or direction anywhere on a road instantaneously no matter what speed they are traveling at.

Input

The first line of the input consists of three integers n, m and e, where $2 \le n \le 100$ describe the number of intersections, $1 \le m \le 5\,000$ describes the number of roads in the city and $1 \le e \le n$ describes the number of highway exits. Then follow m lines, each consisting of three integers a, b, l such that $1 \le a < b \le n$ and $1 \le l \le 100$ describing a road of length l hundred meters from intersection a to intersection b. Then follows a line of e integers, each one a number in $1, \ldots, n$ describing which intersections are connected to highway exits. Finally there is a line with two integers b and p ($1 \le b; p \le n$ and $b \ne p$) describing the intersections where the brothers and the police cars start, respectively.

It will always be possible to travel from any intersection to any other intersection. Roads are only connected at intersection points (although they may cross using bridges or tunnels at others points). Roads can be used in both directions but there cannot be more than one road between two intersections.

Output

The minimal speed in km/h required to escape or the word "`IMPOSSIBLE`" if it is impossible. In the first case any answer with either absolute or relative error smaller than 10^{-6} is acceptable.

Example

escape.in	escape.out

3 2 1
1 2 7
2 3 8
1
3 2
	IMPOSSIBLE

3 2 1
1 2 7
2 3 8
1
2 3
	74.6666666667

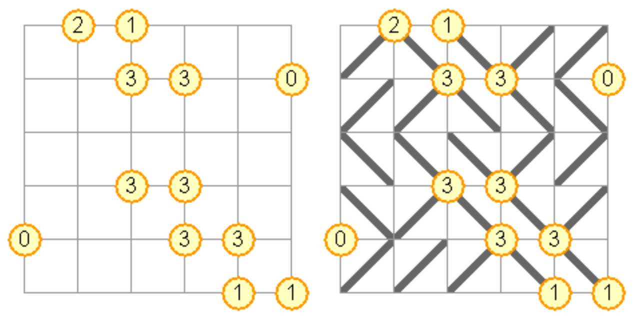
4 4 2
1 4 1
1 3 4
3 4 10
2 3 30
1 2
3 4
	137.142857143

Problem F. Gokigen Naname

Input file:	`gokigen.in`
Output file:	`gokigen.out`
Time limit:	2 seconds
Memory limit:	64 Mebibytes

Gokigen Naname is a Japanese puzzle game played on a square grid in which numbers in circles appear at some of the intersections on the grid.

The objective is to draw diagonal lines in each cell of the grid, such that the number in each circle equals the number of lines extending from that circle. Additionally, it is forbidden for the diagonal lines to form an enclosed loop.



The first figure shows the start position of a puzzle. The second figure shows the solution to the same puzzle. A Gokigen Naname puzzle always has exactly one solution.

Input

The first line of the input contains a single integer n ($2 \leq n \leq 7$), the number of cells along each of the sides in the square grid. Then follow $n+1$ lines containing the contents of the intersections of the grid cells. Each such line will contain a string of $n+1$ characters, either a digit between 0 and 4, inclusive, or a period (“.”) indicating that there is no number at this intersection (arbitrarily many lines may connect to it).

Output

The output should contain n lines, each line containing exactly n characters. Each character should either be a slash or a backslash, denoting how the corresponding grid cell is filled.

Example

gokigen.in	gokigen.out

`3`
`1.1.`
`...0`
`.3..`
`..2.`
	`\//`
	`\\\`
	`/\/`

gokigen.in	gokigen.out

`5`
`.21...`
`..33.0`
`......`
`..33..`
`0..33.`
`....11`
	`/\\//`
	`//\\\`
	`\\\//`
	`\/\\/`
	`///\\`

Problem G. Flight Planning

Input file:	`flight.in`
Output file:	`flight.out`
Time limit:	2 seconds
Memory limit:	64 Mebibytes

The airline company NCPC Airways has flights to and from n cities, numbered from 1 to n, around the entire world. However, they only have $n-1$ different flights (operating in both directions), so in order to travel between any two cities you might have to take several flights. In fact, since the management has made sure that it's possible to travel between any pair of cities, there is exactly one set of flights a passenger have to take in order to travel between two cities (assuming you want to use the same airline).

Recently many of NCPC Airways frequent flyers have complained that they have had to change flights too often to get to their final destination. Since NCPC Airways doesn't want to loose their customers to other airline companies, but still keep the nice property of their flights, they have decided to cancel one of their current flights and replace it with another flight. Help the company by writing a program which finds the best flight to cancel and the best new flight to add so that the maximum number of flight changes a passenger might have to make when travelling between any pair of cities in which NCPC Airways operates is minimized.

The input will be constructed so that it is always possible to improve the maximum number of flight changes needed.

Input

The first line in the input contains the integer n ($4 \leq n \leq 2\,500$), the number of cities NCPC Airways operates in. Then follow $n-1$ lines specifying the flights. Each flight is given as a pair of cities a and b ($1 \leq a, b \leq n$).

Output

The output should consist of three lines. The first line should contain an integer, the minimum number of flights needed to take when travelling between any pair of cities after changing one of the flights. The second line should contain two integers, specifying the two cities between which the flight should be canceled. The third line should contain two integers, specifying the two cities where a new flight should be added.

If there are more than one optimal solution, any one of them will be accepted.

Example

flight.in	flight.out

4
1 2
2 3
3 4
	2
	3 4
	2 4

14
1 2
1 8
2 3
2 4
8 9
8 10
8 11
4 5
4 6
4 7
10 12
10 13
13 14
	5
	1 8
	2 10

Problem H. Beacons

Input file:	`beacons.in`
Output file:	`beacons.out`
Time limit:	10 seconds
Memory limit:	64 Mebibytes

In ancient times, communication was not as swift as it is today. When a kingdom was at war, it could take months to muster all the armed forces. But by using fire-lit beacons at strategic locations, it was still possible to quickly send emergency signals.

When the first beacon is lit, all other beacons within sight from it are also lit. All beacons within sight of these are then lit, and so on until all beacons are lit—assuming of course that all beacons are within sight of each other, directly or indirectly. If they are not, the dire news must be carried by riders between some beacons.

Given the location of all beacons in the kingdom as well as the location and size of all mountain peaks, write a program that determines how many messages must be sent by riders in order for all beacons to be lit when an enemy threatens the country.

For simplicity, we model the country in the following way: a beacon is represented as a point in the xy-plane and a mountain peak is represented as a circle. Two beacons are considered to be within sight of each other if no mountain peak blocks the straight line between the two beacons.

The input will be constructed so that the straight line between any pair of beacons will not touch the circumference of a mountain peak, unless it passes through the interior of another mountain peak. Mountain peaks will not overlap or touch, nor will any beacon be on a mountain peak or on its circumference.

Input

The first line in the input contains two integers n ($1 \le n \le 1\,000$) and m ($0 \le m \le 1\,000$) the number of beacons and the number of mountain peaks, respectively. Then follow n lines specifying the locations of the beacons. The location of each beacon is given as a pair of integers x and y ($0 \le x; y \le 10\,000$). Then follow m lines describing the mountain peaks. Each mountain peak is given as a pair of integers x and y ($0 \le x; y \le 10\,000$) specifying the location of the peak and a radius r ($1 \le r \le 5\,000$).

Output

The output should be a single integer: the number of messages that must be carried by riders for all beacons to be lit.

Example

beacons.in	beacons.out

6 3
1 8
5 4
7 7
9 2
16 6
17 10
4 7 2
6 3 1
12 6 3
	2

4 4
0 4
8 4
4 0
4 8
2 2 1
6 2 1
2 6 1
6 6 1
	1

Problem I. Playfair Cipher

Input file:	`playfair.in`
Output file:	`playfair.out`
Time limit:	2 seconds
Memory limit:	64 Mebibytes

The Playfair cipher is a manual symmetric encryption technique and was the first digraph substitution cipher. The scheme was invented in 1854 by Charles Wheatstone, but bears the name of Lord Playfair who promoted the use of the cipher.

The Playfair cipher uses a 5 by 5 table containing each letter in the English alphabet exactly once (except "`Q`" which is missing). The table constitutes the encryption key. To more easily remember the table, it is typically generated from a key phrase. First fill in the spaces in an empty table with the letters of the key phrase (dropping spaces and duplicate letters), then fill the remaining spaces with the rest of the letters of the alphabet in order. The key phrase is written in the top rows of the table, from left to right. For instance, if the key phrase is "`playfair example`", the encryption key becomes

P	L	A	Y	F
I	R	E	X	M
B	C	D	G	H
J	K	N	O	S
T	U	V	W	Z

To encrypt a message, one would remove all spaces and then break the message into digraphs (groups of 2 letters) such that, for example, "`Hello World`" becomes "`HE LL OW OR LD`". Then map them out on the key table, and apply the rule below that matches the letter combination:

- If both letters are the same (or only one letter is left), add an "`X`" after the first letter. Encrypt the new pair and continue (note that this changes all the remaining digraphs).
- If the letters appear on the same row of your table, replace them with the letters to their immediate right respectively (wrapping around to the left side of the row if a letter in the original pair was on the right side of the row). With the table above, the digraph "`CH`" would be encrypted "`DB`".
- If the letters appear on the same column of your table, replace them with the letters immediately below respectively (wrapping around to the top side of the column if a letter in the original pair was on the bottom side of the column). With the table above, the digraph "`VA`" would be encrypted "`AE`".
- If the letters are not on the same row or column, replace them with the letters on the same row respectively but at the other pair of corners of the rectangle defined by the original pair. The order is important—the first letter of the encrypted pair is the one that lies on the same row as the first letter of the plaintext pair. With the table above, the digraph "`KM`" would be encrypted "`SR`".

Write a program that reads a key phrase and a plaintext to encrypt, and outputs the encrypted text.

The text to encrypt will not contain two "`x`"s following each other, or an "`x`" as the last character, as this might cause the first rule above to repeat itself indefinitely.

Input

The input contains two lines. The first line contains the key phrase. The second line contains the text to encrypt. Each line will contain between 1 and 1 000 characters, inclusive. Each character will be a lower case English letter, "`a`"–"`z`" (except "`q`"), or a space character. Neither line will start or end with a space.

Output

The output should be a single line containing the encrypted text, in upper case. There should be no spaces in the output.

Example

playfair.in	playfair.out

`playfair example`
`hide the gold in the tree stump`
	`BMNDZBXDKYBEJVDMUIXMMNUVIF`

`the magic key`
`i love programming competition`
	`YDVHCWSPKNTAHKUBIPERMHGHDVRU`

Problem J. Code Permutations

Input file:	`permutations.in`
Output file:	`permutations.out`
Time limit:	2 seconds
Memory limit:	64 Mebibytes

You are soon to graduate from the mathemagician school of Hagworts, and you're quite content with yourself; all the hard work and elbowing has finally paid off. Being successful at most of your endeavors you would normally end up a herald of sound reasoning and good mathemagics. You, however, are different; not only have you spent your young years secretly hacking away at com- puters, writing code to do your assigned routine homework for you, but of late you have started planning how to cram all your mathemagical skills into a computer to completely eliminate the need for mathemagicians! To show the others just how great a visionary you are, you plan to make your graduation ceremony something they will never forget.

To do this you need to break into the safe of your arch-nemesis, Hairy Peter. The safe is locked by a code mechanism: All natural numbers from 1 to N need to be typed in in the correct order, set by Hairy Peter. Fortunately you know that Hairy, being a good mathemagician, has a certain weakness; he has a rather unhealthy obsession with the number K. (For instance he always has to introduce himself K times whenever he meets new people, making him quite annoying to be around.) Thus you are certain that his code, when viewed as a permutation of the N first naturals, has order exactly K. (i.e. K is the smallest positive number such that if you K times replace $x \in 1, \ldots, N$ with the position of x in Hairy's code, you end up with the x you started with, for all x. Thus e.g. the order of the permutation corresponding to the code 2 3 1 is 3, as $1 \to 3 \to 2 \to 1$ and $2 \to 1 \to 3 \to 2$ and $3 \to 2 \to 1 \to 3$.) While this does not help you directly, it greatly reduces the number of code inputs you may have to try before you find the correct one. "How many?" is the question you are pondering right now. You must know the exact number, lest you risk preparing too little time for cracking the safe.

Now you also have a certain mathemagical weakness—arguably a bit worse than Hairy Peter's: Because of your dark scheme to program mathemagical computers, you insist there are no numbers greater than what can be represented by a signed 32-bit integer, namely the prime $P = 2^{31} - 1$. Of course there must be nothing your computers cannot count. In fact you hate this upper limit P so intensely that you have decided to make a new mathemagics where P equals 0. Ha, take that! (Well, you are quite aware that you are really just counting modulo P, but it will have to successfulce until you find better ways of punishing P.) In fact this turns out to be quite an advantage for you! For instance, if the number of code permutations you have to check turns out to be 2^{31}, there will actually be just one permutation for you to check, as 2^{31} mod $P = 1$. (Or at least you think so...) That's just magnificent!

Input

The input consists of two integers N ($1 \leq N \leq 100$) and K ($1 \leq K \leq 2^{31} - 1$) respectively.

Output

The number of permutations of N elements of order K, modulo $2^{31} - 1$.

Example

permutations.in	permutations.out
3 2	3
6 6	240
15 12	1789014075