A. Is your horseshoe on the other hoof?

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Valera the Horse is going to the party with friends. He has been following the fashion trends for a while, and he knows that it is very popular to wear all horseshoes of different color. Valera has got four horseshoes left from the last year, but maybe some of them have the same color. In this case he needs to go to the store and buy some few more horseshoes, not to lose face in front of his stylish comrades.

Fortunately, the store sells horseshoes of all colors under the sun and Valera has enough money to buy any four of them. However, in order to save the money, he would like to spend as little money as possible, so you need to help Valera and determine what is the minimum number of horseshoes he needs to buy to wear four horseshoes of different colors to a party.

Input

The first line contains four space-separated integers s_1, s_2, s_3, s_4 $(1 \le s_1, s_2, s_3, s_4 \le 10^9)$ — the colors of horseshoes Valera has.

Consider all possible colors indexed with integers.

Output

Print a single integer — the minimum number of horseshoes Valera needs to buy.

Sample test(s)

input
```
1 7 3 3
```
output
```
1
```

input
```
7 7 7 7
```
output
```
3
```

B. Two Tables

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You've got two rectangular tables with sizes $n_a \times m_a$ and $n_b \times m_b$ cells. The tables consist of zeroes and ones. We will consider the rows and columns of both tables indexed starting from 1. Then we will define the element of the first table, located at the intersection of the i-th row and the j-th column, as $a_{i,j}$; we will define the element of the second table, located at the intersection of the i-th row and the j-th column, as $b_{i,j}$.

We will call the pair of integers (x, y) a *shift* of the second table relative to the first one. We'll call the *overlap factor* of the shift (x, y) value:

$$\sum_{i,j} a_{i,j} \cdot b_{i+x,j+y},$$

where the variables i, j take only such values, in which the expression $a_{i,j} \cdot b_{i+x,j+y}$ makes sense. More formally, inequalities $1 \le i \le n_a$, $1 \le j \le m_a$, $1 \le i+x \le n_b$, $1 \le j+y \le m_b$ must hold. If there are no values of variables i, j, that satisfy the given inequalities, the value of the sum is considered equal to 0.

Your task is to find the shift with the maximum overlap factor among all possible shifts.

Input

The first line contains two space-separated integers n_a, m_a $(1 \le n_a, m_a \le 50)$ — the number of rows and columns in the first table. Then n_a lines contain m_a characters each — the elements of the first table. Each character is either a "`0`", or a "`1`".

The next line contains two space-separated integers n_b, m_b $(1 \le n_b, m_b \le 50)$ — the number of rows and columns in the second table. Then follow the elements of the second table in the format, similar to the first table.

It is guaranteed that the first table has at least one number "`1`". It is guaranteed that the second table has at least one number "`1`".

Output

Print two space-separated integers x, y $(|x|, |y| \le 10^9)$ — a shift with maximum overlap factor. If there are multiple solutions, print any of them.

Sample test(s)

input
```
3 2
01
10
00
2 3
001
111
```
output
```
0 1
```

input
```
3 3
000
010
000
1 1
1
```
output
```
-1 -1
```

C. Fractal Detector

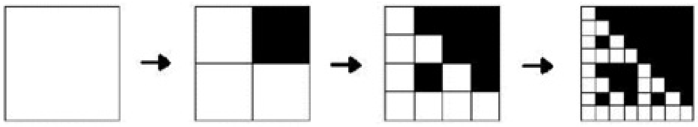
time limit per test: 4 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Little Vasya likes painting fractals very much.

He does it like this. First the boy cuts out a 2×2-cell square out of squared paper. Then he paints some cells black. The boy calls the cut out square a fractal *pattern*. Then he takes a clean square sheet of paper and paints a fractal by the following algorithm:

1. He divides the sheet into four identical squares. A part of them is painted black according to the fractal pattern.
2. Each square that remained white, is split into 4 lesser white squares, some of them are painted according to the fractal pattern. Each square that remained black, is split into 4 lesser black squares.

In each of the following steps step 2 repeats. To draw a fractal, the boy can make an arbitrary positive number of steps of the algorithm. But he need to make at least two steps. In other words step 2 of the algorithm **must be done at least once**. The resulting picture (the square with painted cells) will be a fractal. The figure below shows drawing a fractal (here boy made three steps of the algorithm).



One evening Vasya got very tired, so he didn't paint the fractal, he just took a sheet of paper, painted a $n \times m$-cell field. Then Vasya paint some cells black.

Now he wonders, how many squares are on the field, such that there is a fractal, which can be obtained as described above, and which is equal to that square. Square is considered equal to some fractal if they consist of the same amount of elementary not divided cells and for each elementary cell of the square corresponding elementary cell of the fractal have the same color.

Input

The first line contains two space-separated integers n, m $(2 \le n, m \le 500)$ — the number of rows and columns of the field, correspondingly.

Next n lines contain m characters each — the description of the field, painted by Vasya. Character "`.`" represents a white cell, character "`*`" represents a black cell.

It is guaranteed that the field description doesn't contain other characters than "`.`" and "`*`".

Output

On a single line print a single integer — the number of squares on the field, such that these squares contain a drawn fractal, which can be obtained as described above.

Sample test(s)

input

```
6 11
......*.***
*.*.*....**
.***....*.*
..***.*....
.*.*.....**
......*.*..
```

output

```
3
```

input

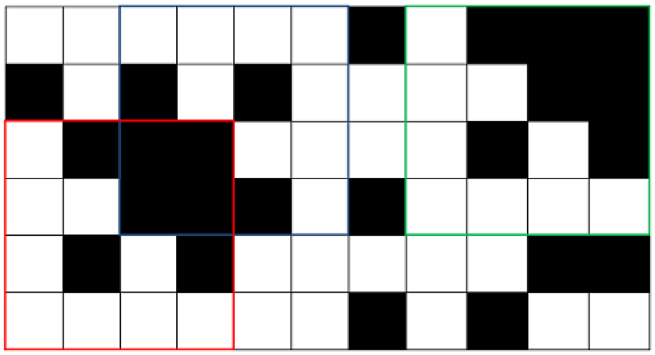
```
4 4
..**
..**
....
....
```

output

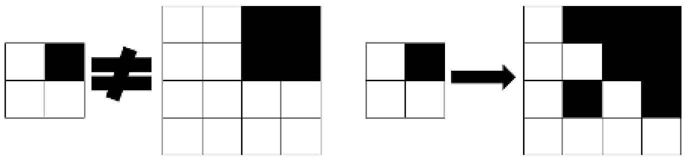
```
0
```

Note

The answer for the first sample is shown on the picture below. Fractals are outlined by red, blue and green squares.



The answer for the second sample is 0. There is no fractal, equal to the given picture.



D. Zigzag

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

The court wizard Zigzag wants to become a famous mathematician. For that, he needs his own theorem, like the Cauchy theorem, or his sum, like the Minkowski sum. But most of all he wants to have his sequence, like the Fibonacci sequence, and his function, like the Euler's totient function.

The Zigag's sequence with the zigzag factor z is an infinite sequence S_i^z $(i \geq 1; z \geq 2)$, that is determined as follows:

- $S_i^z = 2$, when $(i \mod 2(z-1)) = 0$;
- $S_i^z = (i \mod 2(z-1))$, when $0 < (i \mod 2(z-1)) \leq z$;
- $S_i^z = 2z - (i \mod 2(z-1))$, when $(i \mod 2(z-1)) > z$.

Operation $x \mod y$ means taking the remainder from dividing number x by number y. For example, the beginning of sequence S_i^3 (zigzag factor 3) looks as follows: 1, 2, 3, 2, 1, 2, 3, 2, 1.

Let's assume that we are given an array a, consisting of n integers. Let's define element number i $(1 \leq i \leq n)$ of the array as a_i. The Zigzag function is function $Z(l, r, z) = \sum_{i=l}^{r} a_i \cdot S_{i-l+1}^z$, where l, r, z satisfy the inequalities $1 \leq l \leq r \leq n$, $z \geq 2$.

To become better acquainted with the Zigzag sequence and the Zigzag function, the wizard offers you to implement the following operations on the given array a.

1. The assignment operation. The operation parameters are (p, v). The operation denotes assigning value v to the p-th array element. After the operation is applied, the value of the array element a_p equals v.
2. The Zigzag operation. The operation parameters are (l, r, z). The operation denotes calculating the Zigzag function $Z(l, r, z)$.

Explore the magical powers of zigzags, implement the described operations.

Input

The first line contains integer n $(1 \leq n \leq 10^5)$ — The number of elements in array a. The second line contains n space-separated integers: $a_1, a_2, ..., a_n$ $(1 \leq a_i \leq 10^9)$ — the elements of the array.

The third line contains integer m $(1 \leq m \leq 10^5)$ — the number of operations. Next m lines contain the operations' descriptions. An operation's description starts with integer t_i $(1 \leq t_i \leq 2)$ — the operation type.

- If $t_i = 1$ (assignment operation), then on the line follow two space-separated integers: p_i, v_i $(1 \leq p_i \leq n; 1 \leq v_i \leq 10^9)$ — the parameters of the assigning operation.
- If $t_i = 2$ (Zigzag operation), then on the line follow three space-separated integers: l_i, r_i, z_i $(1 \leq l_i \leq r_i \leq n; 2 \leq z_i \leq 6)$ — the parameters of the Zigzag operation.

You should execute the operations in the order, in which they are given in the input.

Output

For each Zigzag operation print the calculated value of the Zigzag function on a single line. Print the values for Zigzag functions in the order, in which they are given in the input.

Please, do not use the `%lld` specifier to read or write 64-bit integers in C++. It is preferred to use `cin`, `cout` streams or the `%I64d` specifier.

Sample test(s)

input

```
5
2 3 1 5 5
4
2 2 3 2
2 1 5 3
1 3 5
2 1 5 3
```

output

```
5
26
38
```

Note

Explanation of the sample test:

- Result of the first operation is $Z(2, 3, 2) = 3 \cdot 1 + 1 \cdot 2 = 5$.
- Result of the second operation is $Z(1, 5, 3) = 2 \cdot 1 + 3 \cdot 2 + 1 \cdot 3 + 5 \cdot 2 + 5 \cdot 1 = 26$.
- After the third operation array a is equal to $2, 3, 5, 5, 5$.
- Result of the forth operation is $Z(1, 5, 3) = 2 \cdot 1 + 3 \cdot 2 + 5 \cdot 3 + 5 \cdot 2 + 5 \cdot 1 = 38$.

E. The Road to Berland is Paved With Good Intentions

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Berland has n cities, some of them are connected by bidirectional roads. For each road we know whether it is asphalted or not.

The King of Berland Valera II wants to asphalt all roads of Berland, for that he gathered a group of workers. Every day Valera chooses exactly one city and orders the crew to asphalt all roads that come from the city. The valiant crew fulfilled the King's order in a day, then workers went home.

Unfortunately, not everything is as great as Valera II would like. The main part of the group were gastarbeiters — illegal immigrants who are enthusiastic but not exactly good at understanding orders in Berlandian. Therefore, having received orders to asphalt the roads coming from some of the city, the group asphalted all non-asphalted roads coming from the city, and vice versa, took the asphalt from the roads that had it.

Upon learning of this progress, Valera II was very upset, but since it was too late to change anything, he asked you to make a program that determines whether you can in some way asphalt Berlandian roads in at most n days. Help the king.

Input

The first line contains two space-separated integers n, m $(1 \le n \le 100;\ 1 \le m \le \frac{n \cdot (n-1)}{2})$ — the number of cities and roads in Berland, correspondingly. Next m lines contain the descriptions of roads in Berland: the i-th line contains three space-separated integers a_i, b_i, c_i $(1 \le a_i, b_i \le n;\ a_i \ne b_i;\ 0 \le c_i \le 1)$. The first two integers (a_i, b_i) are indexes of the cities that are connected by the i-th road, the third integer (c_i) equals 1, if the road was initially asphalted, and 0 otherwise.

Consider the cities in Berland indexed from 1 to n, and the roads indexed from 1 to m. It is guaranteed that between two Berlandian cities there is not more than one road.

Output

In the first line print a single integer x $(0 \le x \le n)$ — the number of days needed to asphalt all roads. In the second line print x space-separated integers — the indexes of the cities to send the workers to. Print the cities in the order, in which Valera send the workers to asphalt roads. If there are multiple solutions, print any of them.

If there's no way to asphalt all roads, print "`Impossible`" (without the quotes).

Sample test(s)

input

```
4 4
1 2 1
2 4 0
4 3 1
3 2 0
```

output

```
4
3 2 1 3
```

input

```
3 3
1 2 0
2 3 0
3 1 0
```

output

```
Impossible
```