XXXIV Maratón Nacional de Programación Colombia - ACIS / REDIS / CCPL 2020 ICPC

Problems

(This set contains 7 problems; problem pages numbered from 1 to 15)

General Information. Unless otherwise stated, the conditions stated below hold for all the problems. However, some problems might have specific requirements and it is important to read the problem statements carefully.

Program name. Each source file (your solution!) must be called

`<codename>.c`, `<codename>.cpp`, `<codename>.java`, or`<codename>.py`

as instructed below each problem title.

Input.

1. The input must be read from the standard input.
2. In most problems, the input can contain several test cases. Each test case is described using a number of lines that depends on the problem.
3. In most cases, when a line of input contains several values, they are separated by single spaces. No other spaces appear in the input. There are no empty lines.
4. Every line, including the last one, has the usual end-of-line mark.
5. If no end condition is given, then the end of input is indicated by the end of the input stream. There is no extra data after the test cases in the input.

Output.

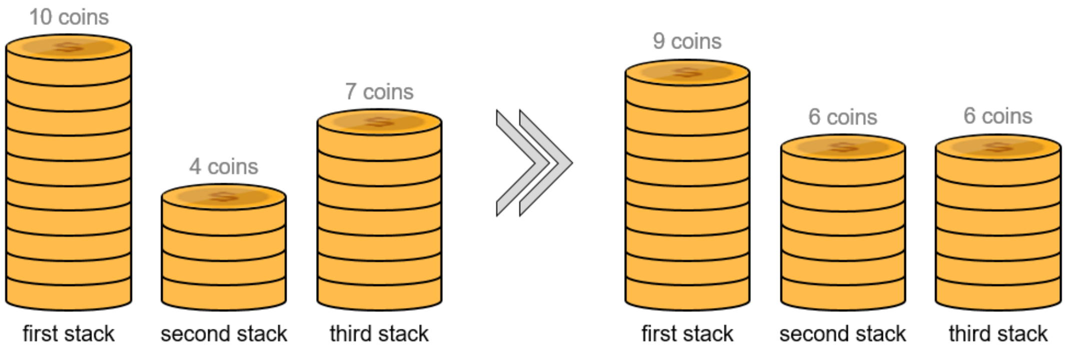
1. The output must be written to the standard output.
2. The result of each test case must appear in the output using a number of lines, which depends on the problem.
3. When a line of results contains several values, they must be separated by single spaces. No other spaces should appear in the output. There should be no empty lines.
4. Every line, including the last one, must have the usual end-of-line mark.
5. After the output of all test cases, no extra data must be written to the output.
6. To output real numbers, if no particular instructions are given, round them to the closest rational with the required number of digits after the decimal point. Ties are resolved rounding to the nearest upper value.

A: Coin Collection

Source file name: `coin.c, coin.cpp`, `coin.java`, *or* `coin.py`
Author: `Andrés Mejía`

You are a coin collector and the proud owner of an Impressive Collection of Pretty Coins (ICPC). Your coin collection is organized in $N > 0$ stacks, each with a possibly different *size* (number of coins).

Your annoying mathematician friend saw your collection and challenged you to a game: in one *movement*, you can take one coin from the top of a stack and place it on top of another stack. Given an integer $M > 0$, is it possible that after some movements every stack has a size that is a multiple of M? In that case, what is the minimum number of movements required to accomplish that?



For example, let's say $M = 3$ and your collection consists of three stacks that initially contain 10, 4 and 7 coins, respectively. If you move one coin from the first stack to the second stack and one coin from the third stack to the second stack, you would end up with stacks of sizes 9, 6 and 6, respectively, which are all multiples of 3. Furthermore, it's not possible to accomplish the same in just one movement, so two has to be the minimum number of movements required in this case.

Assume that a stack won't tip over even if it is arbitrarily large, so there is no upper limit on the number of coins you can place on the same stack. Stacks of size 0 are also allowed. Note that it might not be possible to accomplish the challenge.

Input

The input consists of several test cases, each one consisting of two lines.

The first line in each test case contains two blank-separated integers N and M: N is the number of stacks in your coin collection ($1 \leq N \leq 10^5$) and M is the desired factor for the stacks' sizes as described above ($1 \leq M \leq 10^9$). The following line contains N blank-separated integers S_i for $1 \leq i \leq N$, the number of coins in each stack ($0 \leq S_i \leq 10^9$). It is guaranteed that the total number of coins among all stacks will not exceed 10^9.

The end of the input is indicated by a single line containing `0 0`, that should not be processed.

The input must be read from standard input.

Output

For each test case, output a single line with the minimum number of movements required to make the size of every stack divisible by M, whenever possible. If it's not possible, output the word `Impossible` instead.

The output must be written to standard output.

Sample Input	**Sample Output**

```
3 3
10 4 7
4 2
2 4 6 8
2 2
1 3
2 2
1 2
4 10
3 9 9 9
6 10
4 8 2 7 0 9
0 0
```
	```
	2
	0
	1
	Impossible
	3
	6
	```

B: **Decoding Scrambled Messages**

Source file name: `decode.c, decode.cpp`, `decode.java`, *or* `decode.py`
Author: `Camilo Rocha`

Sensitive information is often encoded before transmission. Codes are designed to be difficult to break in order to keep the information they represent safe from unintended recipients.

For the purpose of this problem, it is said that a message M is *coded* by two binary strings A and B if and only if the digits of M can be colored either red or blue in such a way that the concatenation of the red digits is equal to A^a and the concatenation of the blue digits is equal to B^b, for some integers $a \geq 0$ and $b \geq 0$. Note that for any string X and any non-negative integer n, the expression X^n denotes the concatenation of n copies of X; if $n = 0$, then X^n denotes the empty string.

For example, suppose M = `0101011011010101`, A = `1011` and B = `00`. Then M is coded by A and B because M can be colored in a such a way that its red digits (denoted below by $M_{\texttt{red}}$) are equal to A^3 and its blue digits (denoted below by $M_{\texttt{blue}}$) are equal to B^2 (blue digits are also underlined in case you are color blind, or are using a black and white printer):

$$M = \underline{\texttt{0}}\texttt{101}\underline{\texttt{0}}\texttt{11011}\underline{\texttt{0}}\texttt{101}\underline{\texttt{0}}\texttt{1}$$

$$M_{\texttt{red}} = \texttt{101110111011} = (\texttt{1011})^3 = A^3$$

$$M_{\texttt{blue}} = \underline{\texttt{0000}} = (\underline{\texttt{00}})^2 = B^2$$

In this particular case, there is only one way to code M. Notice, however, that in some situations a message M could be coded by A and B in more than one way. For example, if M = `01010101`, A = `0101`, and B = `01`, then there are 8 possible ways to color M so that it is coded by A and B:

$$C_1 = \underline{\texttt{01010101}}$$

$$C_2 = \texttt{0101}\underline{\texttt{0101}}$$

$$C_3 = \texttt{01}\underline{\texttt{01}}\texttt{01}\underline{\texttt{01}}$$

$$C_4 = \underline{\texttt{01}}\texttt{0101}\underline{\texttt{01}}$$

$$C_5 = \texttt{01}\underline{\texttt{0101}}\texttt{01}$$

$$C_6 = \underline{\texttt{01}}\texttt{01}\underline{\texttt{01}}\texttt{01}$$

$$C_7 = \underline{\texttt{0101}}\texttt{0101}$$

$$C_8 = \texttt{01010101}$$

For each of these 8 colorings, integers $a \geq 0$ and $b \geq 0$ can be found such that the red digits form A^a and the blue digits form B^b. For example, for $C_1 = \underline{\texttt{01010101}}$ the red digits equal A^0 (the empty string) and the blue digits equal B^4 (four concatenated copies of B). For $C_5 = \texttt{01}\underline{\texttt{0101}}\texttt{01}$ the red digits equal A^1 and the blue digits equal B^2. On the other hand, $\underline{\texttt{0}}\texttt{101}\underline{\texttt{01}}\texttt{0}\underline{\texttt{1}}$ is not a valid coloring because there does not exist an integer $b \geq 0$ such that the blue digits $\underline{\texttt{0011}}$ are equal to B^b.

This problem aims to discover in how many different ways a message can be coded by two binary strings. Given M, A, and B, count how many possible colorings exist such that M can be coded by A and B. Two colorings are considered different if at least one digit changes its color between them.

Input

The input consists of several test cases. Each test case comprises a single line with three blank-separated binary strings M, A, and B ($1 \leq |M| \leq 500$, $1 \leq |A| \leq 50$, and $1 \leq |B| \leq 50$). It is guaranteed that M, A, and B contain only zeroes and ones.

The input must be read from standard input.

Output

For each test case, output the number of ways M can be coded by A and B. Since this number can be extremely large, output the answer modulo the decimal number 1010101.

The output must be written to standard output.

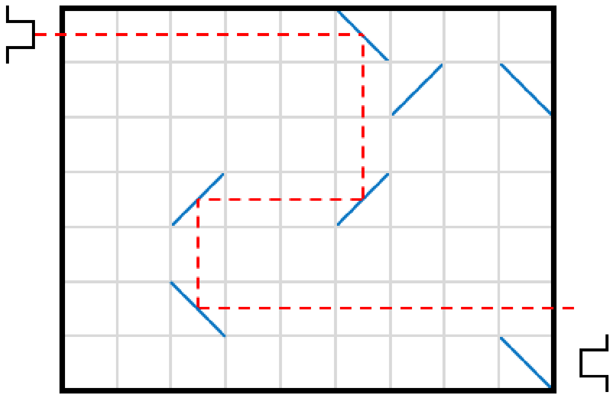
Sample Input	**Sample Output**
`0101011011010101 1011 00`	`1`
`01010101 0101 01`	`8`
`101 1 0`	`1`
`1 10 0`	`0`
`0101 10 01`	`2`
`111 1 11`	`4`

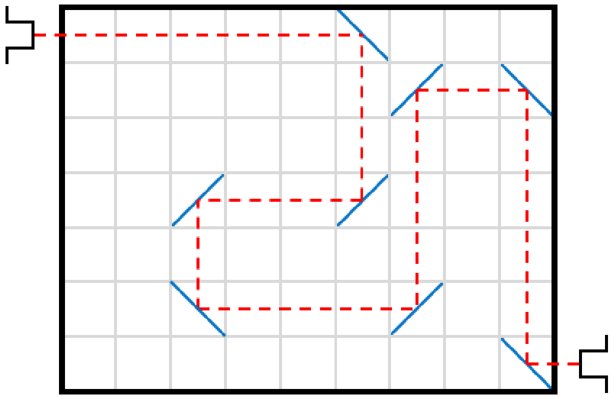
C: Lasers and Mirrors

Source file name: `laser.c,` `laser.cpp`, `laser.java`, *or* `laser.py`
Author: `Andrés Mejía`

You work as a technician in a laboratory that does research with lasers. You operate a machine that has a laser *emitter* and a laser *receiver*, and a set of *mirrors* that control the route that the laser travels. The goal is to make the laser leave the emitter, travel through the machine, and eventually reach the receiver after possibly being reflected in some of the mirrors.

The machine is described by a rectangular grid that shows where the mirrors are. Each mirror occupies exactly one cell and is always aligned at either a 45° or 135° angle with respect to the bottom horizontal edge of the machine, which causes the laser to be reflected at a 90° angle every time it hits one of the mirrors. Both sides of the mirrors are reflective and work the same way. There can't be two mirrors on the same cell.





The image at the left shows a sample machine that is broken because the laser doesn't reach the receiver. In this particular situation, it is possible to fix this machine by adding exactly one mirror, as it is shown in the image at the right.

Given the size of the machine and the configuration of the mirrors, write a program that determines if the machine is already working; if it is not working, find out if it can be fixed by adding exactly one mirror (without changing or removing any of the existing mirrors). Assume the laser is very strong, so there is no limit on the number of mirrors it can bounce off before reaching the receiver. It is allowed for the laser to intersect itself. There are no restrictions on how many mirrors the laser needs to touch; the only thing that matters is whether or not it eventually reaches the receiver (in other words, there may be mirrors that remain unused even when the machine is working).

Input

The input consists of several test cases.

The first line in each test case contains two blank-separated integers R and C, the number of rows and columns in the grid, respectively ($1 \leq R \leq 1000$, $1 \leq C \leq 1000$). The following R lines contain C characters each one and describe the machine. An empty cell is represented by a dot (`.`) and a mirror is represented by a forward slash (/) or a backslash (\).

Both the emitter and receiver are always oriented horizontally. The emitter is always located on the first row to the left of the leftmost cell and the receiver is always located on the last row to the right of the rightmost cell (just like it's shown in the examples above).

The end of the input is indicated by a single line containing `0 0`, which should not be processed.

The input must be read from standard input.

Output

For each test case, output a single line with the following text, depending on the situation:

- `It works`: If the machine is working and the laser already reaches the receiver;
- `I can fix it`: If the machine is broken but it is possible to make the laser reach the receiver just by adding one mirror;
- `Send help`: If the machine is broken and it can't be fixed or you'd need more than 1 mirror to fix it.

The output must be written to standard output.

Sample Input

```
6 9
.....\...
....../.\
.........
../../...
..\......
........\
6 9
.....\...
....../.\
.........
../../...
..\.../..
........\
2 3
..\
\..
2 3
..\
\.\
4 9
.........
/.......\
......../
\........
4 9
.........
/.......\
....\.../
\........
0 0
```

Sample Output

```
I can fix it
It works
I can fix it
It works
Send help
I can fix it
```

D: **Package Delivery**

Source file name: `delivery.c, delivery.cpp, delivery.java`, *or* `delivery.py`
Author: `Rafael García`

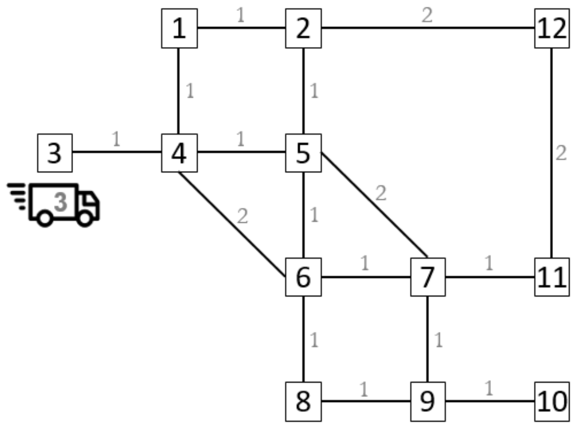
Loading, transporting, and delivering packages to clients or companies in a safe and timely manner is the business of the Colombian Center for Package Location - CCPL (a subsidiary of the International Center for Package Coordination - ICPC).

To do the job, CCPL receives a daily list of packages to be picked up and delivered. For each package in the list, it is indicated the place where it must be picked up and the place where it must be delivered.

To collect, transport, and deliver the packages, CCPL has a single vehicle with a fixed capacity K, that starts its journey parked at a specific place. Assuming that all the packages have weight 1, the vehicle would be able to transport maximum K packages simultaneously.

Your task is to help CCPL and ICPC to do their job efficiently. That is, knowing the capacity of the vehicle, the detailed map of the city that includes the times to go from a place to another, and the list of orders that have been assigned to that vehicle, you must find the minimum time that the vehicle should use to deliver the packages as required.

For example, if the city map is the one in the figure, the vehicle (whose fixed capacity is $K = 3$) is at place 3, and the list of packages indicates that a package must be picked up at location 8 and delivered at location 1, a second package must be picked up at location 11 and delivered at 2, and a third package must be collected at place 10 and delivered at 12, then the packages can be collected and delivered in 14 minutes using the route $3 \to 4 \to 6 \to 8 \to 9 \to 10 \to 9 \to 7 \to 11 \to 12 \to 2 \to 1$.



Input

The input consists of several test cases, each one described as follows:

- The first line contains three blank-separated natural numbers V, E, and N, where V is the number of places in the city that can be reached by the vehicle ($2 \leq V \leq 500$), E is the number of roads connecting different places in the city ($V - 1 \leq E \leq V \cdot (V - 1)/2$), and N is the number of orders ($1 \leq N \leq 12$). Assume that the places are numbered from 1 to V.
- Each one of the following E lines contains three blank-separated natural numbers s, t, and c, describing a road in the city, where (s, t) indicates the two connected places ($1 \leq s \leq V$, $1 \leq t \leq V$, $s \neq t$), and c indicates the time in minutes to travel the road ($1 \leq c \leq 100$). You may suppose that the time to travel from s to t is the same that the time to travel from t to s and that each road is described once.

- The next line contains two blank-separated natural numbers K and x, where K is the fixed capacity of the vehicle in number of packages ($1 \leq K \leq 4$), and x is the number of the place where the vehicle is parked at before starting its journey ($1 \leq x \leq V$). You may suppose that it is possible to reach all places starting at the place x.
- The next N lines describe the list of orders. Each line contains two blank-separated natural numbers a and b, being a the number of the place where a package must be picked up ($1 \leq a \leq V$), and b the number of the place where the package must be delivered ($1 \leq b \leq V$, $b \neq a$). There could be repeated orders in the list.

$V = E = N = 0$ is the last line in the input and not need to be processed.

The input must be read from standard input.

Output

For each test case, print one line with a natural number indicating the minimum time required to collect, transport, and deliver the entire list of N packages.

The output must be written to standard output.

Sample Input	Sample Output

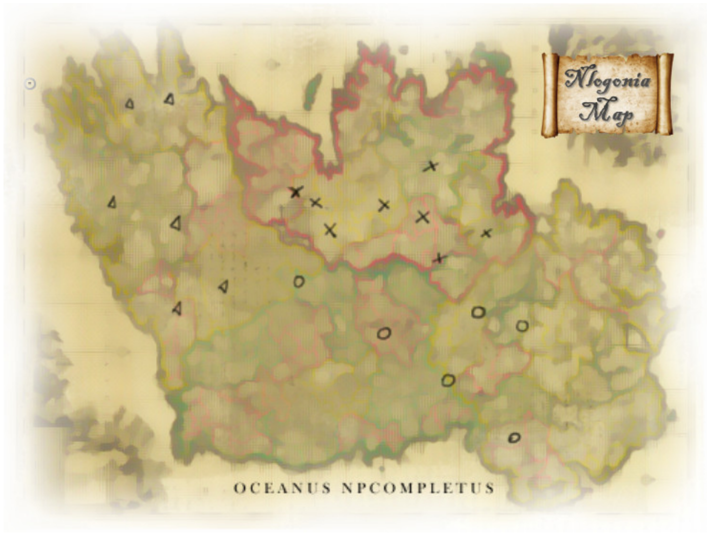
```
4 3 1
1 2 1
2 3 1
3 4 1
1 1
2 4
12 16 3
1 2 1
2 12 2
12 11 2
5 2 1
1 4 1
3 4 1
4 5 1
6 4 2
5 7 2
5 6 1
6 7 1
8 6 1
8 9 1
11 7 1
7 9 1
10 9 1
3 3
8 1
11 2
10 12
0 0 0
```
	```
	3
	14
	```

E: **Territorial Division**

Source file name: `tdivision.c, tdivision.cpp`, `tdivision.java`, *or* `tdivision.py`
Author: `Rafael García`

Dividing a country in regions has never been an easy task. Technically, a territorial division is a set of non-overlapping areas or *regions*, each one contained in the country, and whose union equals the whole country area. A region is defined by the *cities* contained in it.

Nlogonia is a country whose area is a flat rectangular map, with $N \geq 1$ cities which are represented by points in the map. According to the country legislation, a territorial division is *admissible* if, for every region, any two cities in the region may be connected with a straight road that lies completely within the region (i.e., without touching the region's border). Here is an example of a map, where cities with the same choices are marked with similar icons ($\circ$, $\triangle$, and $\times$).



The central government has decided to define an admissible territorial division consisting of at most $K \geq 1$ regions. For that purpose, they have formulated K declarations of fundamental principles, each one to rule one of the possible regions. Then, every Nlogonian city has chosen (using an internal procedure) the region to which it wants to belong in accordance with one of the K declarations. An admissible division is shown below.

Harvey & van der Hoeven Land
Nlogonia Map
Tony Hoare Region
II
III
I
Maharg Plains
OCEANUS NPCOMPLETUS

The government would like to know if the cities' choices are consistent with the idea of defining an admissible territorial division of at most K regions. You are hired by Nlogonia in order to establish if this is the case.

Input

The input consists of several test cases, each one described as follows:

- The first line contains two blank-separated natural numbers N and K, the number of Nlogonian cities ($1 \leq N \leq 10^4$) and the number of regions to define ($1 \leq K \leq 10^4$).
- Next there are N lines, one for each Nlogonian city. The line corresponding to a given city contains three blank-separated natural numbers x, y, r. The pair (x, y) defines a point that represents the city in the Nlogonian map ($0 \leq x \leq 10^4$, $0 \leq y \leq 10^4$). Finally, r identifies the region chosen by the city ($1 \leq r \leq K$). You can assume that no two cities are located at the same point in the map.

The input ends with a line containing two zeroes. This line does not define a case to be processed.

The input must be read from standard input.

Output

For each test case, print one line containing either `Y` (yes) or `N` (not) depending on whether there exists an admissible territorial division consistent with the cities' regional choices.

The output must be written to standard output.

Sample Input	**Sample Output**

```
9 4
1 1 2
2 1 2
5 1 3
4 2 3
3 3 4
2 3 1
1 4 1
4 4 4
5 4 4
6 3
1 1 1
2 1 2
4 1 1
3 2 2
2 3 2
4 3 1
0 0
```
	```
	Y
	N
	```

F: Tic-Tac-Points!

Source file name: `tictac.c, tictac.cpp`, `tictac.java`, *or* `tictac.py`
Author: `Nicolás Cardozo`

Tic-Tac-Points! is a new perspective on everyone's favorite game tic-tac-toe. The classic tic-tac-toe game requires you to take turns with another player to place $\times$'s or $\circ$'s on a square board of side d until one of you gets a line of length equal to d.

In this modification of the game, you will be playing on your own, following the game play dynamics described below. You are given a board of *side d* filled out with different point boosters (of one digit, from 1 to 9) for each cell in the board. If you conquer a cell, then you will earn the points of taking the cell (10) boosted by its multiplier.

	0	**1**	**2**	**3**	**4**
0	1	2	3	2	1
1	1	6	1	5	1
2	1	2	1	1	2
3	1	1	3	1	1
4	1	7	1	5	1

Figure 1: 5x5 board for Tic-Tac-Points!

Given a board of side d, a *target* amount of points t and a *length* L between 2 and d, you are asked to determine whether you can show a *straight line* of length L within the board such that you can earn at least the target number of points. With a *straight line* we mean a contiguous set of horizontal, vertical or diagonal cells. Note that if you conquer d number of cells to reach the solution, an additional 10 points are granted for being such a good player!

The example in the figure shows a board of size $d = 5$. Define a game on this board, with target $t = 150$ and length $l = 5$. In this game, you can get to the target score by conquering the cells in column 3. The points obtained by conquering that column are $2 * 10 + 5 * 10 + 1 * 10 + 1 * 10 + 5 * 10 + 10$ for a total of 150 points! As another example, with the same board but with target $t = 79$ and length $l = 2$, you can get a score of 80 points with the cells at $(0, 2)$ and $(1, 3)$. It should be clear that there may be many different ways to obtain at least the required target amount t.

Input

There are several test cases to solve. Each test case starts with a line consisting of three blank-separated natural numbers: the target number t ($20 \leq t \leq 9010$), the board dimensions d ($2 \leq d \leq 100$), and the line length L ($2 \leq L \leq d$). The following d lines contain the d blank-separated point boosters of each row of the board. Each point booster is a digit between 1 and 9.

The end of the input data is signaled by a line with three blank-separated zeroes.

The input must be read from standard input.

Output

For each test case you should print a line just with a letter Y or a letter N indicating if you can obtain a score at least as high as the target amount.

The output must be written to standard output.

Sample Input	**Sample Output**

```
150 5 5
1 2 3 2 1
1 6 1 5 1
1 2 1 1 2
1 1 3 1 1
1 7 1 5 1
140 2 2
3 1
1 9
100 5 5
1 2 3 2 1
1 6 1 5 1
1 2 1 1 2
1 1 3 1 1
1 7 1 5 1
0 0 0
```
	```
	Y
	N
	Y
	```

G: **Vaccine**

Source file name: `vaccine.c, vaccine.cpp`, `vaccine.java`, *or* `vaccine.py`
Author: `Diego Caballero`

A new virus has been discovered recently across the kingdom. It is affecting most of the population and it is quickly spreading around. The minister of health is tracking down the virus by randomly selecting affected people and taking a sample from them. Early results have shown that the virus has been mutating as it is passed from one individual to another showing differences in the genetic code extracted for each of the individual samples. The genetic code is composed of Adenine (`A`), Cytosine (`C`), Guanine (`G`), and Thymine (`T`).

The minister is trying to develop a new vaccine that can suppress all the known mutations. For a vaccine to suppress the virus, it needs to have the same genetic code than the virus allowing **at most** three positions to be different. For example, if we take the genetic code of a vaccine to be `GCTAC`, this vaccine will be effective for a mutation with genetic code `GCTTG` but it would not be effective for a mutation `CGTCG`.

The minister is asking for your help to find a single vaccine to suppress all the identified mutations.

Input

The input consists of several test cases. Each test case starts with a line containing two blank-separated integers n and m, where n represents the length of each mutation ($1 \leq n \leq 50$) and m represents the amount of samples collected by the minister ($1 \leq m \leq 100$). The next m lines contain n characters each one (`A`, `C`, `G`, or `T`), representing the genetic code of a single sample. Please note that the samples are not necessarily different because the virus does not mutate in every interaction.

The input must be read from standard input.

Output

For each test case, output the lexicographically smallest vaccine that will suppress all the mutations or print `IMPOSSIBLE` if the vaccine cannot be found.

The output must be written to standard output.

Sample Input

```
5 4
AAAAC
GCTCA
GAAAA
GCTAA
5 2
GCTTG
CGTCG
7 5
GCTACCT
AGGCTAC
CGCCTAG
AGCTAGC
TAAGTCC
```

Sample Output

```
AATAA
AATAG
IMPOSSIBLE
```