Problem A
Alien Integers

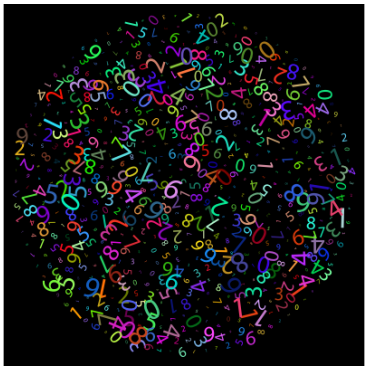


Photo by Gordon Dylan Johnson

Exploratory robots are essential to expanding our understanding of the moon, Mars, and other celestial bodies. When there are two or more robots in the same vicinity, they need to be marked by humanly readable integers for purposes of visual tracking. To reduce the possibility of error in visual recognition of the robots in dark and dusty environments, numbers are chosen so that they have no digits in common. More formally, two non-negative integers are *alien* to each other if there is no digit which occurs in both of their decimal representations. For example, 11 229 and 67 840 are alien to each other, while 2 022 and 427 are not. No integer is alien to 1 234 567 890.

The numbers on robots in the same area should also be close to each other numerically (for instance, to simplify processing of the marks by the software, to make them easy to remember, to distinguish them from other groups of robots marked in similar manner, ...).

The Institute for Computerized Planetary Circumambulation needs a program to identify the nearest number that is alien to a given number. Can you help?

Input

The input consists of an integer N ($1 \leq N \leq 10^{15}$) given on a single line.

Output

When there is one non-negative alien integer Y closest to the input number N, output the value of Y. When there are two such integers that are equally close to the input number N, output both of them in ascending order, on a single line. When there is no integer alien to the input number N, output `Impossible`.

Sample Input 1	Sample Output 1
24	19

Sample Input 2	Sample Output 2
605	499 711

Sample Input 3	Sample Output 3
98765432011	Impossible

This page is intentionally left blank.

Problem B
Avoiding Asteroids

You escaped the planet, but enemy starfighters are right behind you! To make things worse, in between you and the safety of your base is an asteroid field. The starfighters would never follow you into the asteroids, but unfortunately you used all your fuel in the escape, other than one small drop that will be enough for just one final burst of the engines. You've got two choices. You can surrender and face a long prison term. Or, you can aim your starship toward your base, and fire that one burst. There's no way to know how fast that burst will make you move, but you know it'll eventually get you to the safety of your base. That is, unless you collide with one of the asteroids.

You have a scan of the asteroid field, and need to determine whether you are sure to have a safe route through the field, or if surrender is the better option. Some important notes:

- Your ship and the base are insignificant compared to the size of the asteroids, and can be treated as points.
- Your scan of the asteroids will give the center of mass, the direction of motion (of the center of mass), and the points on the convex hull of the asteroid (all measured at the time of the scan).
- The center of mass of each asteroid translates over time in the given direction, but at an unknown speed. (You do know that the speed is non-negative: asteroids will not move backwards.)
- The asteroids can rotate about their center of mass, in any direction, at any angular velocity. You don't have any information about the tumbling of the asteroid, and must allow for any possibility.
- Asteroids do not bounce off of each other (treat asteroids as though they pass through each other if they happen to collide).
- If you decide not to surrender, your ship will move through the asteroid field at an unknown speed: perhaps extremely slow, or perhaps extremely fast. You have no control over your ship's speed and no ability to steer around asteroids.
- You may assume that no asteroid can "graze" the ship. Formally, for each asteroid either:
  1. for any combination of your ship's speed, the asteroid's speed, and the asteroid's angular velocity, the distance between your ship and the asteroid will be at least 10^{-6} units, or
  2. there exists some combination of your ship's speed, the asteroid's speed, and the asteroid's angular velocity such that your ship intersects the asteroid and the distance between your ship and the nearest point on the surface of the asteroid is at least 10^{-6} units.
- No asteroid can collide with (or graze) your base: for any combination of an asteroid's speed and angular velocity, the distance between your base and the asteroid will be at least 10^{-6} units at all times.
- Your ship starts at least distance 10^{-6} units away from your base.

Input

The first line of input consists of six real numbers $s_x, s_y, s_z, b_x, b_y, b_z$ and a single integer n, separated by spaces. These give the 3D coordinates of your ship (s_x, s_y, s_z), the 3D coordinates of the base (b_x, b_y, b_z), and the number of asteroids, n ($0 \leq n \leq 30$).

The next $2n$ lines describe each of the asteroids, with one asteroid per pair of lines. The first line of each pair contains six real numbers $p_x, p_y, p_z, d_x, d_y, d_z$ and a single integer m, separated by spaces, giving the position (p_x, p_y, p_z) and direction (d_x, d_y, d_z) of the asteroid's center of mass, as well as the number of points m on the asteroid's convex hull ($4 \leq m \leq 8$). The second line of each pair contains $3m$ real numbers, giving m triples of points $c_{i,x}, c_{i,y}, c_{i,z}$, all separated by spaces. These are the 3D coordinates of the points on the convex hull of the asteroid, taken at the time of the scan. The asteroid center of mass is located somewhere within the convex hull (but not necessarily at the geometric center).

All real values in the input are in the range $[-2 \cdot 10^6, 2 \cdot 10^6]$ and have at most 6 digits after the decimal point. Each asteroid's direction is a unit vector, up to numerical tolerance, and will satisfy $\left|\sqrt{d_x^2 + d_y^2 + d_z^2} - 1\right| \leq 10^{-6}$.

Output

If you are guaranteed to have a safe route through the asteroid field, then print `Go`. Otherwise, print `Surrender`.

Sample Input 1

```
10.0 0.0 0.0 0.0 0.0 0.0 1
5.0 3.0 0.0 0.0 -1.0 0.0 6
7.0 3.0 0.0 3.0 3.0 0.0 5.0 1.0 0.0 5.0 5.0 0.0 5.0 3.0 1.0 5.0 3.0 -1.0
```

Sample Output 1

```
Surrender
```

Sample Input 2

```
10.0 0.0 0.0 0.0 0.0 0.0 1
5.0 3.0 0.0 0.0 1.0 0.0 6
7.0 3.0 0.0 3.0 3.0 0.0 5.0 1.0 0.0 5.0 5.0 0.0 5.0 3.0 1.0 5.0 3.0 -1.0
```

Sample Output 2

```
Go
```

Sample Input 3

```
10.0 0.0 0.0 0.0 0.0 0.0 1
15.0 0.0 0.0 1.0 0.0 0.0 6
14.0 0.0 0.0 16.0 0.0 0.0 15.0 1.0 0.0 15.0 -1.0 0.0 15.0 0.0 1.0 15.0 0.0 -1.0
```

Sample Output 3

```
Go
```

Problem C
Common Factors

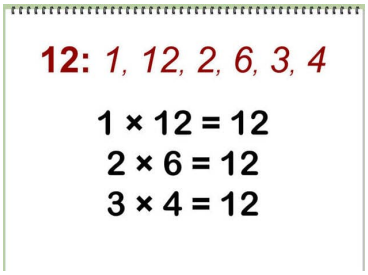


Photo by Bob Chao

Everyone likes to share things in common with other people.

Numbers are the same way! Numbers like it when they have a factor in common.

For example, 4 and 6 share a common factor of 2, which gives them something to talk about.

For a given integer n, we define a function, $f(n)$, equal to the number of integers in the range $[1, n]$ that share a common factor greater than 1 with n.

Furthermore, we can define a second function, $g(n)$, which characterizes the fraction of numbers that like a given number as follows: $g(n) = \frac{f(n)}{n}$

What we really want to know though, is, for any integer $2 \leq k \leq n$, what is the maximum value of $g(k)$?

Input

The input consists of a single integer n ($2 \leq n \leq 10^{18}$), the value of n for the input case.

Output

For the provided test case, output the result as a fraction, in lowest terms, in the form p/q where the greatest common divisor of p and q is 1.

Sample Input 1	Sample Output 1
10	2/3

Sample Input 2	Sample Output 2
100	11/15

This page is intentionally left blank.

Problem D
Dimensional Analysis

You are deriving some equations for physics homework and are worried you've made some mistakes. To debug them, you will apply dimensional analysis. In dimensional analysis, you remove all magnitudes and units from a set of equations until you are left only with strings representing *physical quantities* and the special dimensionless constant 1. The following is an example set of such equations:

```
velocity * time = length
frequency = 1 / time
acceleration = velocity / time
force = mass * acceleration
force = mass * length / time / time
```

Your equations, like the ones above, involve only multiplication and division (no addition, exponentiation, etc.) As you can see, some quantities (like velocity) are defined in terms of other, more basic quantities (length, time). But you're not sure what the correct relationships are between the quantities. You do know that none of the quantities in your equations are unitless: it should not be possible, through any set of algebraic manipulations, to prove that any quantity is equal to the dimensionless constant 1. (You may assume that all physical quantities in your equations have positive real magnitudes. In particular, you may freely divide any equation by any quantity without worrying about division by zero; and you may take square or higher roots of any quantity.)

A set of equations violating this condition is *invalid* (and a set of equations is valid otherwise). The above example is a valid system. Here is an invalid one:

```
foo * bar = xyzzy
foo = xyzzy * bar
```

By substituting the second equation into the first, and dividing both sides by `xyzzy`, you get `bar * bar = 1`. Taking a square root of both sides, you get `bar = 1` and so `bar` is dimensionless.

Given the set of equations in your dimensional analysis, compute whether the equations are valid.

Input

The first line of the input is a single integer N, the number of equations ($1 \leq N \leq 100$). Each of the N subsequent lines of input contains one equation. An equation consists of two expressions, separated by an equal sign surrounded by spaces " `=` ". Each expression contains one or more atoms, separated by either " `*` " or " `/` "; each atom is either the character '`1`' or a physical quantity, represented by a lowercase string (containing one or more ASCII characters between '`a`' and '`z`'). At most 100 unique physical quantities appear in total across all equations, each equation contains at most 100 atoms, and the total number of characters in all atoms across all equations will not exceed 100 000.

There will be exactly one space before and after each '=', '`*`', and '`/`' and the equations will contain no other whitespace or other extraneous punctuation. See the sample input for examples of how equations are formatted.

The operator `*` represents multiplication and `/` represents division. Expressions follow the usual associativity rules: `a / b * c` is the same as `a * c / b` but different from `a / b / c`.

Output

Print `invalid` if it is possible to prove that at least one of the physical quantities in an equation in the input must be dimensionless. Print `valid` otherwise.

Sample Input 1

```
5
velocity * time = length
frequency = 1 / time
acceleration = velocity / time
force = mass * acceleration
force = mass * length / time / time
```

Sample Output 1

```
valid
```

Sample Input 2

```
2
foo * bar = xyzzy
foo = xyzzy * bar
```

Sample Output 2

```
invalid
```

Sample Input 3

```
5
time * power = energy
energy = work
work = force * distance
distance = distance
1 / 1 = 1 * 1 / 1
```

Sample Output 3

```
valid
```

Problem E
Eye of Sauron

Photo by Usuario Mararie

Little Elrond is obsessed with the Lord of the Rings series. Between lectures he likes to doodle the central tower of the great fortress Barad-dûr in the margins of his notebook. Afterward, he always double checks his drawings to ensure they are accurate: with the Eye of Sauron located in the very center of the tower. If any are incorrect, he makes sure to fix them.

You are to write a program that reads a representation of his tower, and ensures that the drawing is correct, with a properly centered eye.

Input

Input consists of a single string of length n, where $4 \leq n \leq 100$. Input strings will consist only of three types of characters: vertical bars, open parentheses, and closing parentheses. Input strings contain one or more vertical bars followed by a set of matching parentheses (the "eye"), followed by one or more vertical bars. For a drawing to be "correct", the number of vertical bars on either side of the "eye" must match. Input will always contain a pair of correctly matched parentheses, with no characters between them. No other characters will appear in the string.

Output

On a single line print the word "correct" if the drawing is accurate or the word "fix" if there is an error that needs addressing.

Sample Input 1	Sample Output 1
`|()||`	`fix`

Sample Input 2	Sample Output 2
`||||()||||`	`correct`

Sample Input 3	Sample Output 3
`|()|`	`correct`

Sample Input 4	Sample Output 4
`|||()|`	`fix`

This page is intentionally left blank.

Problem F
ilove Strings

It's that time of year when love is in the air. You're no stranger to love. You are obsessed with strings but not just any strings. You love "ilove" Strings. An "ilove" String is a string of length 5 with the following properties:

- Alternates between vowels (excluding 'y' and 'Y') and consonants (including 'y' and 'Y')
- Begins with a vowel (excluding 'y' and 'Y')
- Consists of 5 pairwise distinct characters (distinguishing between upper and lower case)

Examples of "ilove" Strings includes "`ilove`", "`image`", "`IxoXO`", and "`abide`". Examples of non-"ilove" Strings include , "`ideas`", "`maker`", "`inane`", "`oxOXo`" and "`abides`".

The loveliness of a string is the number of subsequences of the string that form an "ilove" String. Although "`ilooove`" is not an "ilove" String, it does have a loveliness of 3.

Input

Input contains a single string of between 1 and 100 000 lowercase and uppercase Latin characters, representing the string whose loveliness is to be computed.

Output

For the provided string, print one line with a single integer L — the loveliness of the string modulo $10^9 + 7$.

Sample Input 1	Sample Output 1
`ilovestrings`	`4`

Sample Input 2	Sample Output 2
`idont`	`0`

Sample Input 3	Sample Output 3
`CAPital`	`1`

This page is intentionally left blank.

Problem G
MrCodeFormatGrader

Instructor Bob is building an automatic source code formatting grading website. The name of the website is "MrCodeFormatGrader". When you run Bob's program and submit your source code to it, the program will output C, the number of lines of source code; N, the number of source code lines that have format errors; and a list of line numbers that are not properly formatted. The line numbers are separated by spaces and in increasing order.

So for a 100-line program, with 10 errors occurring on lines 2, 3, 5, 10, 11, 12, 25, 26, 88, and 89, the original program might output this:

```
100 10
2 3 5 10 11 12 25 26 88 89
```

However, the people that used this program said that this output is not very readable. Your job is to read the output above and show two lists: a compressed list of errors and another compressed list of correct lines.

Input

Input consists of two lines. The first line contains two integers: C, the number of lines of source code in the program being checked, where $2 \leq C \leq 100\,000$; and N, the number of source code lines containing format errors, where $1 \leq N \leq \min(C, 1\,000)$. The second line of input will contain N values, identifying the line numbers of source code lines with format errors. These values are positive integers less than or equal to C and are listed in strictly increasing order.

Output

Output consists of two lines. The first line should start with the string: "`Errors:`", followed by a space, followed by a list of lines with errors. The second line of output should start with the string: "`Correct:`", followed by a space, followed by a list of lines without errors. For both lists, a contiguous set of line numbers should be represented by listing the first and last line, separated by a hyphen ('-'). In lists with two or more items, items should be separated by a comma and space except for the last value (or last range of contiguous values) of each list, which should be separated with an "and" instead of a comma and space. See the sample output. Note: every program will have at least one error and at least 1 correct line.

Sample Input 1

```
100 10
2 3 5 10 11 12 25 26 88 89
```

Sample Output 1

```
Errors: 2-3, 5, 10-12, 25-26 and 88-89
Correct: 1, 4, 6-9, 13-24, 27-87 and 90-100
```

Sample Input 2

```
40 18
1 3 4 6 7 8 9 12 13 14 20 25 26 27 28 30 38 40
```

Sample Output 2

```
Errors: 1, 3-4, 6-9, 12-14, 20, 25-28, 30, 38 and 40
Correct: 2, 5, 10-11, 15-19, 21-24, 29, 31-37 and 39
```

Problem H
Mult!

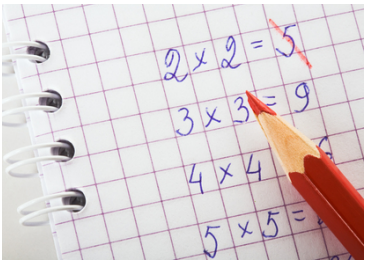


Photo by Larisa Lofitskaya

Nora Mainder has a game she plays with her students to help them learn multiplication. She calls out a sequence of numbers and the students have to determine when she names a whole number multiple of the first number. When a student recognizes such a multiple, he or she must call out "Mult!", ending this round of the game. Then a new round begins with a new initial number. Fortunately her students are very bright and never fail to recognize a multiple, so they all cry out at once—a "multitude" of shouts.

For instance, if she calls out "8, 3, 12, 6, 24," her students all yell "Mult!" when she reaches 24 because it is a multiple of the first number, 8. If she begins a second round of the game with the sequence "14, 12, 9, 70," the class will call out "Mult!" when she reaches 70, a multiple of the first number, 14.

Given a sequence of numbers called out by Nora during several rounds of the game, identify which numbers ought to produce a shout of "Mult!"

Input

The first line of input contains an integer n, $2 \leq n \leq 1\,000$, the length of the number sequence. The following n lines contains the sequence, one number per line. All numbers in the sequence are positive integers less than or equal to 100. The sequence is guaranteed to contain at least one complete round of the game (but may end with an incomplete round).

Output

Print all of the sequence elements that will cause the class to shout "Mult!" Each value should be printed on a separate line.

Sample Input 1	Sample Output 1
10 8 3 12 6 24 14 12 9 70 5	24 70

Sample Input 2	Sample Output 2
5 3 3 2 5 7	3

Problem I
Pizza Party!

Photo by Bob Chao

You are co-organizing a computer science conference, and you are in charge of a pizza party for the conference guests. Each guest holds preferences over combinations of toppings, and guests are seated in groups by table in the conference center ballroom. One pizza is served to each table. You must make sense of each table's collective preferences by finding pizza toppings that make all guests happy at a particular table. Since you are paying by the topping, the conference organizers wish to find the *minimal* set of satisfying toppings for each table's pizza.

Pizza preferences are specified as statements in either an *absolute* or *implicative* form. An *absolute preference* for `pepperoni` is a statement that pepperoni must be on the pizza in order to satisfy a particular guest. An *implicative preference* is a conditional statement. For example, the preference `if pepperoni and sausage then mushroom` indicates that a pizza with both pepperoni and sausage must also have mushrooms. Note that the implicative preference says nothing about a preference for mushrooms when either pepperoni or sausage are absent from the pizza.

Guests are already organized by table and each table's preferences are aggregated. It is your job to find a topping assignment for the pizza at each table.

Input

The first line of input consists of a single integer c ($1 \leq c \leq 1\,000$), the number of preferences for the pizza you are trying to create. This is followed by c lines containing either an *absolute* or *implicative* preference.

The name of each topping is a single word, not exceeding 10 characters in length, consisting of only lowercase English letters. The words `if`, `and`, `or`, and `then` are not valid names for pizza toppings.

Absolute preferences consist of a single topping name. All *implicative preferences* are either of the form `if` t_1 `and` t_2 `and` ... `and` t_k `then` t_{k+1}, *or* `if` t_1 `or` t_2 `or` ... `or` t_k `then` t_{k+1}, where each of $t_1, t_2, \ldots, t_{k+1}$ are topping names and $1 \leq k \leq 500$.

Output

For the provided test case, print one line with a single integer t — the minimal number of toppings for a pizza that satisfies all guests at the table.

Sample Input 1

```
4
peppers
if spinach and olives then tomatoes
spinach
feta
```

Sample Output 1

```
3
```

Sample Input 2

```
5
pepperoni
pineapple
if pepperoni and sausage then mushroom
ham
if pineapple and ham then bacon
```

Sample Output 2

```
4
```

Sample Input 3

```
4
pepperoni
sausage
if pepperoni and sausage then mushrooms
if mushrooms or peppers then cheese
```

Sample Output 3

```
4
```

Problem J
Stacking Up

Jack and Jason's Pancakes, CC BY-SA 4.0, via Wikimedia Commons

Stacy has recently started work at Stacks"R"Us, a leading manufacturer of stack-related products such as children's blocks, pancake spatulas, and dining hall tray dispensers. As a brand new employee, she has been tasked with testing the latest product in Stacks"R"Us's line of automated electronic stack machines, the *Stackulator 3000*.

The Stackulator 3000 holds a single stack of positive integers in its memory, and supports three instructions, `1`, `d`, and `+`. The first two instructions are fairly standard:

- `1`: push the number 1 onto the top of the stack.
- `d`: duplicate the number on top of the stack, that is, take the number currently on top of the stack and push another copy of it on top. It is an error to execute the `d` instruction on an empty stack.

Unfortunately, due partly to miscommunication between the designer and the hardware team, and partly to confusion about the difference between a stack *pointer* and a stack *entry*, the `+` instruction ended up being a little wonky:

- `+`: pop the top two numbers from the stack, *decrement all remaining stack entries by one*, then add the two popped elements and push the result onto the stack. It is an error to execute the `+` instruction on a stack containing fewer than two integers.

For example, executing the program `1d+11+` produces the sequence of stack states illustrated in Figure J.1, ultimately resulting in a stack containing two entries, 1 on the bottom and 2 on the top.

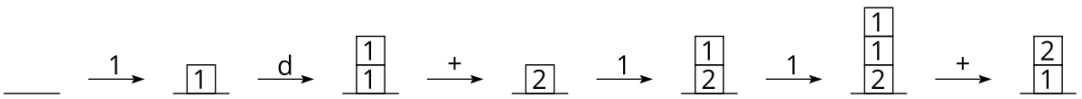


Figure J.1: Execution of `1d+11+`

If a stack entry of 1 is decremented during an addition operation, it is simply removed from the stack, since the stack can only hold positive integers.

Since she is still an apprentice stack tester, Stacy was assigned to work with Stan, a more experienced tester. Stan proposes that they split the work evenly: Stan will come up with stacks, and it will then be Stacy's job to create Stackulator 3000 programs which should generate the given stacks. They can then run Stacy's programs to ensure the Stackulator 3000 gives the expected results.

Stacy could do this, but she is now too busy looking for a different job. That's where you come in: please help Stacy by creating Stackulator 3000 programs that can generate the stacks given by Stan.

Input

The first line of input consists of an integer $1 \leq n \leq 1\,000$. The following line contains n space-separated positive integers, indicating the desired contents of the stack from bottom to top. Each integer x will be in the range $1 \leq x \leq 10^6$.

Output

Output a single line containing a valid Stackulator 3000 program, which, when executed beginning with an empty stack, results in a stack holding the desired contents. Your program must consist of only the characters `1`, `d`, and `+` and be no more than $100\,000$ characters in length. If there is more than one valid program which results in the desired stack, you may output any of them.

Sample Input 1	Sample Output 1
3 1 1 1	1dd

Sample Input 2	Sample Output 2
3 1 2 3	11+1+1+1+1+11+1+1+11+1+

Problem K
Stamp Combinations

Photo by John Keyser

You're on your way to pick up a package from the store, wrap it, and mail it. You want to bring a certain number of stamps to mail it, and you have a long roll of stamps that you can use. Now, over time, you've occasionally pulled off a stamp from somewhere in the middle of the roll, and now you're left with a strip where stamps are clustered in groups, separated by empty spaces.

Being practical, you don't want to tear individual stamps, and so you'll only tear the roll in between the clusters of stamps. And, you don't want to be left with multiple rolls of stamps, so you can only pull these groups off of the beginning or end of the roll. Is it possible to do this with the roll of stamps you have?

Input

The first line of input consists of two integers, m and n, $1 \leq m, n \leq 10^6$, $1 \leq m \cdot n \leq 10^7$, giving the number of clusters of stamps and the number of queries you will have.

The second line contains m integers, $a_i, 1 \leq a_i \leq 100$, separated by spaces. These give the number of stamps in each cluster, in order from the beginning to the end of the roll.

The next n lines each contain one integer, q, $0 \leq q \leq 10^8$, giving a query, where a query is the number of stamps needed to mail a package.

Output

For each query, output one line containing "`Yes`" if it is possible to bring that number of stamps, or "`No`" if it is not possible to bring that number of stamps.

Explanation of Sample

The roll contains 5 clusters of stamps.

A group of 2 stamps can be taken by tearing just the first cluster of stamps off of the roll.

A group of 10 stamps can be taken by tearing the first two clusters of stamps off of the roll.

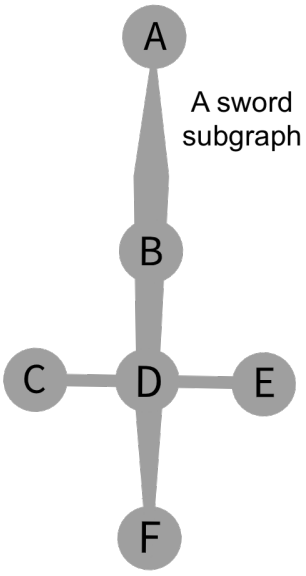
A group of 5 stamps can be taken by tearing the first cluster (of 2) and the last cluster (of 3) off of the roll.

A group of 17 stamps can be taken by taking the entire roll.

It is not possible to take just 1 stamp, since tearing the roll to take the single stamp in the middle of the roll would not leave you with a single roll of stamps, and it is not possible to tear in the middle of a cluster.

Sample Input 1	Sample Output 1
5 5 2 8 1 3 3 2 10 5 17 1	Yes Yes Yes Yes No

Problem L
Sword Counting



Michael is planning to open a graph store. A graph store is a type of store which specializes in selling various graphs in different sizes and shapes. Michael has done extensive research on the market and the business seems very profitable. He has also invented a graph generator device which is able to produce any graph that he wants.

However, there is one small problem which is keeping him from starting the next booming business: Michael does not know how each graph should be priced. After weeks of reading, he found that a graph's value can be calculated based on the number of its sword subtrees. A group of six distinct vertices (let's represent them with letters A to F) form a sword subtree if the following edges exist between them (see figure):

- A is connected to B.
- B is connected to A and D.
- C is connected to D.
- D is connected to B, C, E, and F.
- E is connected to D.
- F is connected to D.

Two sword subtrees T_1 and T_2 are considered to be different if there is any edge e which exists in T_1 but does not exist in T_2.

As a highly knowledgeable person and his business partner, your task is to help Michael count the number of sword subtrees in his generated graphs. Given an undirected graph, write a program to count the number of its sword subtrees.

Input

The first line of input will contain two integers N and M, $(1 \leq N, M \leq 100\,000)$, representing the number of vertices and edges in the graph. The next M lines each will contain two integers u_i and v_i $(1 \leq u_i, v_i \leq N)$, the endpoints of an undirected edge. It is guaranteed that the graph described by these edges does not contain multiple edges or self loops.

Output

Output a single integer, the number of sword subtrees in the given graph.

Sample Input 1

```
8 7
1 2
1 3
4 1
1 5
5 6
5 7
8 5
```

Sample Output 1

```
6
```

Sample Input 2

```
6 15
1 2
1 3
1 4
1 5
1 6
2 3
2 4
2 5
2 6
3 4
3 5
3 6
4 5
4 6
5 6
```

Sample Output 2

```
120
```

Problem M
Tic-Tac State

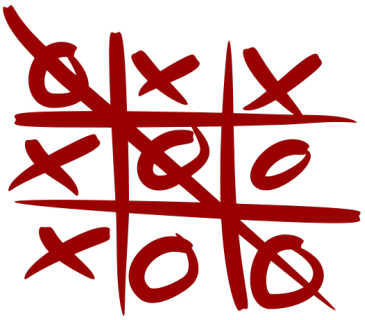


Photo by Symode09

Congratulations! You are starting your internship for the famous digital archaeologist, Endiana Jones. You have been assigned to evaluate the results of saved games of a 1980's version of Tic-Tac-Toe. In those days, programmers had very little storage, so they saved game state as compactly as possible. In this case, the state was in a 32-bit register. Bits $0-8$ stored the positions that had been played and bits $9-17$ indicated an X or O. A set bit (1 bit) indicated a played position for bits $0-8$ or that X played for bits $9-17$. Bit 18 indicated the next player to play. (Bits are numbered from right to left, starting at the least-significant bit.) If bit 18 is set (is 1), it is X's turn to play next. Visually the bits were laid out as shown in Figure M.1:

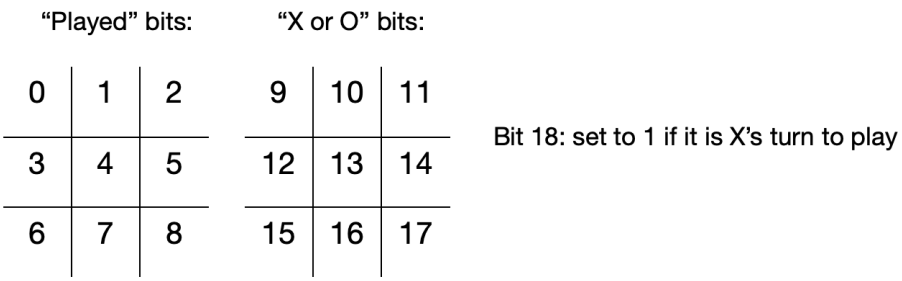


Figure M.1: Bits in a game state

The game represented by the picture would have bits $0-8$ set because all positions have been played. Bits 10, 11, 12, and 15 would be set because those positions contain an X. Bit 18 would be set because it would be X's turn next. The state would be represented in binary as: `1 001 001 110 111 111 111` or in octal as 01116777.

The Tic-Tac-Toe implementation was very simple, and a cat's game (draw or tie) was not called until all positions had been played. Your task is to interpret the state of the game given an **octal** integer.

Quick review of Tic-Tac-Toe: Two players play the game. Either player may go first. One player's mark is X and the other's is O. Each player takes turns placing their mark in one of the empty squares. If a player gets three marks in a horizontal, vertical, or diagonal row, that player wins. If there is no winner and there are no empty spaces left, the game stops, and the game is declared "Cat's" game.

Input

The first line of input consists of a single decimal integer c ($1 \leq c \leq 10\,000$), the number of states to evaluate. Each of the following c lines will have a single **octal** number representing the state of a game. All numbers will follow the convention of writing octal numbers with a leading 0. All game states will be legal, that is, achievable in a real game of Tic-Tac-Toe.

Output

For each game state number print a single line indicating the state of the game. The four possible output lines are:

```
X wins
O wins
Cat's
In progress
```

Sample Input 1	Sample Output 1
4 01116777 07037 01416777 050055	O wins X wins Cat's In progress

Problem N
Venn Intervals

A *Venn interval diagram* is a graphical way to illustrate intersections of sets in 1D, similar to the more familiar Venn diagrams in 2D. A Venn interval diagram is defined by assigning a non-zero-length integer interval $[l_i, u_i)$ to each set S_i. The *regions* where different intervals overlap correspond to intersections of different combinations of the sets (see Figure N.1).

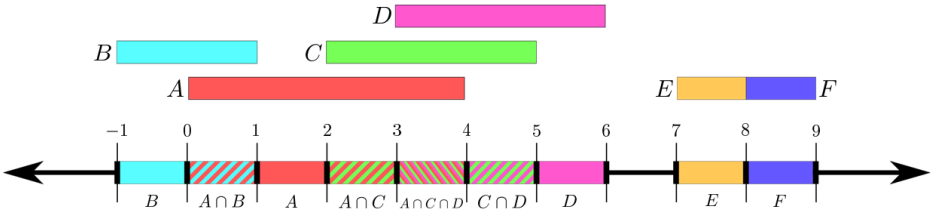


Figure N.1: Venn interval diagram corresponding to Sample Input 1. The intervals assigned to each of the six sets A through F are shown at the top; the regions encoded by the diagram are labeled along the number line. This Venn interval diagram has nine regions, and is not degenerate (since no interval is contained in any other.)

More formally, given the interval assigned to each set, you can compute the regions that appear in the Venn interval diagram with the following algorithm:

- Combine together all of the interval starting points l_i and ending points u_i into a single set, and sort them to get an integer sequence $p_1 < p_2 < \cdots < p_m$.
- Label each interval (p_j, p_{j+1}) spanning consecutive p values with all the sets S_i for which (p_j, p_{j+1}) is contained within $[l_i, u_i)$. (Note that partial overlaps are not possible: the p-interval is either contained within, or disjoint from, each set-interval).
- The set of all non-empty labels are the *regions* of the Venn interval diagram. Each label $\{S_a, S_b, \ldots\}$ represents an intersection of sets $S_a \cap S_b \cap \cdots$.

In the example Venn interval diagram shown above, there are six sets (A through F) and nine regions (B, $A \cap B$, A, $A \cap C$, $A \cap C \cap D$, $C \cap D$, D, E, and F). This example Venn interval diagram has a "hole" where no sets overlap (between the end of region D and the start of E); this is allowed. Note also that $E \cap F$ is *not* a region: E and F's intervals touch at their endpoints, but the two do not overlap on a p-interval.

Not every list of regions has a corresponding illustration as a Venn interval diagram. Consider for example the regions $A \cap B$, $B \cap C$, and $C \cap A$. There is no way to lay out intervals for A, B, and C on the real line to form exactly these three regions (any Venn interval diagram that includes these three regions must also include $A \cap B \cap C$). In addition, a Venn interval diagram is *degenerate* if any interval is contained within another interval: if $l_i \leq l_j$ and $u_j \leq u_i$ for some $i \neq j$. For example, if C's right endpoint were at 4 instead of 5, the Venn interval diagram in Figure N.1 would become degenerate, since the interval assigned to the set C would then be contained within A's interval (corresponding to Sample Input 3).

Given a list of regions, construct a **non-degenerate** Venn interval diagram containing exactly those regions, if possible.

Input

The first line of the input contains a single integer n, the number of regions required in your diagram ($1 \leq n \leq 4\,000$). The next n lines each describe one region. Each line begins with an integer k, the number of sets that intersect to form the region ($1 \leq k \leq 2\,000$), followed by k space-separated set names. Each set name contains only lowercase or uppercase letters (`a-z`, `A-Z`) and no set name is longer than ten characters. All set names listed on the same line are distinct, and no two regions list the exact same collection of set names. No more than 2 000 unique set names appear in total across all regions.

Output

If a non-degenerate Venn interval diagram does not exist whose regions match the input, print `IMPOSSIBLE` and produce no further output.

Otherwise, print the intervals describing any one valid diagram. For each set name that appears in the input, print a line of output starting with that set name, followed by two space-separated integers l and r: the left and right endpoints of the interval assigned to that set name in your diagram. The endpoints must satisfy $-10^6 \leq l < u \leq 10^6$. You may list the sets in any order, but every set name that appears in the input must correspond to exactly one line of output.

Every region in the input must appear at least once in your Venn interval diagram, and your diagram must not contain any regions other than those specified in the input. You diagram must not be degenerate: no interval should be enclosed inside another.

Sample Input 1	Sample Output 1
9 1 A 2 A B 1 B 2 A C 3 A C D 2 C D 1 D 1 E 1 F	B -1 1 A 0 4 C 2 5 D 3 6 E 7 8 F 8 9

Sample Input 2	Sample Output 2
3 2 A B 2 B C 2 C A	IMPOSSIBLE

Sample Input 3	Sample Output 3
8 1 A 2 A B 1 B 2 A C 3 A C D 1 D 1 E 1 F	IMPOSSIBLE

This page is intentionally left blank.