A. Chord

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

Vasya studies music.

He has learned lots of interesting stuff. For example, he knows that there are 12 notes: `C`, `C#`, `D`, `D#`, `E`, `F`, `F#`, `G`, `G#`, `A`, `B`, `H`. He also knows that the notes are repeated cyclically: after `H` goes `C` again, and before `C` stands `H`. We will consider the `C` note in the row's beginning and the `C` note after the `H` similar and we will identify them with each other. The distance between the notes along the musical scale is measured in tones: between two consecutive notes there's exactly one semitone, that is, 0.5 tone. The distance is taken from the lowest tone to the uppest one, that is, the distance between `C` and `E` is 4 semitones and between `E` and `C` is 8 semitones

Vasya also knows what a chord is. A chord is an unordered set of no less than three notes. However, for now Vasya only works with triads, that is with the chords that consist of exactly three notes. He can already distinguish between two types of triads — major and minor.

Let's define a major triad. Let the triad consist of notes X, Y and Z. If we can order the notes so as the distance along the musical scale between X and Y equals 4 semitones and the distance between Y and Z is 3 semitones, then the triad is major. The distance between X and Z, accordingly, equals 7 semitones.

A minor triad is different in that the distance between X and Y should be 3 semitones and between Y and Z — 4 semitones.

For example, the triad "`C E G`" is major: between `C` and `E` are 4 semitones, and between `E` and `G` are 3 semitones. And the triplet "`C# B F`" is minor, because if we order the notes as "`B C# F`", than between `B` and `C#` will be 3 semitones, and between `C#` and `F` — 4 semitones.

Help Vasya classify the triad the teacher has given to him.

Input

The only line contains 3 space-separated notes in the above-given notation.

Output

Print "`major`" if the chord is major, "`minor`" if it is minor, and "`strange`" if the teacher gave Vasya some weird chord which is neither major nor minor. Vasya promises you that the answer will always be unambiguous. That is, there are no chords that are both major and minor simultaneously.

Sample test(s)

input
```
C E G
```
output
```
major
```

input
```
C# B F
```
output
```
minor
```

input
```
A B H
```
output
```
strange
```

B. Keyboard

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

Vasya learns to type. He has an unusual keyboard at his disposal: it is rectangular and it has n rows of keys containing m keys in each row. Besides, the keys are of two types. Some of the keys have lowercase Latin letters on them and some of the keys work like the "Shift" key on standard keyboards, that is, they make lowercase letters uppercase.

Vasya can press one or two keys with one hand. However, he can only press two keys if the Euclidean distance between the centers of the keys does not exceed x. The keys are considered as squares with a side equal to 1. There are no empty spaces between neighbouring keys.

Vasya is a very lazy boy, that's why he tries to type with one hand as he eats chips with his other one. However, it is possible that some symbol can't be typed with one hand only, because the distance between it and the closest "Shift" key is strictly larger than x. In this case he will have to use his other hand. Having typed the symbol, Vasya returns other hand back to the chips.

You are given Vasya's keyboard and the text. Count the minimum number of times Vasya will have to use the other hand.

Input

The first line contains three integers n, m, x ($1 \le n, m \le 30$, $1 \le x \le 50$).

Next n lines contain descriptions of all the keyboard keys. Each line contains the descriptions of exactly m keys, without spaces. The letter keys are marked with the corresponding lowercase letters. The "Shift" keys are marked with the "`S`" symbol.

Then follow the length of the text q ($1 \le q \le 5 \cdot 10^5$). The last line contains the text T, which consists of q symbols, which are uppercase and lowercase Latin letters.

Output

If Vasya can type the text, then print the minimum number of times he will have to use his other hand. Otherwise, print "-1" (without the quotes).

Sample test(s)

input

```
2 2 1
ab
cd
1
A
```

output

```
-1
```

input

```
2 2 1
ab
cd
1
e
```

output

```
-1
```

input

```
2 2 1
ab
cS
5
abcBA
```

output

```
1
```

input

```
3 9 4
qwertyuio
asdfghjkl
SzxcvbnmS
35
TheQuIcKbRoWnFOXjummsovertHeLazYDOG
```

output

```
2
```

Note

In the first sample the symbol "`A`" is impossible to print as there's no "Shift" key on the keyboard.

In the second sample the symbol "`e`" is impossible to print as there's no such key on the keyboard.

In the fourth sample the symbols "`T`", "`G`" are impossible to print with one hand. The other letters that are on the keyboard can be printed. Those symbols come up in the text twice, thus, the answer is 2.

C. Trains

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

Vasya the programmer lives in the middle of the Programming subway branch. He has two girlfriends: Dasha and Masha, who live at the different ends of the branch, each one is unaware of the other one's existence.

When Vasya has some free time, he goes to one of his girlfriends. He descends into the subway at some time, waits the first train to come and rides on it to the end of the branch to the corresponding girl. However, the trains run with different frequencies: a train goes to Dasha's direction every a minutes, but a train goes to Masha's direction every b minutes. If two trains approach at the same time, Vasya goes toward the direction with the lower frequency of going trains, that is, to the girl, to whose directions the trains go less frequently (see the note to the third sample).

We know that the trains begin to go simultaneously before Vasya appears. That is the train schedule is such that there exists a moment of time when the two trains arrive simultaneously.

Help Vasya count to which girlfriend he will go more often.

Input

The first line contains two integers a and b ($a \neq b, 1 \leq a, b \leq 10^6$).

Output

Print "`Dasha`" if Vasya will go to Dasha more frequently, "`Masha`" if he will go to Masha more frequently, or "`Equal`" if he will go to both girlfriends with the same frequency.

Sample test(s)

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

input
```
5 3
```
output
```
Masha
```

input
```
2 3
```
output
```
Equal
```

Note

Let's take a look at the third sample. Let the trains start to go at the zero moment of time. It is clear that the moments of the trains' arrival will be periodic with period 6. That's why it is enough to show that if Vasya descends to the subway at a moment of time inside the interval $(0, 6]$, he will go to both girls equally often.

If he descends to the subway at a moment of time from 0 to 2, he leaves for Dasha on the train that arrives by the second minute.

If he descends to the subway at a moment of time from 2 to 3, he leaves for Masha on the train that arrives by the third minute.

If he descends to the subway at a moment of time from 3 to 4, he leaves for Dasha on the train that arrives by the fourth minute.

If he descends to the subway at a moment of time from 4 to 6, he waits for both trains to arrive by the sixth minute and goes to Masha as trains go less often in Masha's direction.

In sum Masha and Dasha get equal time — three minutes for each one, thus, Vasya will go to both girlfriends equally often.

D. Vasya and Types

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

Programmer Vasya is studying a new programming language &K*. The &K* language resembles the languages of the C family in its syntax. However, it is more powerful, which is why the rules of the actual C-like languages are unapplicable to it. To fully understand the statement, please read the language's description below carefully and follow it and not the similar rules in real programming languages.

There is a very powerful system of pointers on &K* — you can add an asterisk to the right of the existing type X — that will result in new type $X*$. That is called pointer-definition operation. Also, there is the operation that does the opposite — to any type of X, which is a pointer, you can add an ampersand — that will result in a type $\&X$, to which refers X. That is called a dereference operation.

The &K* language has only two basic data types — `void` and `errtype`. Also, the language has operators `typedef` and `typeof`.

- The operator "`typedef` A B" defines a new data type B, which is equivalent to A. A can have asterisks and ampersands, and B cannot have them. For example, the operator `typedef void** ptptvoid` will create a new type `ptptvoid`, that can be used as `void**`.
- The operator "`typeof` A" returns type of A, brought to `void`, that is, returns the type `void**...*`, equivalent to it with the necessary number of asterisks (the number can possibly be zero). That is, having defined the `ptptvoid` type, as shown above, the `typeof ptptvoid` operator will return `void**`.

An attempt of dereferencing of the `void` type will lead to an error: to a special data type `errtype`. For `errtype` the following equation holds true: `errtype*` = `&errtype` = `errtype`. An attempt to use the data type that hasn't been defined before that will also lead to the `errtype`.

Using `typedef`, we can define one type several times. Of all the definitions only the last one is valid. However, all the types that have been defined earlier using this type do not change.

Let us also note that the dereference operation has the lower priority that the pointer operation, in other words $\&T*$ is always equal to T.

Note, that the operators are executed consecutively one by one. If we have two operators "`typedef &void a`" and "`typedef a* b`", then at first `a` becomes `errtype`, and after that `b` becomes `errtype* = errtype`, but **not** `&void* = void` (see sample 2).

Vasya does not yet fully understand this powerful technology, that's why he asked you to help him. Write a program that analyzes these operators.

Input

The first line contains an integer n $(1 \le n \le 100)$ — the number of operators. Then follow n lines with operators. Each operator is of one of two types: either "`typedef` A B", or "`typeof` A". In the first case the B type differs from `void` and `errtype` types, and besides, doesn't have any asterisks and ampersands.

All the data type names are non-empty lines of no more than 20 lowercase Latin letters. The number of asterisks and ampersands separately in one type in any operator does not exceed 10, however if we bring some types to `void` with several asterisks, their number may exceed 10.

Output

For every `typeof` operator print on the single line the answer to that operator — the type that the given operator returned.

Sample test(s)

input

```
5
typedef void* ptv
typeof ptv
typedef &&ptv node
typeof node
typeof &ptv
```

output

```
void*
errtype
void
```

input

```
17
typedef void* b
typedef b* c
typeof b
typeof c
typedef &b b
typeof b
typeof c
typedef &&b* c
typeof c
typedef &b* c
typeof c
typedef &void b
typeof b
typedef b******* c
typeof c
```

```
typedef &&b* c
typeof c
```

output

```
void*
void**
void
void**
errtype
void
errtype
errtype
errtype
```

Note

Let's look at the second sample.

After the first two queries `typedef` the `b` type is equivalent to `void*`, and `c` — to `void**`.

The next query `typedef` redefines `b` — it is now equal to `&b = &void* = void`. At that, the `c` type doesn't change.

After that the `c` type is defined as `&&b* = &&void* = &void = errtype`. It doesn't influence the `b` type, that's why the next `typedef` defines `c` as `&void* = void`.

Then the `b` type is again redefined as `&void = errtype`.

Please note that the `c` type in the next query is defined exactly as `errtype******* = errtype`, and not `&void******* = void******`. The same happens in the last `typedef`.

E. Interesting Game

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

Two best friends Serozha and Gena play a game.

Initially there is one pile consisting of n stones on the table. During one move one pile should be taken and divided into an arbitrary number of piles consisting of $a_1 > a_2 > ... > a_k > 0$ stones. The piles should meet the condition $a_1 - a_2 = a_2 - a_3 = ... = a_{k-1} - a_k = 1$. Naturally, the number of piles k should be no less than two.

The friends play in turns. The player who cannot make a move loses. Serozha makes the first move. Who will win if both players play in the optimal way?

Input

The single line contains a single integer n $(1 \le n \le 10^5)$.

Output

If Serozha wins, print k, which represents the minimal number of piles into which he can split the initial one during the first move in order to win the game.

If Gena wins, print "-1" (without the quotes).

Sample test(s)

```
input
3
output
2
```

```
input
6
output
-1
```

```
input
100
output
8
```