A. 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.

B. 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`.

C. 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 $a_1 > a_2 > \ldots > a_k > 0$ stones. The piles should meet the condition $a_1 - a_2 = a_2 - a_3 = \ldots = a_{k-1} - a_k = 1$. 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
```

D. Beautiful Road

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

A long time ago in some country in Asia were civil wars.

Each of n cities wanted to seize power. That's why sometimes one city gathered an army and sent it to campaign against another city.

Road making was difficult, so the country had few roads, exactly $n - 1$. Also you could reach any city from any other city going on those roads.

Even during the war the Oriental people remain spiritually rich and appreciate the beauty of nature. And to keep the memory of this great crusade for the centuries to come, they planted one beautiful tree by the road on which the army spent most time. The Oriental people love nature, that's why if there were several such roads, then one tree was planted by each of them.

Recently, when the records of the war were found, it became clear that each city attacked each other one exactly once. There were exactly $n(n - 1)$ attacks in total. Everyone has been wondering what road after those wars became the most beautiful, that is, by which road they planted the largest number of beautiful trees.

Input

The first line contains an integer n ($2 \le n \le 10^5$), which represents the number of cities. Next $n - 1$ lines contain three integers each: the numbers of cities a_i, b_i ($1 \le a_i, b_i \le n$), connected by the i-th road and the number of days d_i the army spends to go on it ($1 \le d_i \le 10^9$). The lengths of several roads may coincide.

Output

Print on the first line two integers — the number of beautiful trees on the most beautiful road and the number of the most beautiful roads. Print on the second line the list of the most beautiful roads in the sorted order by the numbers' increasing. The roads are numbered from 1 to $n - 1$ in the order in which they are given in the input data.

Please, do not use `%lld` specificator to write 64-bit integers in C++. It is preferred to use the `cout` stream (also you may use the `%I64d` specificator).

Sample test(s)

input

```
2
2 1 5
```

output

```
2 1
1
```

input

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

output

```
16 1
2
```

E. Mogohu-Rea Idol

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

A long time ago somewhere in the depths of America existed a powerful tribe governed by the great leader Pinnie-the-Wooh. Once the tribe conquered three Maya cities. Pinnie-the-Wooh grew concerned: there had to be some control over the conquered territories. That's why he appealed to the priests of the supreme god Mogohu-Rea for help.

The priests conveyed the god's will to him: to control these three cities he should put an idol to Mogohu-Rea — that will create a religious field over the cities. However, the idol is so powerful that it can easily drive the people around it mad unless it is balanced by exactly three sacrifice altars, placed one in each city. To balance the idol the altars should be placed so that the center of mass of the system of these three points coincided with the idol. When counting the center of mass consider that all the altars have the same mass.

Now Pinnie-the-Wooh is thinking where to put the idol. He has a list of hills, that are suitable to put an idol there. Help him to identify on which of them you can put an idol without risking to fry off the brains of the cities' population with the religious field.

Each city has a shape of a convex polygon such that no three vertexes lie on a straight line. The cities can intersect. Each altar should be attached to the city through a special ceremony, besides, it must be situated on the city's territory (possibly at the border). Thus, there may be several altars on a city's territory, but exactly one of them will be attached to the city. The altars, the idol and the hills are points on the plane, some of them may coincide.

The hills are taken into consideration independently from each other, the altars' location for different hills may also be different.

Input

First follow descriptions of the three cities, divided by empty lines. The descriptions are in the following format:

The first line contains an integer n, which represent the number of the polygon's vertexes ($3 \le n \le 5 \cdot 10^4$). Next n lines contain two integers x_i, y_i each, they are the coordinates of the polygon's i-th vertex in the counterclockwise order.

After the cities' description follows the integer m ($1 \le m \le 10^5$), which represents the number of hills. Next m lines each contain two integers x_j, y_j, they are the coordinates of the j-th hill.

All the coordinates in the input data do not exceed $5 \cdot 10^8$ in the absolute value.

Output

For each hill print on a single line "YES" (without the quotes) or "NO" (without the quotes), depending on whether the three sacrifice altars can be put to balance the idol or not.

Sample test(s)

input

```
3
0 0
1 0
1 1

4
8 8
5 5
6 4
8 4

3
-1 -1
-3 -1
-2 -2

5
0 0
2 1
7 1
1 1
5 3
```

output

```
NO
YES
NO
YES
NO
```

Note

For the hill at $(2, 1)$ the altars can be placed at the points $(1, 0)$, $(7, 5)$, $(-2, -2)$, for the hill at $(1, 1)$ — at the points $(0, 0)$, $(6, 4)$, $(-3, -1)$. Many other groups of three points can do the trick. There are no suitable points for other hills.