Codeforces Round #123 (Div. 2)

A. Let's Watch Football

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

Valeric and Valerko missed the last Euro football game, so they decided to watch the game's key moments on the Net. They want to start watching as soon as possible but the connection speed is too low. If they turn on the video right now, it will "hang up" as the size of data to watch per second will be more than the size of downloaded data per second.

The guys want to watch the whole video without any pauses, so they have to wait some **integer** number of seconds for a part of the video to download. After this number of seconds passes, they can start watching. Waiting for the whole video to download isn't necessary as the video can download after the guys started to watch.

Let's suppose that video's length is c seconds and Valeric and Valerko wait t seconds before the watching. Then for any moment of time t_0, $t \le t_0 \le c + t$, the following condition must fulfill: the size of data received in t_0 seconds is not less than the size of data needed to watch $t_0 - t$ seconds of the video.

Of course, the guys want to wait as little as possible, so your task is to find the minimum integer number of seconds to wait before turning the video on. The guys must watch the video without pauses.

Input

The first line contains three space-separated integers a, b and c $(1 \le a, b, c \le 1000, a > b)$. The first number ($a$) denotes the size of data needed to watch one second of the video. The second number (b) denotes the size of data Valeric and Valerko can download from the Net per second. The third number (c) denotes the video's length in seconds.

Output

Print a single number — the minimum integer number of seconds that Valeric and Valerko must wait to watch football without pauses.

Examples

input
4 1 1
output
3

input
10 3 2
output
5

input
13 12 1
output
1

Note

In the first sample video's length is 1 second and it is necessary 4 units of data for watching 1 second of video, so guys should download $4 \cdot 1 = 4$ units of data to watch the whole video. The most optimal way is to wait 3 seconds till 3 units of data will be downloaded and then start watching. While guys will be watching video 1 second, one unit of data will be downloaded and Valerik and Valerko will have 4 units of data by the end of watching. Also every moment till the end of video guys will have more data then necessary for watching.

In the second sample guys need $2 \cdot 10 = 20$ units of data, so they have to wait 5 seconds and after that they will have 20 units before the second second ends. However, if guys wait 4 seconds, they will be able to watch first second of video without pauses, but they will download 18 units of data by the end of second second and it is less then necessary.

B. After Training

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

After a team finished their training session on Euro football championship, Valeric was commissioned to gather the balls and sort them into baskets. Overall the stadium has n balls and m baskets. The baskets are positioned in a row from left to right and they are numbered with numbers from 1 to m, correspondingly. The balls are numbered with numbers from 1 to n.

Valeric decided to sort the balls in the order of increasing of their numbers by the following scheme. He will put each new ball in the basket with the least number of balls. And if he's got several variants, he chooses the basket which stands closer to the middle. That means that he chooses the basket for which $|\frac{m+1}{2} - i|$ is minimum, where i is the number of the basket. If in this case Valeric still has multiple variants, he chooses the basket with the minimum number.

For every ball print the number of the basket where it will go according to Valeric's scheme.

Note that the balls are sorted into baskets in the order of increasing numbers, that is, the first ball goes first, then goes the second ball and so on.

Input

The first line contains two space-separated integers n, m ($1 \le n, m \le 10^5$) — the number of balls and baskets, correspondingly.

Output

Print n numbers, one per line. The i-th line must contain the number of the basket for the i-th ball.

Examples

input
4 3

output
2 1 3 2

input
3 1

output
1 1 1

C. Try and Catch

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

Vasya is developing his own programming language VPL (Vasya Programming Language). Right now he is busy making the system of exceptions. He thinks that the system of exceptions must function like that.

The exceptions are processed by try-catch-blocks. There are two operators that work with the blocks:

1. The `try` operator. It opens a new try-catch-block.
2. The `catch(<exception_type>, <message>)` operator. It closes the try-catch-block that was started last and haven't yet been closed. This block can be activated only via exception of type <exception_type>. When we activate this block, the screen displays the <message>. If at the given moment there is no open try-catch-block, then we can't use the `catch` operator.

The exceptions can occur in the program in only one case: when we use the `throw` operator. The `throw(<exception_type>)` operator creates the exception of the given type.

Let's suggest that as a result of using some `throw` operator the program created an exception of type a. In this case a try-catch-block is activated, such that this block's `try` operator was described in the program earlier than the used `throw` operator. Also, this block's `catch` operator was given an exception type a as a parameter and this block's `catch` operator is described later that the used `throw` operator. If there are several such try-catch-blocks, then the system activates the block whose `catch` operator occurs earlier than others. If no try-catch-block was activated, then the screen displays message "`Unhandled Exception`".

To test the system, Vasya wrote a program that contains only `try`, `catch` and `throw` operators, one line contains no more than one operator, the whole program contains exactly one `throw` operator.

Your task is: given a program in VPL, determine, what message will be displayed on the screen.

Input

The first line contains a single integer: n ($1 \le n \le 10^5$) the number of lines in the program. Next n lines contain the program in language VPL. Each line contains no more than one operator. It means that input file can contain empty lines and lines, consisting only of spaces.

The program contains only operators `try`, `catch` and `throw`. It is guaranteed that the program is correct. It means that each started try-catch-block was closed, the `catch` operators aren't used unless there is an open try-catch-block. The program has exactly one `throw` operator. The program may have spaces at the beginning of a line, at the end of a line, before and after a bracket, a comma or a quote mark.

The exception type is a nonempty string, that consists only of upper and lower case english letters. The length of the string does not exceed 20 symbols. Message is a nonempty string, that consists only of upper and lower case english letters, digits and spaces. Message is surrounded with quote marks. Quote marks shouldn't be printed. The length of the string does not exceed 20 symbols.

Length of any line in the input file does not exceed 50 symbols.

Output

Print the message the screen will show after the given program is executed.

Examples

input

8
try
  try
  throw (AE)
  catch (BE, "BE in line 3")
 	
  try
  catch(AE, "AE in line 5")
catch(AE,"AE somewhere")

output
AE somewhere

input

8
try
  try
  throw (AE)
  catch (AE, "AE in line 3")
 	
  try
  catch(BE, "BE in line 5")
catch(AE,"AE somewhere")

output

```
AE in line 3
```

input

```
8
try
    try
        throw ( CE )
    catch ( BE, "BE in line 3")

    try
    catch(AE, "AE in line 5")
catch(AE,"AE somewhere")
```

output

```
Unhandled Exception
```

Note

In the first sample there are 2 try-catch-blocks such that `try` operator is described earlier than `throw` operator and `catch` operator is described later than `throw` operator: `try-catch(BE,"BE in line 3")` and `try-catch(AE,"AE somewhere")`. Exception type is AE, so the second block will be activated, because operator `catch(AE,"AE somewhere")` has exception type AE as parameter and operator `catch(BE,"BE in line 3")` has exception type BE.

In the second sample there are 2 try-catch-blocks such that `try` operator is described earlier than `throw` operator and `catch` operator is described later than `throw` operator: `try-catch(AE,"AE in line 3")` and `try-catch(AE,"AE somewhere")`. Exception type is AE, so both blocks can be activated, but only the first one will be activated, because operator `catch(AE,"AE in line 3")` is described earlier than `catch(AE,"AE somewhere")`

In the third sample there is no blocks that can be activated by an exception of type CE.

D. Analyzing Polyline

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

As Valeric and Valerko were watching one of the last Euro Championship games in a sports bar, they broke a mug. Of course, the guys paid for it but the barman said that he will let them watch football in his bar only if they help his son complete a programming task. The task goes like that.

Let's consider a set of functions of the following form:

$$y(x) = \begin{cases} kx + b, & \text{if } kx + b \geq 0, \\ 0, & \text{if } kx + b < 0. \end{cases}$$

Let's define a sum of n functions $y_1(x), \ldots, y_n(x)$ of the given type as function $s(x) = y_1(x) + \ldots + y_n(x)$ for any x. It's easy to show that in this case the graph $s(x)$ is a polyline. You are given n functions of the given type, your task is to find the number of angles that do not equal 180 degrees, in the graph $s(x)$, that is the sum of the given functions.
Valeric and Valerko really want to watch the next Euro Championship game, so they asked you to help them.

Input

The first line contains integer n ($1 \leq n \leq 10^5$) — the number of functions. Each of the following n lines contains two space-separated integer numbers k_i, b_i ($-10^9 \leq k_i, b_i \leq 10^9$) that determine the i-th function.

Output

Print a single number — the number of angles that do not equal 180 degrees in the graph of the polyline that equals the sum of the given functions.

Examples

input
1 1 0
output
1

input
3 1 0 0 2 -1 1
output
2

input
3 -2 -4 1 7 -5 1
output
3

E. Building Forest

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

An *oriented weighted forest* is an acyclic weighted digraph in which from each vertex at most one edge goes.

The *root* of vertex v of an oriented weighted forest is a vertex from which no edge goes and which can be reached from vertex v moving along the edges of the weighted oriented forest. We denote the root of vertex v as $root(v)$.

The *depth* of vertex v is the sum of weights of paths passing from the vertex v to its root. Let's denote the depth of the vertex v as $depth(v)$.

Let's consider the process of constructing a weighted directed forest. Initially, the forest does not contain vertices. Vertices are added sequentially one by one. Overall, there are n performed operations of adding. The i-th ($i > 0$) adding operation is described by a set of numbers (k, v_1, x_1, v_2, x_2, ... , v_k, x_k) and means that we should add vertex number i and k edges to the graph: an edge from vertex $root(v_1)$ to vertex i with weight $depth(v_1) + x_1$, an edge from vertex $root(v_2)$ to vertex i with weight $depth(v_2) + x_2$ and so on. If $k = 0$, then only vertex i is added to the graph, there are no added edges.

Your task is like this: given the operations of adding vertices, calculate the sum of the weights of all edges of the forest, resulting after the application of all defined operations, modulo 1000000007 ($10^9 + 7$).

Input

The first line contains a single integer n ($1 \le n \le 10^5$) — the number of operations of adding a vertex.

Next n lines contain descriptions of the operations, the i-th line contains the description of the operation of adding the i-th vertex in the following format: the first number of a line is an integer k ($0 \le k \le i - 1$), then follow $2k$ space-separated integers: $v_1, x_1, v_2, x_2, \dots, v_k, x_k$ ($1 \le v_j \le i - 1$, $|x_j| \le 10^9$).

The operations are given in the order, in which they should be applied to the graph. It is guaranteed that sum k of all operations does not exceed 10^5, also that applying operations of adding vertexes does not result in loops and multiple edges.

Output

Print a single number — the sum of weights of all edges of the resulting graph modulo 1000000007 ($10^9 + 7$).

Examples

input
6 0 0 1 2 1 2 1 5 2 2 1 1 2 1 3 4
output
30

input
5 0 1 1 5 0 0 2 3 1 4 3
output
9

Note

Conside the first sample:

1. Vertex 1 is added. $k = 0$, thus no edges are added.
2. Vertex 2 is added. $k = 0$, thus no edges are added.
3. Vertex 3 is added. $k = 1$. $v_1 = 2$, $x_1 = 1$. Edge from vertex $root(2) = 2$ to vertex 3 with weight $depth(2) + x_1 = 0 + 1 = 1$ is added.
4. Vertex 4 is added. $k = 2$. $v_1 = 1$, $x_1 = 5$. Edge from vertex $root(1) = 1$ to vertex 4 with weight $depth(1) + x_1 = 0 + 5 = 5$ is added. $v_2 = 2$, $x_2 = 2$. Edge from vertex $root(2) = 3$ to vertex 4 with weight $depth(2) + x_1 = 1 + 2 = 3$ is added.

5. Vertex 5 is added. $k = 1$. $v_1 = 1$, $x_1 = 2$. Edge from vertex $root(1) = 4$ to vertex 5 with weight $depth(1) + x_1 = 5 + 2 = 7$ is added.
6. Vertex 6 is added. $k = 1$. $v_1 = 3$, $x_1 = 4$. Edge from vertex $root(3) = 5$ to vertex 6 with weight $depth(3) + x_1 = 10 + 4 = 14$ is added.

The resulting graph is shown on the pictore below: