A. Function Height

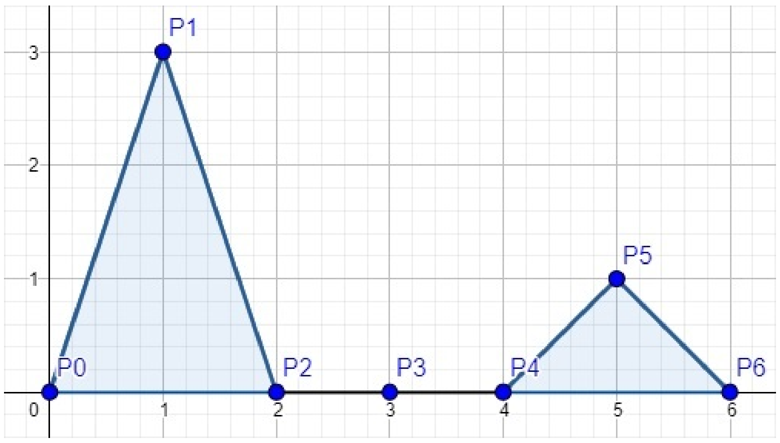
time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a set of $2n+1$ integer points on a Cartesian plane. Points are numbered from 0 to $2n$ inclusive. Let P_i be the i-th point. The x-coordinate of the point P_i equals i. The y-coordinate of the point P_i equals zero (initially). Thus, initially $P_i=(i,0)$.

The given points are vertices of a plot of a piecewise function. The j-th piece of the function is the segment P_jP_{j+1}.

In one move you can increase the y-coordinate of any point with odd x-coordinate (i.e. such points are $P_1, P_3, \dots, P_{2n-1}$) by 1. Note that the corresponding segments also change.

For example, the following plot shows a function for $n=3$ (i.e. number of points is $2 \cdot 3+1=7$) in which we increased the y-coordinate of the point P_1 three times and y-coordinate of the point P_5 one time:



Let the area of the plot be the area below this plot and above the coordinate axis OX. For example, the area of the plot on the picture above is 4 (the light blue area on the picture above is the area of the plot drawn on it).

Let the height of the plot be the maximum y-coordinate among all initial points in the plot (i.e. points $P_0, P_1, \dots, P_{2n}$). The height of the plot on the picture above is 3.

Your problem is to say which minimum possible height can have the plot consisting of $2n+1$ vertices and having an area equal to k. Note that it is unnecessary to minimize the number of moves.

It is easy to see that any answer which can be obtained by performing moves described above always exists and is an integer number not exceeding 10^{18}.

Input

The first line of the input contains two integers n and k ($1 \le n, k \le 10^{18}$) — the number of vertices in a plot of a piecewise function and the area we need to obtain.

Output

Print one integer — the minimum possible height of a plot consisting of $2n+1$ vertices and with an area equals k. It is easy to see that any answer which can be obtained by performing moves described above always exists and is an integer number not exceeding 10^{18}.

Examples

input
```
4 3
```
output
```
1
```

input
```
4 12
```
output
```
3
```

input

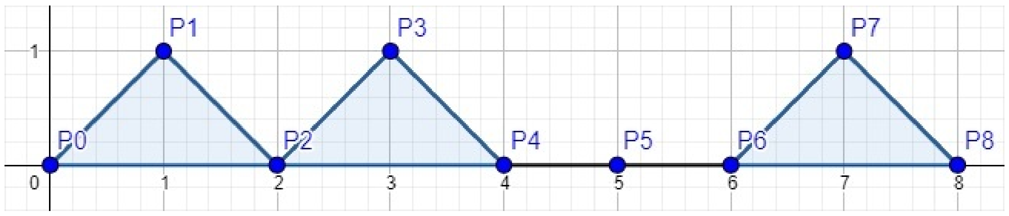
```
999999999999999999 999999999999999986
```

output

```
1
```

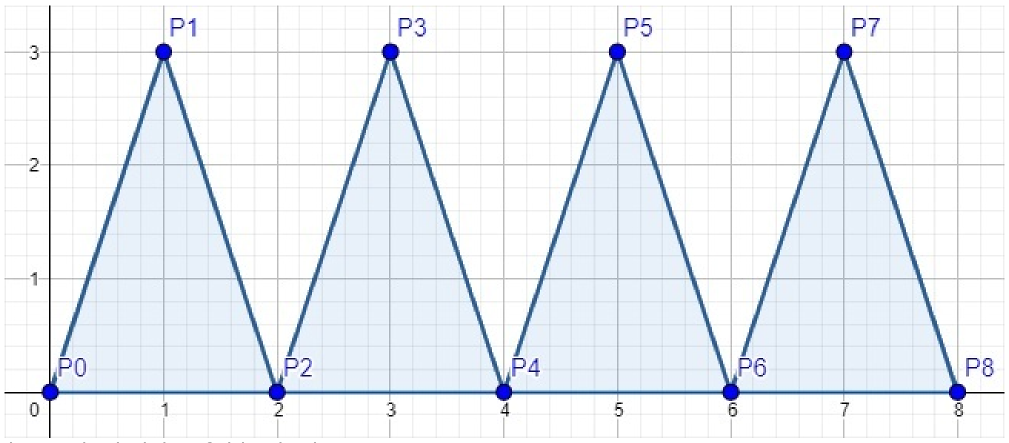
Note

One of the possible answers to the first example:



The area of this plot is 3, the height of this plot is 1.

There is only one possible answer to the second example:



The area of this plot is 12, the height of this plot is 3.

B. Diagonal Walking v.2

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

Mikhail walks on a Cartesian plane. He starts at the point $(0,0)$, and in one move he can go to any of eight adjacent points. For example, if Mikhail is currently at the point $(0,0)$, he can go to any of the following points in one move:

- $(1,0)$;
- $(1,1)$;
- $(0,1)$;
- $(-1,1)$;
- $(-1,0)$;
- $(-1,-1)$;
- $(0,-1)$;
- $(1,-1)$.

If Mikhail goes from the point $(x1,y1)$ to the point $(x2,y2)$ in one move, and $x1 \ne x2$ and $y1 \ne y2$, then such a move is called a *diagonal move*.

Mikhail has q **queries**. For the i-th query Mikhail's target is to go to the point (n_i,m_i) from the point $(0,0)$ in **exactly** k_i moves. Among all possible movements he want to choose one with the maximum number of *diagonal moves*. Your task is to find the maximum number of *diagonal moves* or find that it is impossible to go from the point $(0,0)$ to the point (n_i,m_i) in k_i moves.

Note that Mikhail **can** visit any point any number of times (even the destination point!).

Input

The first line of the input contains one integer q ($1 \le q \le 10^4$) — the number of queries.

Then q lines follow. The i-th of these q lines contains three integers n_i, m_i and k_i ($1 \le n_i, m_i, k_i \le 10^{18}$) — x-coordinate of the destination point of the query, y-coordinate of the destination point of the query and the number of moves in the query, correspondingly.

Output

Print q integers. The i-th integer should be equal to -1 if Mikhail cannot go from the point $(0,0)$ to the point (n_i,m_i) in **exactly** k_i moves described above. Otherwise the i-th integer should be equal to the the maximum number of *diagonal moves* among all possible movements.

Example

input

```
3
2 2 3
4 3 7
10 1 9
```

output

```
1
6
-1
```

Note

One of the possible answers to the first test case: $(0,0) \to (1,0) \to (1,1) \to (2,2)$.

One of the possible answers to the second test case: $(0,0) \to (0,1) \to (1,2) \to (0,3) \to (1,4) \to (2,3) \to (3,2) \to (4,3)$.

In the third test case Mikhail cannot reach the point $(10,1)$ in 9 moves.

C. Classy Numbers

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

Let's call some positive integer *classy* if its decimal representation contains no more than 3 non-zero digits. For example, numbers 4, 200000, 10203 are *classy* and numbers 4231, 102306, 7277420000 are not.

You are given a segment $[L; R]$. Count the number of *classy* integers x such that $L \le x \le R$.

Each testcase contains several segments, for each of them you are required to solve the problem separately.

Input

The first line contains a single integer T ($1 \le T \le 10^4$) — the number of segments in a testcase.

Each of the next T lines contains two integers L_i and R_i ($1 \le L_i \le R_i \le 10^{18}$).

Output

Print T lines — the i-th line should contain the number of *classy* integers on a segment $[L_i; R_i]$.

Example

input

```
4
1 1000
1024 1024
65536 65536
999999 1000001
```

output

```
1000
1
0
2
```

D. Vasya and Arrays

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

Vasya has two arrays A and B of lengths n and m, respectively.

He can perform the following operation arbitrary number of times (possibly zero): he takes some consecutive subsegment of the array and replaces it with a single element, equal to the sum of all elements on this subsegment. For example, from the array $[1, 10, 100, 1000, 10000]$ Vasya can obtain array $[1, 1110, 10000]$, and from array $[1, 2, 3]$ Vasya can obtain array $[6]$.

Two arrays A and B are considered equal if and only if they have the same length and for each valid i $A_i = B_i$.

Vasya wants to perform some of these operations on array A, some on array B, in such a way that arrays A and B become equal. Moreover, the lengths of the resulting arrays should be maximal possible.

Help Vasya to determine the maximum length of the arrays that he can achieve or output that it is impossible to make arrays A and B equal.

Input

The first line contains a single integer n $(1 \le n \le 3 \cdot 10^5)$ — the length of the first array.

The second line contains n integers $a_1, a_2, \cdots, a_n$ $(1 \le a_i \le 10^9)$ — elements of the array A.

The third line contains a single integer m $(1 \le m \le 3 \cdot 10^5)$ — the length of the second array.

The fourth line contains m integers $b_1, b_2, \cdots, b_m$ $(1 \le b_i \le 10^9)$ - elements of the array B.

Output

Print a single integer — the maximum length of the resulting arrays after some operations were performed on arrays A and B in such a way that they became equal.

If there is no way to make array equal, print "-1".

Examples

input

```
5
11 2 3 5 7
4
11 7 3 7
```

output

```
3
```

input

```
2
1 2
1
100
```

output

```
-1
```

input

```
3
1 2 3
3
1 2 3
```

output

```
3
```

E. Covered Points

time limit per test: 2 seconds

memory limit per test: 256 megabytes

input: standard input

output: standard output

You are given n segments on a Cartesian plane. Each segment's endpoints have integer coordinates. Segments can intersect with each other. No two segments lie on the same line.

Count the number of distinct points with **integer coordinates**, which are covered by at least one segment.

Input

The first line contains a single integer n $(1 \le n \le 1000)$ — the number of segments.

Each of the next n lines contains four integers Ax_i, Ay_i, Bx_i, By_i $(-10^6 \le Ax_i, Ay_i, Bx_i, By_i \le 10^6)$ — the coordinates of the endpoints A, B ($A \ne B$) of the i-th segment.

It is guaranteed that no two segments lie on the same line.

Output

Print a single integer — the number of distinct points with integer coordinates, which are covered by at least one segment.

Examples

input

```
9
0 0 4 4
-1 5 4 0
4 0 4 4
5 2 11 2
6 1 6 7
5 6 11 6
10 1 10 7
7 0 9 8
10 -1 11 -1
```

output

```
42
```

input

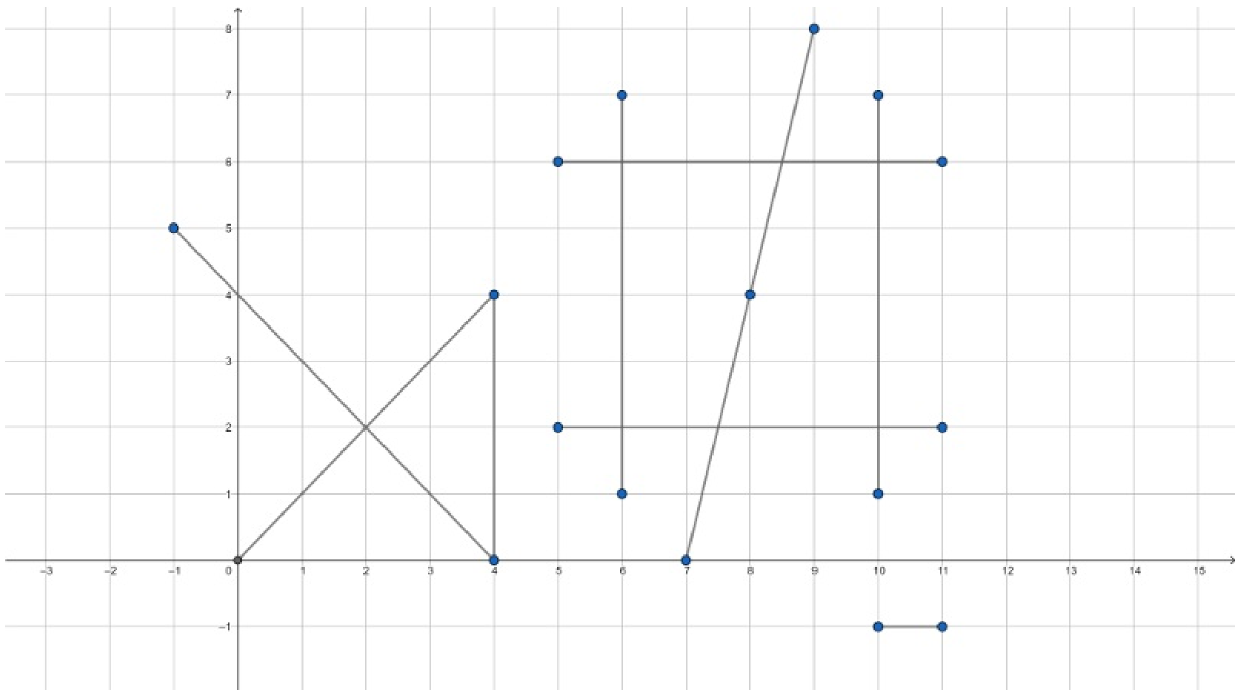
```
4
-1 2 1 2
-1 0 1 0
-1 0 0 3
0 3 1 0
```

output

```
7
```

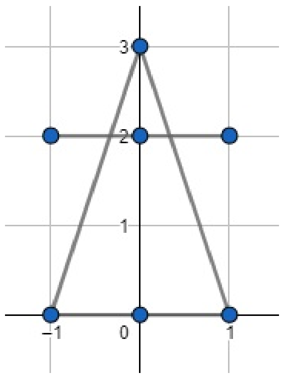
Note

The image for the first example:



Several key points are marked blue, the answer contains some non-marked points as well.

The image for the second example:



F. Relatively Prime Powers

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

Consider some positive integer x. Its prime factorization will be of form $x = 2^{k_1} \cdot 3^{k_2} \cdot 5^{k_3} \cdot \dots$

Let's call x *elegant* if the greatest common divisor of the sequence $k_1, k_2, \dots$ is equal to 1. For example, numbers $5 = 5^1$, $12 = 2^2 \cdot 3$, $72 = 2^3 \cdot 3^2$ are *elegant* and numbers $8 = 2^3$ ($GCD = 3$), $2500 = 2^2 \cdot 5^4$ ($GCD = 2$) are not.

Count the number of *elegant* integers from 2 to n.

Each testcase contains several values of n, for each of them you are required to solve the problem separately.

Input

The first line contains a single integer T ($1 \le T \le 10^5$) — the number of values of n in the testcase.

Each of the next T lines contains a single integer n_i ($2 \le n_i \le 10^{18}$).

Output
Print T lines — the i-th line should contain the number of *elegant* numbers from 2 to n_i.

Example
input
```
4
4
2
72
10
```
output
```
2
1
61
6
```

Note
Here is the list of **non-elegant** numbers up to 10:

- $4 = 2^2, GCD = 2$;
- $8 = 2^3, GCD = 3$;
- $9 = 3^2, GCD = 2$.

The rest have $GCD = 1$.

G. Sources and Sinks

time limit per test: 7 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

You are given an acyclic directed graph, consisting of n vertices and m edges. The graph contains no multiple edges and no self-loops.

The vertex is called a *source* if it has no incoming edges. The vertex is called a *sink* if it has no outgoing edges. These definitions imply that some vertices can be both source and sink.

The number of sources in the given graph is equal to the number of sinks in it, and each of these numbers doesn't exceed 20.

The following algorithm is applied to the graph:

1. if the graph has no sources and sinks then quit;
2. choose arbitrary source s, arbitrary sink t, add an edge from t to s to the graph and go to step 1 (that operation pops s out of sources and t out of sinks). Note that s and t may be the same vertex, then a self-loop is added.

At the end you check if the graph becomes strongly connected (that is, any vertex is reachable from any other vertex).

Your task is to check that the graph becomes strongly connected no matter the choice of sources and sinks on the second step of the algorithm.

Input
The first line contains two integers n and m ($1 \le n, m \le 10^6$) — the number of vertices and the number of edges in the graph, respectively.

Each of the next m lines contains two integers v_i, u_i ($1 \le v_i, u_i \le n$, $v_i \ne u_i$) — the description of the i-th edge of the original graph.

It is guaranteed that the number of sources and the number of sinks in the graph are the same and they don't exceed 20. It is guaranteed that the given graph contains no multiple edges. It is guaranteed that the graph contains no cycles.

Output
Print "YES" if the graph becomes strongly connected no matter the choice of sources and sinks on the second step of the algorithm. Otherwise print "NO".

Examples
input
```
3 1
1 2
```
output
```
NO
```
input

```
3 3
1 2
1 3
2 3
```

output

```
YES
```

input

```
4 4
1 2
1 3
4 2
4 3
```

output

```
YES
```