A. Perfect Permutation

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

A *permutation* is a sequence of integers $p_1, p_2, ..., p_n$, consisting of n distinct positive integers, each of them doesn't exceed n. Let's denote the i-th element of permutation p as p_i. We'll call number n the size of permutation $p_1, p_2, ..., p_n$.

Nickolas adores permutations. He likes some permutations more than the others. He calls such permutations perfect. A *perfect* permutation is such permutation p that for any i $(1 \le i \le n)$ (n is the permutation size) the following equations hold $p_{p_i} = i$ and $p_i \ne i$. Nickolas asks you to print any perfect permutation of size n for the given n.

Input

A single line contains a single integer n $(1 \le n \le 100)$ — the permutation size.

Output

If a perfect permutation of size n doesn't exist, print a single integer -1. Otherwise print n distinct integers from 1 to n, $p_1, p_2, ..., p_n$ — permutation p, that is perfect. Separate printed numbers by whitespaces.

Sample test(s)

input
```
1
```
output
```
-1
```

input
```
2
```
output
```
2 1
```

input
```
4
```
output
```
2 1 4 3
```

B. Non-square Equation

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

Let's consider equation:

$$x^2 + s(x) \cdot x - n = 0,$$

where x, n are positive integers, $s(x)$ is the function, equal to the sum of digits of number x in the decimal number system.

You are given an integer n, find the smallest positive integer root of equation x, or else determine that there are no such roots.

Input

A single line contains integer n $(1 \le n \le 10^{18})$ — the equation parameter.

Please, do not use the `%lld` specifier to read or write 64-bit integers in C++. It is preferred to use `cin`, `cout` streams or the `%I64d` specifier.

Output

Print -1, if the equation doesn't have integer positive roots. Otherwise print such smallest integer x $(x > 0)$, that the equation given in the statement holds.

Sample test(s)

input
```
2
```
output
```
1
```

input
```
110
```
output
```
10
```

input
```
4
```
output
```
-1
```

Note

In the first test case $x = 1$ is the minimum root. As $s(1) = 1$ and $1^2 + 1 \cdot 1 - 2 = 0$.

In the second test case $x = 10$ is the minimum root. As $s(10) = 1 + 0 = 1$ and $10^2 + 1 \cdot 10 - 110 = 0$.

In the third test case the equation has no roots.

C. Cycles

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

John Doe started thinking about graphs. After some thought he decided that he wants to paint an undirected graph, containing exactly k cycles of length 3.

A cycle of length 3 is an unordered group of three distinct graph vertices a, b and c, such that each pair of them is connected by a graph edge.

John has been painting for long, but he has not been a success. Help him find such graph. Note that the number of vertices there shouldn't exceed 100, or else John will have problems painting it.

Input

A single line contains an integer k $(1 \le k \le 10^5)$ — the number of cycles of length 3 in the required graph.

Output

In the first line print integer n $(3 \le n \le 100)$ — the number of vertices in the found graph. In each of next n lines print n characters "0" and "1": the i-th character of the j-th line should equal "0", if vertices i and j do not have an edge between them, otherwise it should equal "1". Note that as the required graph is undirected, the i-th character of the j-th line must equal the j-th character of the i-th line. The graph shouldn't contain self-loops, so the i-th character of the i-th line must equal "0" for all i.

Sample test(s)

input

```
1
```

output

```
3
011
101
110
```

input

```
10
```

output

```
5
01111
10111
11011
11101
11110
```

D. Table

time limit per test: 4 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

John Doe has an $n \times m$ table. John Doe can paint points in some table cells, not more than one point in one table cell. John Doe wants to use such operations to make each square subtable of size $n \times n$ have exactly k points.

John Doe wondered, how many distinct ways to fill the table with points are there, provided that the condition must hold. As this number can be rather large, John Doe asks to find its remainder after dividing by 1000000007 $(10^9 + 7)$.

You should assume that John always paints a point exactly in the center of some cell. Two ways to fill a table are considered distinct, if there exists a table cell, that has a point in one way and doesn't have it in the other.

Input

A single line contains space-separated integers n, m, k $(1 \le n \le 100;\ n \le m \le 10^{18};\ 0 \le k \le n^2)$ — the number of rows of the table, the number of columns of the table and the number of points each square must contain.

Please, do not use the `%lld` specifier to read or write 64-bit integers in C++. It is preferred to use the `cin`, `cout` streams or the `%I64d` specifier.

Output

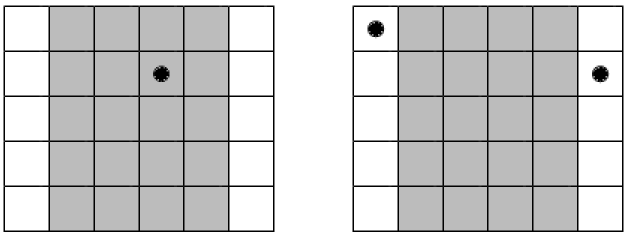
In a single line print a single integer — the remainder from dividing the described number of ways by 1000000007 $(10^9 + 7)$.

Sample test(s)

input
```
5 6 1
```
output
```
45
```

Note

Let's consider the first test case:



The gray area belongs to both 5×5 squares. So, if it has one point, then there shouldn't be points in any other place. If one of the white areas has a point, then the other one also must have a point. Thus, there are about 20 variants, where the point lies in the gray area and 25 variants, where each of the white areas contains a point. Overall there are 45 variants.

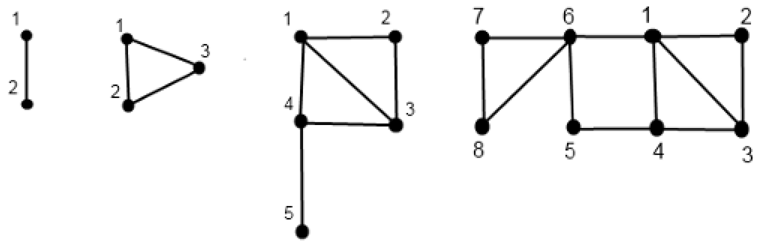
E. Doe Graphs

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

John Doe decided that some mathematical object must be named after him. So he invented the Doe graphs. The Doe graphs are a family of undirected graphs, each of them is characterized by a single non-negative number — its order.

We'll denote a graph of order k as $D(k)$, and we'll denote the number of vertices in the graph $D(k)$ as $|D(k)|$. Then let's define the Doe graphs as follows:

- $D(0)$ consists of a single vertex, that has number 1.
- $D(1)$ consists of two vertices with numbers 1 and 2, connected by an edge.
- $D(n)$ for $n \geq 2$ is obtained from graphs $D(n - 1)$ and $D(n - 2)$. $D(n - 1)$ and $D(n - 2)$ are joined in one graph, at that numbers of all vertices of graph $D(n - 2)$ increase by $|D(n - 1)|$ (for example, vertex number 1 of graph $D(n - 2)$ becomes vertex number $1 + |D(n - 1)|$). After that two edges are added to the graph: the first one goes between vertices with numbers $|D(n - 1)|$ and $|D(n - 1)| + 1$, the second one goes between vertices with numbers $|D(n - 1)| + 1$ and 1. Note that the definition of graph $D(n)$ implies, that $D(n)$ is a connected graph, its vertices are numbered from 1 to $|D(n)|$.



The picture shows the Doe graphs of order 1, 2, 3 and 4, from left to right.

John thinks that Doe graphs are that great because for them exists a polynomial algorithm for the search of Hamiltonian path. However, your task is to answer queries of finding the shortest-length path between the vertices a_i and b_i in the graph $D(n)$.

A path between a pair of vertices u and v in the graph is a sequence of vertices $x_1, x_2, \ldots, x_k$ $(k > 1)$ such, that $x_1 = u$, $x_k = v$, and for any i $(i < k)$ vertices x_i and x_{i+1} are connected by a graph edge. The length of path $x_1, x_2, \ldots, x_k$ is number $(k - 1)$.

Input

The first line contains two integers t and n $(1 \leq t \leq 10^5;\ 1 \leq n \leq 10^3)$ — the number of queries and the order of the given graph. The i-th of the next t lines contains two integers a_i and b_i $(1 \leq a_i, b_i \leq 10^{16},\ a_i \neq b_i)$ — numbers of two vertices in the i-th query. It is guaranteed that $a_i, b_i \leq |D(n)|$.

Please, do not use the `%lld` specifier to read or write 64-bit integers in C++. It is preferred to use `cin`, `cout` streams or the `%I64d` specifier.

Output

For each query print a single integer on a single line — the length of the shortest path between vertices a_i and b_i. Print the answers to the queries in the order, in which the queries are given in the input.

Sample test(s)

input

```
10 5
1 2
1 3
1 4
1 5
2 3
2 4
2 5
3 4
3 5
4 5
```

output

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