Problem A. Array Factory

Input file:	`arrayfactory.in`
Output file:	`arrayfactory.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

Vasily works on a factory that produces arrays. The factory produces only integer arrays of length n. Recently, company salesmen found out that customers prefer to buy arrays with sum of elements not exceeding s. Naturally, the decision was made to sell only such arrays from now on. The renovation turned out to be too expensive, so a simpler solution was found: one could cut some elements from the beginning and the end of an array so that the sum of the remaining elements would be appealing to customers.

This is the essence of Vasily's job. Moreover, the customers pay more for longer arrays, so Vasily should cut minimal possible amount of elements so the company would not suffer losses. The job is very boring, so Vasily needs your help.

Input

The first line of input contains two integers n and s ($1 \le n \le 200\,000$, $-10^{18} \le s \le 10^{18}$).

The second line contains n integers a_1, a_2, $\ldots$, a_n: the elements of the array ($-10^9 \le a_i \le 10^9$).

Output

If each nonempty subsegment of the array has sum of elements strictly more than s, print -1.

Otherwise, on the first line, print one integer: the length of the array after cutting. On the second line, print two integers: the amounts of elements cut from the beginning and from the end of the array, respectively.

If there are several possible answers, print the one with the minimum possible number of elements cut from the beginning.

Example

arrayfactory.in	arrayfactory.out
5 3 2 4 -2 1 1	3 1 1

Problem B. Purchases and Bonuses

Input file:	`bonuses.in`
Output file:	`bonuses.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

In the Aweson internet shop, the following payment rules apply. All purchases are paid by thalers, the internal Aweson currency. When a new customer is registered in the system, he gets a bonus account, initially with 0 thalers. At the moment of each purchase, a customer has a choice: either to use available bonuses or to accumulate them. When using available bonuses, a customer may pay for any part of the purchase with bonus thalers, and the remaining part of the cost, if any, is paid with regular thalers. When accumulating bonuses, a customer may not use existing bonus thalers, but exactly 1% of the cost is added to the bonus account.

Mihail made a plan of purchases on Aweson. Now he wants to know which choice to make for each purchase and how to pay for them so that the total number of regular thalers he spends is minimum possible.

Input

The first line of input contains an integer n: the number of purchases ($1 \le n \le 100$). The second line contains n integers $p_1, p_2, \ldots, p_n$ separated by single spaces: the cost of the first, second, …, n-th purchase in thalers ($0 < p_i \le 10^5$, all p_i divide evenly by 100).

Output

On the first line, print one integer s: the minimum total amount of regular thalers with which Mihail can make all his purchases in the given order.

On the next n lines, print a strategy which allows to spend exactly s regular thalers. Each of these lines must contain two integers c_k and b_k separated by a «plus» sign surrounded with single spaces: how much regular thalers and how much bonus thalers Mihail should spend on k-th purchase. Obviously, these numbers must be nonnegative, and their sum must be equal to the cost of k-th purchase. If $b_k = 0$, then k-th purchase is accumulating bonuses, and if $b_k > 0$, it is using bonuses. Naturally, during each purchase, Mihail can not spend any more bonuses than he currently has, and the sum of all c_k must be exactly s.

The number of bonus thalers after all n purchases is irrelevant. If there are several strategies with the minimum possible s, print any one of them.

Examples

bonuses.in	bonuses.out

`2`
`100 100`
	`199`
	`100 + 0`
	`99 + 1`

`3`
`100 10000 100`
	`10100`
	`100 + 0`
	`10000 + 0`
	`0 + 100`

Explanations

In the first example, Mihail gets one bonus thaler from the first purchase. Then we spend it to pay for a part of the second purchase.

In the second example, Mihail also gets one bonus thaler from the first purchase. However, it is not profitable to spend it on the second purchase. Instead, Mihail can get 100 more thalers from the second purchase. After that, he can fully pay for the third purchase with bonus thalers, and even have one bonus thaler left.

Problem C. Number of Solutions

Input file:	`c2sat.in`
Output file:	`c2sat.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

You are given a boolean expression F of n variables $x_1, x_2, \ldots, x_n$ which has the form $\bigwedge\limits_{i=1}^{m} a_i \vee b_i$, where each of a_i and b_i is equal either to one of the variables or to negation of one of the variables. Each x_j can be either true or false.

Your task is to count the number of satisfying assignments for F. More precisely, you have to find the number of ways to choose values of all x_j such that, for all i, at least one of a_i and b_i is true.

See example explanation for better understanding.

Input

The first line of input contains two integers n and m ($1 \le n \le 50$). Each of the next m lines contains two integers a_i and b_i ($1 \le |a_i|, |b_i| \le n$). A positive value k from 1 to n means variable x_k, and a negative value k from -1 to $-n$ means a negation $\neg x_{|k|}$.

It is guaranteed that, for all i, $a_i < b_i$. Additionally, all pairs (a_i, b_i) are distinct.

Output

Print one integer: the number of satisfying assignments for F.

Example

c2sat.in	c2sat.out

3 2
-2 1
2 3
	4

Explanation

In the example, the expression is $F(x_1, x_2, x_3) = (\neg x_2 \vee x_1) \wedge (x_2 \vee x_3)$. The following table shows the values of this expression and its parts for all possible values of variables. There are four ways to choose values so that F is true.

x_1	x_2	x_3	$\neg x_2 \vee x_1$	$x_2 \vee x_3$	F
0	0	0	1	0	0
0	0	1	1	1	1
0	1	0	0	1	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

Problem D. Cutting Potatoes

Input file:	`cut-potatoes.in`
Output file:	`cut-potatoes.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

Petr wants to boil potatoes. He has n potatoes of volumes a_1, a_2, …, a_n, and he wants to boil all of them.

Petr believes that potatoes are best boiled when they have roughly the same volume. To achieve that, he agrees to cut some potatoes into parts, but he refuses to cut any potato into more than k parts.

Petr invented the following formal model. Suppose we place some potatoes, whole or in parts, in a pot. Let the *unevenness* of this set of objects be the ratio of the largest of their volumes to the smallest one. Help Petr choose the number of equal parts into which to cut each of his potatoes so that the unevenness of the resulting set is as small as possible.

Input

The first line of input contains two integers n and k: the number of potatoes and the maximum possible number of parts into which a potato can be cut ($1 \le n, k \le 100$). The second line contains n integers a_1, a_2, …, a_n separated by single spaces: the volumes of the potatoes ($1 \le a_i \le 100$).

Output

Print a line consisting of n integers c_1, c_2, …, c_n separated by single spaces: the number of equal parts the potatoes 1, 2, …, n should be cut into so that the unevenness of the resulting set is as small as possible. Each c_i must be from 1 to k inclusive. Keep in mind that a potato can be cut into parts with non-integer volume. If there are several possible answers with minimum unevenness, print any one of them.

Examples

cut-potatoes.in	cut-potatoes.out
4 3 5 5 3 2	3 3 2 1
2 2 1 4	1 2

Explanations

In the first example, the potatoes have volumes 5, 5, 3 and 2. Petr can cut each of them into one, two, or three parts. The optimal solution is to get six parts of volume 5/3, two parts of volume 3/2 and one part of volume 2/1. The unevenness is $(2/1)/(3/2) = 4/3$.

In the second example, Petr should cut the large potato into two parts, and leave the small potato whole. The unevenness is 2/1.

Problem E. Divide and Conquer

Input file:	`dnc.in`
Output file:	`dnc.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

Victoria is studying the "divide and conquer" method. She takes a polygon, not a random one but a convex polygon, which additionally has to have an even number of vertices. Victoria knows that, in order to "divide and conquer", one must first choose how to divide. Therefore, she is implementing the following recursive function `Fun` which receives a subset S of polygon vertices as argument (initially, S is the whole set of vertices):

- From the set S, choose two vertices A and B so that $(x_A, y_A) < (x_B, y_B)$ (first we compare x-coordinates, and if they are equal, y-coordinates).
- After that, divide the set S into two parts by the segment between these two vertices: the "left" part will contain all points (x, y) such that $(x_B - x_A) \cdot (y - y_A) > (x - x_A) \cdot (y_B - y_A)$, and the "right" part will contain all points such that $(x_B - x_A) \cdot (y - y_A) < (x - x_A) \cdot (y_B - y_A)$.
- Furthermore, run the function `Fun` recursively for the "left" part, and when it is finished, run `Fun` recursively for the "right" part.
- Finally, there must be a part of the function where the results of the two recursive calls are combined to produce the result of the current call of `Fun`. But for now, Victoria decided to omit this part for simplicity and focus on division.
- Surely, one has to pay attention to corner cases. If `Fun` is run on an empty set of vertices S, it does nothing and exits immediately. As Victoria does not want to handle the case of a single vertex, she decided that each division must produce only even parts.

Victoria decided to look at different division strategies. To do this, as soon as a call to `Fun` chooses the two vertices A and B, Victoria appends their coordinates to the strategy log. In the end, the log contains all vertices in some order. This log is how a strategy is written down.

Imagine that all possible strategy logs for the given polygon are written down and ordered as sequences of numbers. Help Victoria find the strategy which has number k in this order. The strategies are numbered by integers, starting from zero.

Input

The first line of input contains an integer n: the number of polygon vertices ($4 \le n \le 30$, the number n is even).

Each of the next n lines contains two integers x and y: the coordinates of a vertex ($|x|, |y| \le 10\,000$). Vertices are listed in counter-clockwise order. It is guaranteed that no two vertices coincide and no three vertices lie on the same line, and that the given polygon is convex.

Finally, the next line contains an integer k: the number of the strategy Victoria wants to see. The strategies are numbered by integers, starting from zero. It is guaranteed that the strategy with the required number exists.

Output

Print the coordinates of n vertices in the order they follow in the list corresponding to k-th strategy. The coordinates of each vertex must be printed on a separate line.

Examples

dnc.in	dnc.out

4
0 0
1 0
1 1
0 1
3
	1 0
	1 1
	0 0
	0 1

6
0 0
1 -1
2 0
2 1
1 2
0 1
8
	0 0
	2 1
	0 1
	1 2
	1 -1
	2 0

Explanations

In the first example, there are only four strategies, and they have numbers 0, 1, 2 and 3:
$(0,0)$, $(0,1)$; $(1,0)$, $(1,1)$;
$(0,0)$, $(1,0)$; $(0,1)$, $(1,1)$;
$(0,1)$, $(1,1)$; $(0,0)$, $(1,0)$;
$(1,0)$, $(1,1)$; $(0,0)$, $(0,1)$.

In the second example, the strategy with number 8 starts by using the segment $(0,0)$–$(2,1)$ to divide the vertices of the polygon into two subsets: the "left" subset contains $(0,1)$ and $(1,2)$, while the "right" one is composed of $(1,-1)$ and $(2,0)$. In this order, all remaining vertices appear in the strategy log.

Problem F. Doubling

Input file:	`doubling.in`
Output file:	`doubling.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

A program in the "Doubling" language is a string consisting of characters "`1`", "`[`" and "`]`". The result of executing such a program is an integer. Here are the rules of construction and result calculation for this language:

- $R(\varepsilon) = 0$: the result of an empty program is zero.
- $R(\texttt{1}) = 1$: the program "`1`" gives the result 1.
- $R(\texttt{[}A\texttt{]}) = 2 \cdot R(A)$: square brackets double the result of the program inside them.
- $R(AB) = R(A) + R(B)$: the result of concatenation of two programs is the sum of their individual results.

It can be shown that the above recursive definition uniquely defines the set of possible programs and the results of their execution. A program which can not be constructed by the above rules is invalid, and the result of executing such a program is not defined.

You are given an integer n from one to one billion. Print the shortest possible program which gives result n. In case of several possible answers, print any one of them.

Input

The first line of input contains an integer n: the required number ($1 \le n \le 10^9$).

Output

On the first line, print the shortest program which gives result n. If there are several such programs, print any one of them.

Examples

doubling.in	doubling.out
`1`	`1`
`10`	`[11111]`

Explanations

In the first example, the program "`1`" gives the result 1.

In the second example, the program "`11111`" gives the result 5, and thus the program "`[11111]`" will give 10, the required result. There are also other possible answers: "`[[11]1]`" and "`[1[11]]`".

Problem G. New Collection

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	256 mebibytes

This is an interactive problem.

A new expansion set was just released for CCG, an online collectible card game. Players may purchase random cards of the expansion one by one: each purchased card will, with uniform probability and independently of other purchases, turn out to be one of the cards from the expansion set.

Petr is one of the main authors of CCG Scene, the players' site which collects known data about all CCG cards, along with their advantages, disadvantages, usage strategies and much more.

One of the important parameters of an expansion is the total number of cards in it. To maintain his reputation in the community, Petr wants to deliver this information as soon as possible. It is known that, in each of the previous expansions, the number of cards was an integer power of 10: from ten to ten million inclusive. Petr assumed that this rule holds for the new expansion as well, and that the number of cards can be any of the possible powers of ten with equal probability.

Petr has enough resources to purchase as much as 10 000 cards. Each CCG card has an identifier, but it is only known that the identifiers are the same for equal cards and different for distinct cards. Petr can purchase cards one by one, and after each purchase, he learns the identifier of the card he just bought. Moreover, at any moment, Petr can stop the purchases and write the number of cards in the new expansion at CCG Scene site.

Write a program which will help Petr find the right answer.

Interaction Protocol

Your solution must print each action on a separate line to the standard output stream. Each action must have either the form "`+`" for purchasing the next card or the form "`=` n" for writing the answer.

After each purchase, a new line appears at the standard input stream: the identifier of the purchased card. Each identifier consists of exactly twelve hexadecimal digits (`0`–`9` and `A`–`F`) and may contain leading zeroes.

After writing the answer, the solution must immediately terminate gracefully.

To prevent output buffering, flush the output buffer after each request: this can be done by using, for example, `fflush (stdout)` in C or C++, `System.out.flush ()` in Java, `flush (output)` in Pascal or `sys.stdout.flush ()` in Python.

In each test for this problem, the number of cards n is fixed to be some integer power of 10 from 10^1 to 10^7 inclusive. It is guaranteed that each purchased card will, with uniform probability and independently of other purchases, turn out to be one of the n possible cards. Nevertheless, modeling this assumption, in each test, the identifiers of the first 10 000 purchases are fixed in advance.

Example

participant's actions	checking program's answers
+	13A17DA944F1
+	0BE186EC4732
+	1591FAA834F2
+	13A17DA944F1
+	13A17DA944F1
+	0E703FE27659
+	1A16F6A9EBB6
+	1591FAA834F2
+	0E703FE27659
+	07B569E7D7A4
+	13A17DA944F1
= 10	

Explanation

Please note that the participant's **output** is displayed to the **left**, and the following **input** to the **right**.

The example illustrates the first test in the testing system. The number of cards in the expansion in this test is 10, and their identifiers are:
`07B569E7D7A4`,
`0A2001E19A1F`,
`0BE186EC4732`,
`0E703FE27659`,
`121657A5CA39`,
`13786BA38233`,
`13A17DA944F1`,
`1591FAA834F2`,
`189A2AAE360C`,
`1A16F6A9EBB6`.

The solution starts with purchasing eleven cards one by one. In the process, three cards appeared more than once, and four cards did not appear at all. After that, the solution decides that the obtained information is enough to determine that there are only 10 cards, and writes the answer.

Problem H. Path or Coloring

Input file:	`pathorcoloring.in`
Output file:	`pathorcoloring.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

Do you like NP-hard problems? For example, let us consider the following two:

Coloring problem. You are given an undirected graph G and an integer k. For each vertex of the graph, choose color $\phi(v)$ so that all colors are integers between 1 and k, and any two vertices connected by an edge have different color.

Simple k-path problem. You are given an undirected graph G and an integer k. Find a simple path of length k. Formally, you need to choose such sequence of $k+1$ *unique* vertices $v_1, v_2, \ldots, v_{k+1}$ that any two consecutive verices are connected by an edge.

You are given an undirected graph G and an integer k. Solve any one of these two problems for them. The choice is yours.

Input

The first line of input contains an integer T, the number of test cases ($1 \le T \le 1000$). The description of each test case consists of several lines.

The first line of each description of containts two integers n and m: the number of vertices and the number of edges in the graph ($1 \le n \le 1000$, $1 \le m \le 10\,000$). On the next line, the integer k is given ($1 \le k \le n$). Each of the next m lines contains two integers a and b: the numbers of two verices connected by an edge ($1 \le a, b \le n$).

It is guaranteed that the given graphs have no self-loops and no multiple edges. Additionally, the total number of edges in all given graphs does not exceed 100 000.

Output

For each test case, print one of the following:

- The word "`path`" followed by $k+1$ numbers of vertices: a simple path of length k.
- The word "`coloring`" followed by n integers from 1 to k: a coloring of the graph into k colors.
- The word "`neither`" if there is no simple path of length k and no coloring into k colors.

If there exist both a valid path and a valid coloring, you can print either a path or a coloring. If there are several valid paths or several valid colorings, you can print any one path or any one coloring, respectively.

Example

pathorcoloring.in	pathorcoloring.out

2
4 5
2
1 2
2 3
3 4
4 1
1 3
3 3
3
1 2
2 3
3 1
	path 3 2 1
	coloring 1 2 3

Problem I. Shuffle Again

Input file:	`shuffle-again.in`
Output file:	`shuffle-again.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

Consider the following linear congruential pseudo-random number generator. The generator's current state is denoted by `state` which can be any integer from 0 to $2^{32} - 1$ inclusive. To generate the next random integer from 0 to `range` $- 1$ inclusive, the `random` function is used:

```
Function random (range):
    state := (state * 1664525 + 1013904223) mod $2^{32}$;
    return (state * range) div $2^{32}$;
```

Here, `mod` and `div` are the operations of taking the remainder and integer division. The constants are the ones recommended by the Numerical Recipes textbook. For example, if at some moment `state` is equal to 12 345 and `random (100)` is called, first, `state` becomes equal to 87 628 868, and after that, the function returns the value 2.

To randomly shuffle an array, the `shuffle` function is used:

```
Function shuffle (array):
    n := length of array;
    for i := 0, 1, 2, ..., n - 1:
        j := random (i + 1);
        array[i] <-> array[j];
    return array;
```

Here, operation "`x <-> y`" swaps elements `x` and `y`; if they are one and the same element of the array, nothing happens. The array is indexed by integers from 0 to $n - 1$ inclusive. For example, if at some moment `state` is equal to 12 345 and `shuffle` is called with array `[1, 2, 3, 4, 5]`, the final value of `state` will be 3 908 547 000, and the function will return the array `[2, 3, 4, 1, 5]`.

Fyodor has chosen an integer `seed` from 0 to $2^{32} - 1$ inclusive. After that, he ran the following program:

```
state := seed;
n := 10000;
array := [1, 2, ..., n];
array := shuffle (array);
```

Fyodor keeps the value of `seed` to himself, but revealed the contents of `array` after running this program. Find out what will be the contents of `array` if he appends the following line at the end of his program:

```
array := shuffle (array);
```

Input

The first line of input contains an integer n: the number of elements in the permutation (in this problem, it is always equal to 10 000). The second line contains a permutation of n integers from 1 to n separated by spaces: the contents of the array after running Fyodor's program.

Output

On the first line, print a permutation of n integers from 1 to n separated by spaces: the contents of the array after running the program with the appended line.

Example

shuffle-again.in	shuffle-again.out

`10000`
`7236 5734 5374 ... 796 5348`
	`8150 6681 695 ... 5947 4461`

Explanation

In the example, the `seed` value is 12 345. Full example input and output can be downloaded at: `http://acm.math.spbu.ru/161009/shuffle-again/`.

Problem J. Timer

Input file:	`timer.in`
Output file:	`timer.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

Consider the following construction of a timer. The timer consists of two cylinders: the inner cylinder and the outer cylinder rotating around it. The surface of the inner cylinder is covered with marks and numbers. The marks correspond to minutes. The outer cylinder covers every part of the inner one except a small window. The time displayed by the timer can be determined by looking at the center of the window.

In total, there are 60 marks around the inner cylinder, and each fifth is denoted by the corresponding number: 0, 5, 10, 15, . . ., 55. All distances between two consecutive marks, including the distance between the last and the first marks, are the same. The width of the window is exactly five times greater than the distance between consecutive marks.

Initially, the center of the window in the outer cylinder coincides with mark 0 on the inner cylinder. The timer is set by rotating the outer cylinder clockwise around the inner one; after that, it slowly starts rotating counter-clockwise until it reaches mark 0 again, and then stops. The timer can be set to display any time strictly less than 60 minutes.

Anna is teaching her pet robot Berta to tell the time displayed on the timer. Berta can take a photo of the timer's window. Help Anna write a program for Berta to determine the time displayed on such a photo.

A photo of the window is a raster image 60 pixels in width and 12 pixels in height. For simplicity, we assume that the photo is a perfect image of flattened inner cylinder's surface seen through the window. The distance between consecutive marks is therefore 12 pixels. Another simplification is that the photo is taken when the time displayed is an exact multiple of 5 seconds. Because the window has a non-rectangular shape, small areas at the corners are always covered by the outer cylinder; please refer to the example below to see the exact shape of corners. All other pixels of the photo are either completely white or completely black.

Under these assumptions, each mark is represented by a 2×2 black square in the first two rows of the photo; the real center of the mark is between the two columns of this square. Each number is printed in black in a 8×8 font in rows 4 through 11. Each number (one or two digits long) is horizontally aligned so that the center of its 8×8 or 16×8 representation corresponds to the center of the mark it denotes. All pixels not covered by corners, marks or digits are white.

The shapes of individual digits from 0 to 9 as seen in the 8×8 font are shown below. The font itself is identical to the one stored on *Rendition Vérité 1000* graphics cards back in the 1990s. The shapes of digits can also be downloaded at the following address:
`http://acm.math.spbu.ru/161009/timer/`.

```
.XXXXX.. ...XX... .XXXXX.. .XXXXX.. ...XXX.. XXXXXXX. .XXXXX.. XXXXXXX. .XXXXX.. .XXXXX..
XX..XXX. ..XXX... XX...XX. XX...XX. ..XXXX.. XX...... XX...XX. XX...XX. XX...XX. XX...XX.
XX.XXXX. .XXXX... .....XX. .....XX. .XX.XX.. XXXXXX.. XX...... X....XX. XX...XX. XX...XX.
XXXX.XX. ...XX... ...XXX.. ..XXXX.. XX..XX.. .....XX. XXXXXX.. ....XX.. .XXXXX.. .XXXXXX.
XXX..XX. ...XX... .XXX.... .....XX. XXXXXXX. .....XX. XX...XX. ...XX... XX...XX. .....XX.
XXX..XX. ...XX... XX...... XX...XX. ....XX.. XX...XX. XX...XX. ..XX.... XX...XX. XX...XX.
.XXXXX.. .XXXXXX. XXXXXXX. .XXXXX.. ...XXXX. .XXXXX.. .XXXXX.. ..XX.... .XXXXX.. .XXXXX..
........ ........ ........ ........ ........ ........ ........ ........ ........ ........
```

Input

The input consists of exactly 12 lines, each of which contains exactly 60 characters: the photo of the timer's window. Because the window has a non-rectangular shape, small areas at the corners are always covered by "`-`" (minus signs); please refer to the example below to see the exact shape of corners. All other characters correspond to the inner cylinder and are either "`.`" (dot) for a white pixel or "`X`" (large letter ex) for a black pixel.

It is guaranteed that the timer correctly displays some time which is strictly less than 60 minutes and is an exact multiple of 5 seconds.

Output

Print the displayed time on a single line, formatted as "`MM:SS`". Here, "`MM`" are two digits representing minutes and "`SS`" are two digits representing seconds.

Example

timer.in

```
-------X..........XX..........XX..........XX.........-------
----..XX..........XX..........XX..........XX..........XX----
-..........................................................-
X..................................................XXXXXXX..
XX.................................................XX.......
XX.................................................XXXXXX...
XX......................................................XX..
XX......................................................XX..
XX.................................................XX...XX..
-...................................................XXXXX..-
----....................................................----
-------..............................................-------
```

timer.out

```
07:05
```

Explanation

In the example, the center of the window is between marks `10` and `5`. The mark denoted by `5` is clearly visible to the right. A part of the zero in `10` can be seen to the left. Thorough examination tells that the exact displayed time is 7 minutes and 5 seconds.

Problem K. Ultraprime Numbers

Input file:	`ultraprime.in`
Output file:	`ultraprime.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

As you may know, a *prime* is a positive integer number which has exactly two positive integer divisors: the unity and the number itself. Let us call a number an *ultraprime* if we can erase $x \geq 0$ digits from the front and $y \geq 0$ digits from the back of its decimal notation, and regardless of x and y, the remaining integer — that is, if there are any digits left — is a prime.

For example, the number 37 is ultraprime because the numbers 3, 7 and 37 which result in erasing digits from the front and from the back are prime. On the other hand, the number 5867 is not an ultraprime because, for example, the number 86 is not a prime.

Given an index n, print the n-th ultraprime in ascending order.

Input

The first line of input contains an integer n: the number of the requested ultraprime ($1 \leq n \leq 1000$).

Output

On the first line, print the n-th ultraprime in ascending order or -1 if there is no ultraprime with such index.

Examples

ultraprime.in	ultraprime.out
2	3
6	37

Explanations

The sequence of ultraprimes starts as follows: 2, 3, 5, 7, 23, 37, ...